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 embeddding 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 Noguera *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].

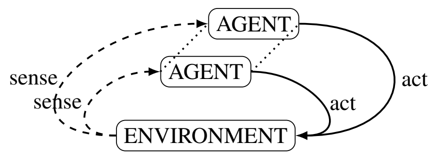


Figure 1: The situated (multi)agent system...

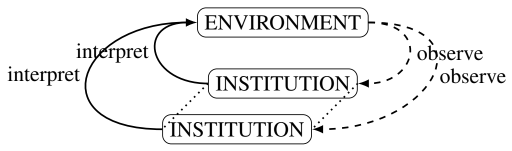
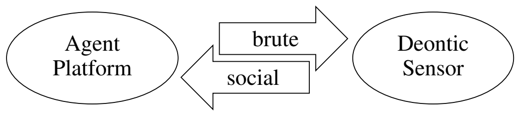


Figure 2: ...and its (multi)institutional extension

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:



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.

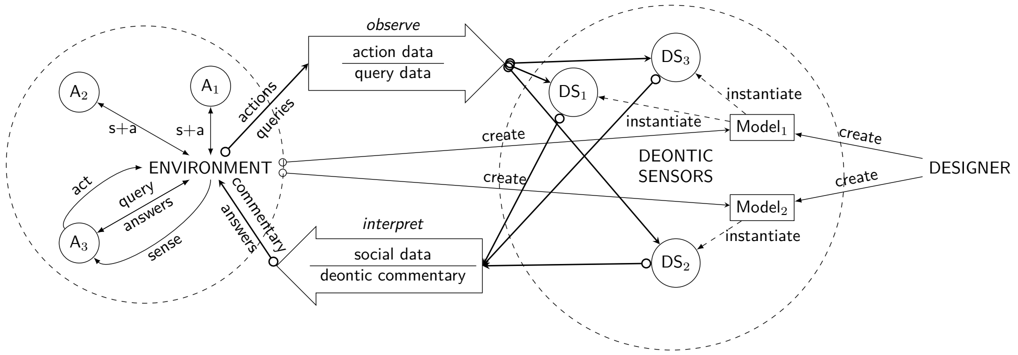


Figure 3: The deontic sensor architecture: agents sense and act via the environment on the left. Actions and queries are propagated from the environment to the deontic sensor platform (US8, US9). Their interpretation according to the instances of the normative models is then fed back to the environment (US10) as percepts for the agents to sense (US11). New models are created by a designer uploading them (US1), although this operation could be used by agent systems as well, then instantiated (US2, US5, US7) to satisfy different users. The institution designer can interact with the DSP via a software client to achieve US3 and US4 (as done for the evaluation in section 5)

Abstract (pattern) endpoints (AEP1-4)	**Concrete (implementation) endpoints (CEP1-4)**
`POST /model/` Creates a model from a specification in a framework-specific format. This is how a client sends a normative specification to a normative reasoning service. Returns a URI `/model/`X.	POST/model/ Creates a new model using the provided set of InstAL/AnsProlog specifications.
`POST /model/`X`/instance/` Creates an instance of model X with data from `POST`. This is how a client sends any necessary additional data to a normative reasoning service. Returns a URI `/model/`X`/instance/`Y.	`POST /model/`X`/instance/` Creates a new grounding of model X, using a provided set of literal data. This allows for grounding a model in more than one way without duplicating files or repeat groundings, but using it for multiple queries.
`POST /model/`X`/instance/`Y`/query/` Creates a query of instance Y, with `POST` data (e.g. events or normative framework specific input). Returns a URI `/model/`X`/instantiate/`Y`/query/`Z.	`POST /model/`X`/instance/`Y`/query/` Creates a query of grounding Y, given a set of query events and/or other parameters. Through this endpoint, queries on the same grounding can be accessed and analysed together.
`GET /model/`X`/instance/`Y`/query/`Z`/output` Reads the result of query Z in a protocol-defined format. In practice, this is defined by the HTTP Accept header, but also depends on what the actual platform supports.	`GET /model/`X`/instance/`Y`/query/`Z`/output` Reads the result of query Z and returns it in a format defined by the HTTP Accept header. The current options for InstAL output are `text` and JSON.

Table 1: The three **C**reate and one **R**ead operations of the abstract and concrete deontic sensor ROA

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 **C**reate, **R**ead, **U**pdate and **D**elete (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. A_3 in Figure 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 commentary, it needs data about the events that occur in the environment through the actions of agents. This corresponds to the *observe* arc in Fig. 2. At the conceptual level, this is the institutional 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 action 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 operators 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 environment. This corresponds to the *interpret* arc in Fig. 2. We refer to the ongoing normative information being provided to agents in the form of percepts as a *deontic commentary*. Following deontic logic, there are three kinds of percepts an agent can receive from an institution, regarding an action A:

$perm(A)$: Action A is permitted, written PA in deontic logic. The agent may perform action A.

$pro(A)$: Action A is prohibited (forbidden), written $\neg PA$ (also FA). The agent may not perform action A.

$obl(A, [D, V])$: The agent is obliged to perform action A (possibly before deadline D or else violation V ensues), 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 conventional environmental percepts and the same applies to normative percepts. Thus, receiving a percept $pro(action)$ does not mean that an agent is *unable* to perform $action$, but – assuming 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 normative 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 institutional 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 platform. 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 [Bordini *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 produces 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 normative interpretations of the action(s)[2].

InstAL-REST[3] provides InstAL as a service with a RESTful interface: this supports the deployment of multiple institutional 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 Celery), 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 construct 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 section 3.1 and detailed in Table 1.

The Jason platform provides both the agent and environment 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 receives 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/

```
1  Percepts for Alice:
2  [pro(action)[a,inst], pro(action)[b,inst]]
3  [alice] Applying for permit A
4  Environment: Permit A acquired.

5  Percepts for Alice:
6  [perm(action)[a,inst], pro(action)[b,inst]]
7  [alice] Applying for permit B
8  Environment: Permit B acquired.

9  Percepts for Alice:
10 [perm(action)[a,inst], perm(action)[b,inst]]
11 [alice] Doing action!
12 Environment: Action performed.

13 [alice] Complete!
```

Listing 1: Alice's normative facts

```
!doAction. // Initial goal
// If permitted to do action, do action.
+!doAction : perm(action) & not pro(action)
<-.print("Doing␣action!"); action; .print("Complete!").
//
// If prohibited to do action by a,
// apply for permit a, and try again.
+!doAction : pro(action)[a]
<- .print("Applying␣for␣permit␣A"); applyPermitA;
   !doAction.
//
// If prohibited to do action by b,
// apply for permit b, and try again.
+!doAction : pro(action)[b]
<- .print("Applying␣for␣permit␣B"); applyPermitB;
   !doAction.
```

Listing 2: Alice's plan in Jason

cepts to agents based on the environmental state. We provide a Java interface for the Jason environment called *Institutionable*, which provides the blueprint for extending the environment to incorporate the observe-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 mechanism to note, for each normative percept, that it is from an institution (inst), and from which institution (a or b).

5 Evaluation

A conventional system evaluation demonstrates performance. The aim of evaluating a pattern is to confirm its effective instantiation, in this case using two MAS components (Jason and InstAL), whose distribution incurs the expected overhead. Instal-REST itself is validated using the InstAL test suite (comprising some 1300 tests). We additionally demonstrate scalability in our instantiation, by means of the Flask micro web framework, that provides the web service function, handling incoming action data, and the Celery asynchronous task queue, that provides support for multiple deontic sensors, running queries on instances of institutional models.

We exercise our implementation of the ROA architecture pattern (Section 4) with a prototypical use case that examines the service performance under load by varying two parameters: the number of queries to be processed and the number of (client) agents. We run two synthetic workload tests between a laptop client and a server across a wide-area network. The first test sends the service a batch of queries, while the second uses a number of synthetic agents to generate batches of queries. The first stresses throughput, while the second aims to mimic workload in a typical system. Timings are averaged over 100 runs and are the difference between the sending of the first query and the receipt of the last result, in each case.

The first case – batched queries – uses a pool of 12 (Celery) workers for an event loop processing N queries, with batch sizes 2, 25, 50, 100 and 250 (see Table 2). Average time per query falls from 0.273 msec to 0.0140 msec, indicating, as expected, that network overheads outweigh query cost.

The second case – simulated agent workload – uses A threads, each corresponding to an agent, to send N queries as above, to 12 (Celery) workers. Table 3 shows results for 1, 5 ,10 ,25 agents and 2, 5 and 10 queries, indicating that doubling up either the agents or the queries does increase the total response time but does not double it, as the network overhead is amortised, meaning that the more agent clients, the greater the potential benefit (up to the capacity of the service, whose scalability is a different engineering problem).

Scenario	Average time	Stdev	Average time/query
2 queries	0.546	0.143	0.273
25 queries	1.21	0.587	0.0485
50 queries	1.82	0.529	0.0364
100 queries	2.40	0.592	0.0240
250 queries	3.49	0.268	0.0140

Table 2: Batched queries using 12 (Celery) workers

Scenario	Average time/agent	Stdev average time/agent	Average total time
1 agent 2 queries	0.547	0.089	0.628
5 agents 5 queries	0.890	0.275	1.80
10 agents 5 queries	0.958	0.153	1.80
10 agents 10 queries	1.45	0.194	2.44
25 agents 10 queries	2.48	0.135	4.38

Table 3: Simulated agent workload

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 frameworks. 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-Magentix2 [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 organisation ($\mathcal{M}$OISE), where $\mathcal{M}$OISE 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 refactored 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

[Aldewereld *et al.*, 2016] Huib Aldewereld, Olivier Boissier, Virginia Dignum, Pablo Noriega, and Julian Padget, editors. *Social Coordination Frameworks for Social Technical Systems*, volume 30 of *Law, Governance and Technology*. Springer, 2016.

[Baral, 2003] Chitta Baral. *Knowledge representation, reasoning and declarative problem solving*. CUP, 2003.

[Beck, 2000] Kent Beck. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.

[Behrens *et al.*, 2011] Tristan Behrens, Koen Hindriks, and Jürgen Dix. Towards an environment interface standard for agent platforms. *Ann. Math. Artif. Intell.*, 61(4):261–295, 2011.

[Boissier *et al.*, 2016] Olivier Boissier, Jomi Hübner, and Alessandro Ricci. The JaCaMo framework. In *Social Coordination Frameworks for Social Technical Systems*, pages 125–151. Springer, 2016.

[Bordini *et al.*, 2007] Rafael Bordini, Jomi Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, Ltd, 2007.

[Carlos Nieves *et al.*, 2011] Juan Carlos Nieves, Julian Padget, Wamberto Vasconcelos, Athanasios Staikopoulos, Owen Cliffe, Frank Dignum, Javier Vázquez-Salceda, Siobhan Clarke, and Chris Reed. Coordination, organisation and model driven approaches for dynamic, flexible, robust software and services engineering. In *Service Engineering*, pages 85–115. Springer, 2011.

[Dastani and Testerink, 2016] Mehdi Dastani and Bas Testerink. Design patterns for multi-agent programming. *Int. J. Agent-Oriented Softw. Eng.*, 5(2/3):167–202, January 2016.

[del Val Noguera *et al.*, 2010] Elena del Val Noguera, Natalia Criado, Carlos Carrascosa, Vicente Julián, Miguel Rebollo, Estefania Argente, and Vicente Botti. THOMAS: a service-oriented framework for virtual organizations. In *AAMAS'10*, pages 1631–1632. IFAAMAS, 2010.

[Dybalova *et al.*, 2013] Daniela Dybalova, Bas Testerink, Mehdi Dastani, and Brian Logan. A framework for programming norm-aware multi-agent systems. In *COIN'13, Revised Selected Papers*, pages 364–380. Springer, 2013.

[Esteva *et al.*, 2004] Marc Esteva, Bruno Rosell, Juan Antonio Rodríguez-Aguilar, and Josep Luís Arcos. Ameli: An agent-based middleware for electronic institutions. In *AAMAS'04*, pages 236–243. IEEE Computer Society, 2004.

[Fielding, 2000] Roy Fielding. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000.

[Fornara *et al.*, 2013] Nicoletta Fornara, Henrique Lopes Cardoso, Pablo Noriega, Eugénio Oliveira, Charalampos Tampitsikas, and Michael I. Schumacher. Modelling agent institutions. In *Agreement Technologies*, pages 277–307. Springer, 2013.

[García *et al.*, 2016] Emilia García, Soledad Valero, and Adriana Giret. ROMAS-Magentix2. In *Social Coordination Frameworks for Social Technical Systems*, pages 153–171. Springer, 2016.

[Grossi *et al.*, 2006] Davide Grossi, John-Jules Ch. Meyer, and Frank Dignum. Counts-as: Classification or constitution? An answer using modal logic. In *Deontic Logic and Artificial Normative Systems*, pages 115–130. Springer, 2006.

[Hindriks and Dix, 2014] Koen Hindriks and Jürgen Dix. GOAL: A multi-agent programming language applied to an exploration game. In *AOSE - Reflections on Architectures, Methodologies, Languages, and Frameworks*, pages 235–258. Springer, 2014.

[Hübner *et al.*, 2011] Jomi Hübner, Olivier Boissier, and Rafael Bordini. A normative programming language for multi-agent organisations. *Ann. Math. Artif. Intell.*, 62(1-2):27–53, 2011.

[Lee *et al.*, 2014] Jeehang Lee, Julian Padget, Brian Logan, Daniela Dybalova, and Natasha Alechina. N-Jason: Run-Time Norm Compliance in AgentSpeak(L). In *Engineering Multi-Agent Systems, Revised Selected Papers*, pages 367–387. Springer, 2014.

[North, 1991] Douglas North. *Institutions, Institutional Change and Economic Performance*. CUP, 1991.

[Ostrom, 1990] Elinor Ostrom. *Governing the Commons. The Evolution of Institutions for Collective Action.* CUP, 1990.

[Padget *et al.*, 2016] Julian Padget, Emad Elakehal, Tingting Li, and Marina De Vos. InstAL: An institutional action language. In *Social Coordination Frameworks for Social Technical Systems*, pages 101–124. Springer, 2016.

[Pokahr *et al.*, 2013] Alexander Pokahr, Lars Braubach, and Kai Jander. The Jadex project: Programming model. In *Multiagent Systems and Applications: Volume 1:Practice and Experience*, pages 21–53. Springer, 2013.

[Searle, 1995] John Searle. *The Construction of Social Reality*. Allen Lane, The Penguin Press, 1995.

[van Riemsdijk *et al.*, 2013] Birna van Riemsdijk, Louise Dennis, Michael Fisher, and Koen Hindriks. Agent reasoning for norm compliance: a semantic approach. In *AAMAS '13*, pages 499–506. IFAAMAS, 2013.

[Vázquez-Salceda *et al.*, 2010] Javier Vázquez-Salceda, Wamberto Vasconcelos, Julian Padget, Frank Dignum, Siobhan Clarke, and Manel Palau Roig. ALIVE: an agent-based framework for dynamic and robust service-oriented applications. In *AAMAS'10*, pages 1637–1638. IFAAMAS, 2010.

[von Wright, 1951] Georg von Wright. Deontic logic. *Mind*, 60(237):1–15, 1951.

[W3C, 2011] W3C. Semantic sensor network ontology. W3C Semantic Sensor Network Incubator Group. https://www.w3.org/2005/Incubator/ssn/ssnx/ssn., 2011. Retrieved 2018-01-18.