InstAL: An Institutional Action Language

Julian Padget, Emad ElDeen Elakehal, Tingting Li, and Marina De Vos

Dept. of Computer Science, University of Bath, United Kingdom

1 Introduction

Inst*AL* denotes both a declarative domain-specific language for the specification of collections of interacting normative systems and a framework for a set of associated tools. The computational model is realized by translating the specification language to Ans-$Prolog$ [6], a logic programming language under the answer set semantics (ASP) [16], and is underpinned by a set-theoretic formal model and a formalized translation process.

There are two novel features that Inst*AL* offers: one theoretical and one technical. The theoretical is the explicit treatment of external events and institutional events, which both makes clear when a physical world action "counts-as" [21] as a valid cyber world action and ensures that the institutional action functions as a guard for associated revisions of the institutional state. The technical contribution is that by using Answer Set Programming an Inst*AL* specification can serve as (i) a model in which to capture and validate (exhaustively) requirements as part of a design process – this is what Inst*AL* was originally developed to do – and (ii) a runtime requirements monitoring mechanism as part of a deployed system – which happens simply as a result of evaluating a model one step at a time.

Inst*AL* supports the specification of both regulative and constitutive norms and incorporates the well-known deontic notions of permission and obligation (but not full-scale prohibition) [37], along with the institutional notion of power [21].Given a normative specification and some initial conditions (fluents), an ASP solver allows the testing of properties of specifications over finite time frames as part of a design process, or the monitoring of agents' normative positions as part of a simulation [26,5] or a live system [35].

The key elements of the Inst*AL* language, as shown in its metamodel (see Figure 1), are the two kinds of data represented, namely events and fluents[1] – a set of fluents constitutes a state – and the three kinds of rules that capture the progression of the state, namely generation rules, consequence rules and non-inertial rules. These elements are brought together in a set-theoretic formal model along with a set of rules to translate the elements of the formal model into $AnsProlog$.

1.1 Brief history

Inst*AL* was initially conceived as a tool for the off-line verification of properties of normative specifications, taking advantage of the capacity of answer set solvers – tools

[1] A fluent is a term that is true by virtue of its presence (in the institutional state) and false when absent.

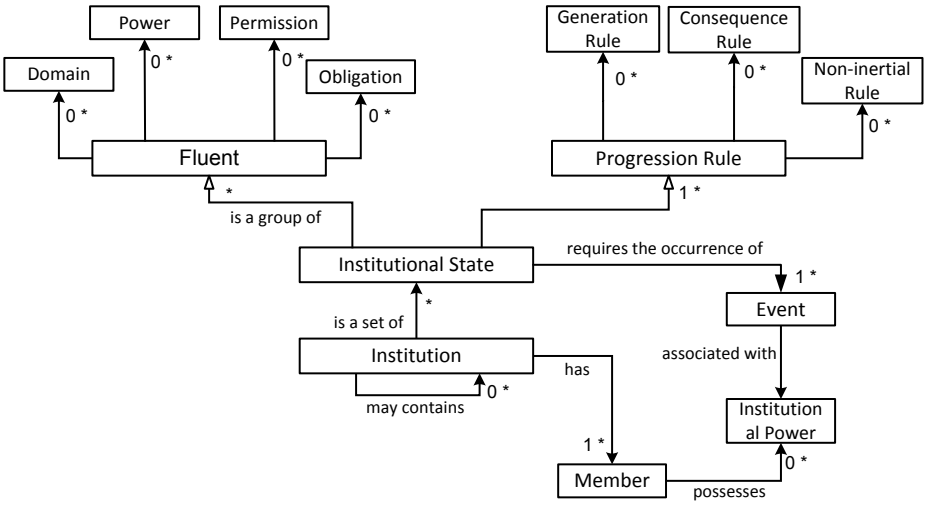

Fig. 1. Inst*AL* metamodel in UML notation

that take an $AnsProlog$ program and compute its answer sets – to generate finite event traces from a model, given some initial condition, and so construct a form of institutional model-checker. The benefit of using ASP is that it offers forward and backward reasoning and planning, all using the same program.

The first full description of Inst*AL* appears in Cliffe's dissertation [9], while an overview of the main concepts is given in [10]. Subsequently, the model was extended to account for a first attempt at interacting institutions [12], then called multi-institutions. The full model for interacting institutions is outlined in [29]

The first implementation of the Inst*AL* to ASP translator was written in Perl and designed to work with the then state-of-the-art ASP solver SMODELS. The current version is written in Python and supports interacting institutions and an improved more general $AnsProlog$ translation that works with the Clingo solver [15].

Variant models The ESI framework [38] proposes a variant Inst*AL*'s formal model. ESI combines state- and event-based norms by combining features from Inst*AL* and Opera/Operetta [1]. Compared to Inst*AL*, ESI offers an explicit representation of scenes, landmarks and roles. These can be indirectly modelled in Inst*AL* as discussed in [11]. ESI models violation fluents and negative obligations, allowing for a full representation of deontic notion of prohibition. Inst*AL* and ESI also differ in the way empowerment is used: the former models it for internal events only while the latter only considers it in respect of external events.

Tendering Use Case We apply the current standard version of the framework to the tendering use case.

1. Each phase of the use case is modelled as an individual institution, each specification comprising the events that it handles and the states (institutional facts) that it uses
2. The interactions between the individual institutions are choreographed by a special kind of institution, called a bridge institution, that specifies how events and states in one institution map to events and states in another
3. We illustrate the operation of the interacting institutions, using two tendering scenarios to show how some aspects of the requirements are met, by visualizing the corresponding traces.

1.2 Applications

Inst*AL* can be helpful, as much in the early stages of system development in support of exploratory design, as it is for deployed in support of monitoring and self-adaptation. The kind of systems for which it has been used are characterized by at least several and perhaps even large numbers of autonomous entities which may needs guidance on correct action selection or whose individual goals may be at odds with some of the system goals. We summarize a few examples of such systems:

1. Balke et al [4] shows how a normative framework can be used to govern the behaviour of handsets in an imagined 4G deployment to enable mesh networking of handsets as a way to ameliorate load on the insfrastructure network.
2. Lee et al [26] illustrates how a normative framework can be used to guide the behaviour of intelligent-agent controlled avatars in a virtual environment to bring about more plausible behaviours for non-player characters in Second Life.
3. Bibu et al [7] and Pieters et al [31] use normative specifications to explore and to establish security properties.
4. Balke et al [5] demonstrates the use of a normative framework to explore the parameter space for an enforcement policy and so establish appropriate levels for fines and (cost) effective levels of policing.
5. Li et al [29] applies Inst*AL* to the modelling of legal frameworks (building on [28]) to consider conflict detection between different jurisdictions.
6. Baines and Padget [3] applies normative guidance to (Jason) agents that control vehicles in a traffic simulation environment that combines SUMO [24] with Open Street Map and actual traffic data [36] to be able to explore the consequences of communication and reasoning about intentions.

2 Metamodel

2.1 Overview

Overview The meta-model in Figure 1 provides an overview of all the components in an Inst*AL* specification or model. In this section, we briefly describe its components. An Inst*AL* model is a collection of interacting institutions. Each interacting institution consists of a number of individual institutions and one or more bridge institutions that specify the (cross) generation and consequence relations between the individual ones.

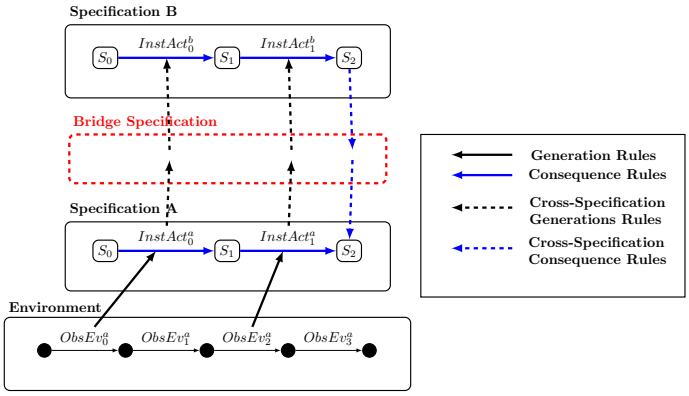

Fig. 2. Interacting institutions progression diagram

Each institution is determined by a set of possible states, a set of fluents that will determine to the make-up of each state, a set of progression rules and an initial state. The set of fluents available to an institution consists of two sub-sets: inertial and non-inertial. The former can be further broken up in domain fluents, that describe the environment in which the institution is active, and normative fluents. These latter ones, indicate which event-based fluents are active at any given time, based on their presence or absence from the state. Inst*AL* models empowerment, permission and obligation. Any event that is not permitted is prohibited. There are two types of events: external and institutional actions. The former are events observed by the institution (i.e actions of participants or environmental events). The latter are actions generated by the institution itself as a consequence of the occurrence of an event. This allows for a representation of the count-as event generation as detailed by [20]. State change in our model is triggered by the occurrence of a single external event. The institutions progression rules are responsible for the creation of the next state. Generation rules collect all the events that are caused by the single external event. The consequence rules determine which fluents need to be added/deleted from the current state to produce the next one and non-inertial rules are responsible for the presence of non-inertial fluents.

The bridge institution consists of cross generation and cross consequence rules that act across individual institutions. These determine if an event in one institution triggers an event in another institution, or adds or deletes a fluent in the state of another, respectively.

Figure 2 shows how interacting institutions can progress over time through the occurrence of a number of external events.

An Inst*AL* specification is a collection of interacting institutions that individually define a collection of event-based norms, i.e. norms that specify which events/actions are permitted, empowered or obliged. The meta-model is made concrete through Inst*AL*'s formal and computational model which support institutional collections where:

1. Each institution functions independently of one another, but may nevertheless affect one another, by establishing conflicting normative positions for an agent subject

2. The institutions are coupled, so that an action in one may bring about an event or a state change in another – these interactions are specified by a *bridge* institution – and, as above, conflicting normative positions can arise, and
3. The institutions are effectively unified, such that there are no conflicting normative positions: this can be the result of careful design or through a computational process of conflict detection and revision [28,29,27].

The formal model is a set-theoretic formulation based around sets of events and sets of facts with relations for the (cross-) generation and (cross-) consequence rules. A detailed description of the formal model can be found in [29] and [27].

The computational model is realised through translation to Answer Set Prolog (commonly also referred to as $AnsProlog$ or ASP) and the subsequent use of an answer set solver, such as Clingo [15]. Full details of the translation can be found in [10].

For ease of use, the framework offers an action language, also referred to as Inst*AL*, that allows the specification of institutions using a restricted semi-natural language. Full details of the language can be found in [9].

The Inst*AL* metamodel is supported by a collection of (command-line) tools for processing specifications (`pyinstal`), verifying their properties (`instql` and `clingo`), visualizing traces (`pyviz`) and incorporating them into distributed applications (the institution manager). An eclipse plug-in is available for modelling single institutions.

2.2 Assumptions

The Inst*AL* language derives from the Situation calculus [32], the Event calculus [23] and the action language $\mathcal{A}$ [17] and consequently has several features in common with the above, such as classical negation, negation as failure, default (non-monotonic) reasoning, the use of inertia to handle the frame problem and circumscription. The formal model is implementation agnostic. We chose to implement our model using answer set semantics in order to be able conveniently to reason about all traces, perform abduction, deduction and planning within a single implementation. The use of the action language allows an extra level of abstraction for ease of use by designers unfamiliar with a declarative logic programming language.

The premise behind Inst*AL* is that observable events – and the effect they have on some system state – are a sound basis on which to build models appropriate to all of cyber, physical and cyber-physical systems. Thus, the fundamental building block of an Inst*AL* model are the events that can be observed and consequently the orders in which they can be observed. A sequence of events is called a trace. The response by a model to an event – that it is designed to recognize – is a new (model) state, which is typically a modification of the state that pertained prior to the observation of the event. Model states are labelled and the labels are ordered, giving an approximation to the passage of time in terms of the occurrence of events.

By focussing on events as the driving force, Inst*AL* enables the modelling of existing physical and computational systems, mixed systems and the modelling of "what-if" systems, as illustrated in [4].

2.3 Main Constructs

We discuss the main features of Inst*AL* under the following headings: (i) ontology: what is the ontology of the metamodel and how is the domain ontology established (ii) events, activities and institutions: how are activities recognized through events and the role of institutions in establishing context (iii) concurrency and coordination: how is the observation of several events handled and how is coordination between activities achieved (iv) organizational structures: how are these modelled (v) implementation and deployment: how can Inst*AL* tools be integrated with other applications (vi) regulation and governance: what forms of regulation are then and how are regulations expressed and communicated (vii) norm revision/evolution: how does Inst*AL* cope with changes in the regulatory framework.

Ontology The ontology of the model is implicitly established by the design process. The author identifies actors, events and facts in the domain to be modelled and uses whatever names they choose to correspond to these elements in the specification. The modelling process might start from (physical world) events, leading to institutional events and then the facts (fluents) that are used to identify key elements of the institutional state. In addition to those fluents that pertain to the domain being modelled, there are three important kinds of fluents that underpin the core concept of the metmodel, namely permission, power and obligation. Permission and power are used to indicate attributes of events, that is, whether they are permitted or empowered. The obligation fluent is used to capture that a certain event should occur or a state be achieved before a certain (deadline) event occurs or a state is achieved, otherwise a special kind of event, a violation event, occurs. The connection between physical events and institutional events is established by the generation rules, named after the notion of conventional generation, established by Searle in the context of his work on speech acts [34,20]. The connection between (institutional) events and institutional facts is established by consequence rules which either initiate (add) or terminate (remove) fluents in the institutional state. Thus, the author introduces the domain ontology informally through the naming convention adopted for the events and fluents.

Events, Activities and Institutions The progression of the model is determined by the observation of an external event; if this is recognized by the institution (it can ignore it, depending on the conditions associated with the generation rule), then an institutional event – which maybe a violation – occurs. Through the consequence rules, the institutional event may affect the fluents (including obligations) in the institutional state.

Activities are connected in part by the rules in the specification and in part by the semantics of Answer Set Programming. However, the activities that occur can be constrained by what is observed in the query program. Inst*AL* was conceived purely for the purpose of modelling institutions and a fresh institution must be instantiated for each set of actors – in effect, grounding a copy of the specification – so that each such (grounded) institution carries out one of the functions of an organization. That is, we see institutions as constituents of organizations and each institution as an instantiation of an institutional specification that collectively form the set of patterns that govern participant behaviour in the organization.

Concurrency and Coordination Institutions are coordinated using bridge rules that communicate events from one institution to another or cause a fluent to be added in another institutional state. The tendering scenario is specified in this way, with individual institutions for the different activities of publication (discussed in detail in Section 2.5), review, decision and notification, while the interactions between them are governed by bridge rules (also discussed in detail in Section 2.5). Abstraction is thus achieved through composition.

Participation in an institution is typically represented by some (domain) fluents in some institutional state, so an agent can be present in more than one institution at the same time. In effect, each institution offers a perspective on an agent's actions as observed through external events. The progression of each institution can take place concurrently and activities can overlap, but in the context of any single institution, (external) events are considered to be totally ordered, which is a necessary condition to compute a stable model.

Organizational structures It is not surprising, given the institutional focus of Inst*AL*, that there are no organizational artefacts, such as groups or roles, explicit in the Inst*AL* syntax. Roles are added to the formal model in ESI [38] and such properties can readily be modelled as arbitrary relations/facts in Inst*AL*, e.g. $\texttt{plays}(\texttt{agent1}, \texttt{author})$, but changing associated permissions and powers when an agent start or stops playing a role requires additional supporting mechanisms if it is not to be cumbersome.

Implementation and Deployment As noted in Section 2.1implementation is achieved via the translation of the Inst*AL* components to $AnsProlog$and then the use of an answer set solver, e.g. CLINGO, (i) to establish global properties of the specification (at design time), (ii) to provide monitoring for compliance (at run time), and (iii) to provide a normative oracle for agents (at run time). The realization of run-time services depends upon the deployment environment. The institution manager component provides these services in conjunction with the Bath Sensor Framework [2] [25] which allows for the connection to an agent platform, such as Jason [8]. The institution manager can also be deployed as a RESTful web service, using the software developed in [14].

Regulation and Governance The two constraints on the action afforded by the model are permission and power [21]. In the absence of permission, an event is considered to be prohibited, but there is no explicit annotation for prohibition. The occurrence of an event that is not permitted leads to the occurrence of a violation event. On the other hand, the occurrence of an event that is not empowered simply has no effect (on the institutional state). Obligations are expressed as a combination of three elements: (i) either an event that ought to happen, or a condition on the institutional state that ought to be satisfied, (ii) either an event that might happen or a condition on the institutional state that might be satisfied, denoting a deadline for the first element (iii) a (violation) event that occurs if the first element is not satisfied before the deadline specified by the

[2] `https://github.com/mas-at-bath/bsf`

second. These language elements, along with domain fluents, allow for the expression of event-based (subject to state) regulatory norms and state-based norms.

It must be emphasized that the detection and handling of violations and obligations are the responsibility of the actors that participate in the institution; the sole purpose of the framework is to observe and maintain (through its progression rules) the social state pertaining to the perspective on events for which the institution has been designed.

Norm revision/evolution The Inst*AL* tools as described here aim to provide the means for the specification of static governance and monitoring mechanisms. The (provably) correct (minimal) revision of an institutional specification is the subject of on-going research (see this series of papers: [28,29,2,13]). Separately, on-going too are the related tasks of norm recognition and norm evolution [33].

2.4 Operations

There are three ways in which to use the Inst*AL* tools, depending whether the objective is model development or deployment and, in the latter case, whether the objective is system monitoring or system direction.

Model-checker Inst*AL* was firstly developed as a language for specifying and model-checking institutional specifications, and depends upon the computation of answer sets – that represent model traces – by means of an answer set solver. Answer sets are returned from solvers as sets of textual representations of atoms, and can be both large and many in number, depending on the number of time steps over which the solver is run and the number of possible orderings of events arising from the specification and the query program. Thus, the Inst*AL* suite of programs provide functions that prepare input for and process the output of the answer set solver: (i) a high-level domain-specific language for institutional specifications which is translated into $AnsProlog$ and combined with some institutional support code, and (ii) a renderer for answer sets to support the visual inspection of the interplay of events and states. The query program, in conjunction with the specification and the solver, can be sufficient to establish whether some model condition is specified, but in practice, models and conditions can become complicated quite rapidly and visualization of the answer set becomes a useful step in the development of the specification. The query program is an arbitrary $AnsProlog$ program, which is used to validate the model against its requirements by such things as: (i) certain sequences of events and (ii) the presence or absence of state traces (answer sets) that satisfy certain conditions. The query program can be written directly in $AnsProlog$ or generated using the InstQL querying language [19] (see Section 3) which allows expression of the query in a form that is aligned with the institutional specification.

Social Sensor The second use of Inst*AL* is as a monitor of the social state, providing in effect a kind of "social sensor" that uses observations of agent actions to progress the institutional model(s). Those agent actions may result in obligations being added to the social state and so the output function of the wrapper program in this usage, extracts

any (fresh) obligations from the single answer set that is produced and delivers them to the agent platform for perception by agents. In this way, agents become aware of the normative consequences of their actions and can decide whether to comply with the obligation or not [4,26,35]. In this mode of operation, Inst*AL* can be used in conjunction with agent-based simulation, virtual environments or live systems.

Social Oracle The third way in which to use Inst*AL* is as an institutional oracle, where the model can be used to extrapolate from the current institutional state to answer queries, such as: (i) whether an action – or a sequence of actions – is norm compliant (ii) whether an action – or a sequence of actions – results in a state satisfying a particular condition, and (iii) what (norm-compliant) action(s) results in a state satisfying a particular condition. These kinds of queries describe a fairly conventional usage of a finite-horizon model checker, but through the use of a domain-specific language and support for normative concepts, provides functionality suitable for multiagent and cyber-physical systems.

2.5 Languages

Models are expressed in Inst*AL*, a semi-natural action language, which is implemented in $AnsProlog$. The implementation language for Inst*AL* is $AnsProlog$ and therefore much of the semantics may be assumed from a knowledge of (Ans)Prolog, including in particular the syntax and treatment of variables (denoted by an upper-case initial letter, e.g. `Agent`) and constants, e.g. `requester`. Figure 3 depicts the encoding of the publication institution which is part of Inst*AL*'s implementation of the tendering use-case. It demonstrates how a single institution can be modelled in Inst*AL*. Figure 4 shows the tendering bridge institution as an example of how Inst*AL* supports the communication of events and states *between* institutions. A more detailed description of the language can be found in [27].

3 Tools and Platform

The purpose of Inst*AL* is to provide the functions described in section 2.4, namely model-checker, social monitor and social oracle. We now describe how each of these functions is realized, using a collection of command-line tools.

3.1 Model-Checker

Inst*AL* is a domain-specific action language whose computational model is realized by its translation to $AnsProlog$ and the subsequent solving to generate answer sets, which is realized by the use of an Answer Set Solver, subject to a query program that describes constraints on the answer sets. The translation is implemented by a command-line Python program, `pyinstal`[3], while for solving we use `clingo`[4]. The query

[3] `pyinstal` is available from `http://www.cs.bath.ac.uk/instal`
[4] `clingo` is available from `http://potassco.sourceforge.net/`

Publication institution

An institutions has a name (publication), some types for the parameters to events and fluents and some (selected) non-inertial fluents:

```
1  institution publication;
2  type RFT;
3  type Agent;
4  type Bid;
5  type Role;
6  fluent roleOf(Agent, Role);
7  fluent bidSubmitted(RFT, Bid, Agent);
```

One of the external observables, its corresponding institutional event, some facts for the initial state permitting both events, assigning some roles and the generation rule that implements counts-as, subject to `Agent` playing the role `requester`:

```
8  exogenous event publishRFT(Agent, RFT);
9  inst event intPublishRFT(Agent, RFT);
10 initially perm(publishRFT(Agent, RFT)),
11           perm(intPublishRFT(Agent, RFT)),
12 initially roleOf(ting, requester),
13           roleOf(alice, bidder),
14           roleOf(bob, bidder);
15 publishRFT(Agent, RFT) generates intPublishRFT(Agent, RFT)
16    if roleOf(Agent, requester);
```

The occurrence of the institutional event to publish the RFT has consequences for the institutional state, permitting and empowering any `bidder` agent to bid:

```
17 intPublishRFT(Agent, RFT) initiates
18    perm(registerBid(Agent1, Bid, RFT)),
19    perm(intRegisterBid(Agent1, Bid, RFT)),
20    pow(intRegisterBid(Agent1, Bid, RFT))
21    if roleOf(Agent1, bidder);
```

An agent is obliged to submit a bid before `intSubmissionDue` or it triggers the violation `lateSubmission`:

```
22 violation event lateSubmission(Agent, Bid, RFT);
23 obligation fluent obl(submitBid(Agent, Bid, RFT),
24                       intSubmissionDue(RFT),
25                       lateSubmission(Agent, Bid, RFT));
```

A non-inertial fluent is used in this example to implement a separation of roles:

```
26 noninertial fluent violated(Agent);
27 violated(Agent) when
28    roleOf(Agent, bidder), roleOf(Agent, requester);
```

Fig. 3. Example Inst*AL* specification for the publication phase

Tender institution

The propagation of events in coordinated institutions is achieved by specifying a cross-generation rule, linking an institutional event in one institution with an external (exogenous) event in another. Here we see several such rules from the tendering scenario that link the four constituent institutions:

```
1  intSubmissionDue(RFT) xgenerates reviewStarts(RFT);
2  intReviewDue(RFT) xgenerates decisionStarts(RFT);
3  lateReview(Agent, Bid) xgenerates waitForReview(Bid);
4  intDecisionDue(RFT, Bid) xgenerates notificationStarts(RFT, Bid);
5  intDecisionDue(RFT, Bid) xgenerates acceptBid(RFT, Bid)
6     if bidResult(RFT, Bid, accepted);
7  intDecisionDue(RFT, Bid) xgenerates rejectBid(RFT, Bid)
8     if bidResult(RFT, Bid, rejected);
```

The propagation of facts between coordinated institutions is achieved by specifying cross-consequence rules, linking the presence of a fact in one institution with either the initiation or termination of a fact in another.

```
9  intSubmitBid(Agent, Bid, RFT) xinitiates
10     readyForReview(RFT) if bidsReceived(RFT, n1);
11 intSubmitBid(Agent, Bid, RFT) xinitiates
12     bidInfo(RFT, Bid, Agent);
13 intReviewStarts(RFT) xinitiates bidDetails(RFT, Bid, Agent)
14     if bidInfo(RFT, Bid, Agent);
15 intMakeDecision(RFT, Bid, Decision) xinitiates
16     bidDecided(RFT, Bid, Agent)
17     if not reviewMissing(RFT, Bid), bidDetails(RFT, Bid, Agent);
```

However, some housekeeping is required to empower one institution to generate events for another:

```
18 cross fluent gpow(Inst, reviewStarts(RFT), Inst);
19 cross fluent gpow(Inst, decisionStarts(RFT), Inst);
20 cross fluent gpow(Inst, waitForReview(Bid), Inst);
21 initially gpow(publication, reviewStarts(RFT), review);
22 initially gpow(review, decisionStarts(RFT), decision);
23 initially gpow(review, waitForReview(Bid), decision);
```

And to empower one institution to initiate fluents in another:

```
24 cross fluent ipow(Inst, readyForReview(RFT), Inst);
25 cross fluent ipow(Inst, bidInfo(RFT, Bid, Agent), Inst);
26 cross fluent ipow(Inst, bidDetails(RFT, Bid, Agent), Inst);
27 initially ipow(publication, readyForReview(RFT), review);
28 initially ipow(publication, bidInfo(RFT, Bid, Agent), review);
29 initially ipow(review, bidDetails(RFT, Bid, Agent), decision);
```

Fig. 4. Example Inst*AL* bridge specification for the tendering scenario

program can be written directly in $AnsProlog$, but we also provide a query specification language called Inst*QL*. The translation of Inst*QL*queries to $AnsProlog$ is implemented by a command-line Python program, `pyinstql`[5]. For single institutions there is also an Eclipse plug-in available that integrates all the command-line tools and offers support to the user.

Careful expression of constraints is a critical aspect of using the solver effectively: on the one hand in order to limit the number of answer sets generated, which in the worst (unconstrained) case is exponential in the number of events, and on the other to restrict the number of answer sets to those that are of interest because they satisfy particular properties of concern to the designer. But even with one or a few answer sets, it can be difficult to identify key features of an answer set as generated by clingo, so we have developed a vizualizer, called `pyviz`[6], also implemented as a command-line Python program, that generates a pdf image of the trace using the TikZ drawing package.

These four components: translator, query generator, solver and visualizer, together support the model-checking function.

3.2 Social Monitor & Social Oracle

The provision of these two functions is achieved by the same framework, so we describe them together. The functions are delivered through the combination of two components: (i) the institution manager and (ii) a publish-subscribe interface. The pub-sub interface provides a facade for the institution manager and is present for the purpose of connecting the institution manager to the Bath Sensor Framework[7] [25], and thence, to an agent platform. The objective of the institution manager is to provide an interface to the model-checking mechanism that offers the following operations:

1. creation of a new instantiation of an institution, that is grounding it with the identities of the actors and entities it should govern
2. delivery of an external event to a given instantiation and running it for one cycle to produce the single answer set that characterises the next state of the institutional model
3. extract any obligations established by the event.

In the context of the BSF interface, the institution manager subscribes to events from the environment, which are then presented to the relevant institution and publishes obligations back to the environment. In this way, we achieve an agnotic interface for institutions and institution management in a distributed setting.

4 Inst*AL* in use

The modelling methodology of the institutions is the same regardless of whether the model is used in model checking, social monitor or oracle or any combination thereof. What differs is possible the granularity of the domain description and the consideration of enforcement. A detailed discussion of differences can be found in [5].

[5] `pyinstql` is available from `http://www.cs.bath.ac.uk/instal`

[6] `pyviz` is available from `http://www.cs.bath.ac.uk/instal`

[7] The Bath Sensor Framework provides distributed communications through the use of a XMPP server and is available from `https://github.com/mas-at-bath/bsf`

4.1 Modelling and Implementation

Inst*AL* models are built around events, who is permitted/empowered to cause them and the effects they have on institutional states, therefore the informal methodology for model specification is to examine the use case for actors, events and facts that need to be remembered. The events typically become external events in the specification, parameterised by relevant information, such the actors responsible for them, while facts become fluents, again parameterised by relevant information, that are initiated or terminated by (institutional) events subject to the institutional state at the time. Some scenarios contain more than one activity, in which case, it can be appropriate to map each activity to a separate institution and then specify cross-generation and cross-consequence rules to capture the interactions between them.

In the case of the tendering scenario, it is clear there are several phases, each of which we have chosen to map to a separate institutional specification, namely publication, review, decision and notification.Various events can be identified from the scenario description associated with each of the phases, which become external events in the specification. Taking the publication phase as an example (see Figure 3), there are also exceptional events, such as a submission after the deadline, which is captured as a violation event, if an actor does not meet the obligation to submit a bid before the end of the submission phase. Lastly, an institutional state or condition can be captured through the use of a non-inertial fluent, in this case to signal that a separation of roles has not been observed.

Once a specification comprises several institutions, it becomes necessary to consider how they may affect one another. This aspect is expressed using cross-generation and cross-consequence rules, as illustrated in Figure 4. In particular, the figure shows that the institutional event `intSubmissionDue` in the publication institution generates the external event `reviewStarts` in the review institution, with corresponding similar cross-generation rules connecting review and decision and decision and notification. Similarly, an event in one institution can bring about a change of state in another, such as the submission of a bid (`intSubmitBid`) in publication leading to the initiation of the fluent `readForReview` in the review institution.

As with conventional programming, an incremental, test-driven approach can be more productive than attempting to model everything first and then test it in its entirety. The same principle is at work in the use of interacting institutions that can be tested independently first and then integrated and tested further, as outlined in the tendering scenario. Visualisation of the answer sets as event/state sequences provides a mechanism of interpreting the answer sets within the problem domain. Issues with the modelling can become a more apparent this way. When writing the specification it is often useful to visualise specific traces as to see if all works as envisaged. This can be achieved by providing providing the solver with a specific of external events encoded as $AnsProlog$ facts.

Figures 5 and 6 are partial visualisations based on the external events provided to the solver. These events are provided at the top of the figure. Clearly several intermediate states have been left out, but also the state has been omitted in several places (S_0, S_1 and S_{23}) because it is large either in itself or in respect of the volume of changes that take place. The visualisation of trace should be read as follows:

1. Events are written in italic and labelled with the institution in which they are defined.
2. States are labelled in order as S_i. State descriptions are lists of selected fluents (specified by terms in the query program), again labelled with the institution in which they are defined. Fluents initiated in S_i are written in bold, persisting fluents in normal font and terminated fluents are struck through.
3. Transitions between states are labelled with the events that have brought about that transition. In this case, the external event observed is listed in the code block above and its occurrence, along with any generated institutional events is shown above the transition.

The trace illustrates how the tendering scenario has been filled in with actors and entities to explore the tendering process for a building contract. The events listed at the top of Figures 5 and Figure 6 show how the process unfolds:

1. `ting` publishes a request for tenders (RFT) for `westBuilding` and `artCentre`
2. `alice` registers a bid, then submits one, which is followed by a notification of its receipt. These actions are then repeated by `Bob`.
3. then, towards the end of the scenario
4. `marina` submits a review of bid `b3` for `westBuilding` in time for the deadline (indicated by `reviewDue`, leading to the start of the decision phase
5. with all the reviews received (review of bid `b1` was submitted earlier), the decisions are made and `b1` is accepted and `b3` is rejected, leading to the start of the notification phase
6. a contract is set up with `alice` for `b1` and a rejection notice is sent to `bob` in respect of `b3`, concluding the process.

4.2 Discussion

The use case serves well to illustrate of the capabilities of the Inst*AL* framework because it shows how the designer can utilise several interacting institutions. In this way, the designer can focus on the events associated with each activity – a form of modularisation – then focus on connecting up institutions through the bridge specification – a form of choreography.

As it is illustrated here, there is a degree of artificiality, in that we are working in isolation on the design, for the purpose of validating the static properties of the specification. This is a benefit, because it allows for the validation of a set use cases through a form of unit testing. What this does not illustrate is the deployment of such a model in either a simulated or live environment, where the events are generated by the actors, either agents or humans, or both.

This illustration, for sake of space, only shows a single (partial) trace of observed events that corresponds to a correct ordering, for the purpose of showing how a desired outcome is achieved. The trace shown is just one of the many possible paths from the initial state. In this case the answer set solver is constrained by the specification of those particular events in that particular order, such that there is only one result. The specification can be used equally to find all traces for all permutations of events, subject to any desired constraints, such as ones in which a particular final state is achieved and perhaps additionally, ones in which certain sub-sequences of events occur. Clearly, this

```
observed(publishRFT(ting, artCentre))
observed(registerBid(alice, b1, westBuilding))
observed(submitBid(alice, b1, westBuilding))
observed(notifyReceipt(alice, b1, westBuilding))
observed(registerBid(bob, b3, westBuilding))
observed(submitBid(bob, b3, westBuilding))
```

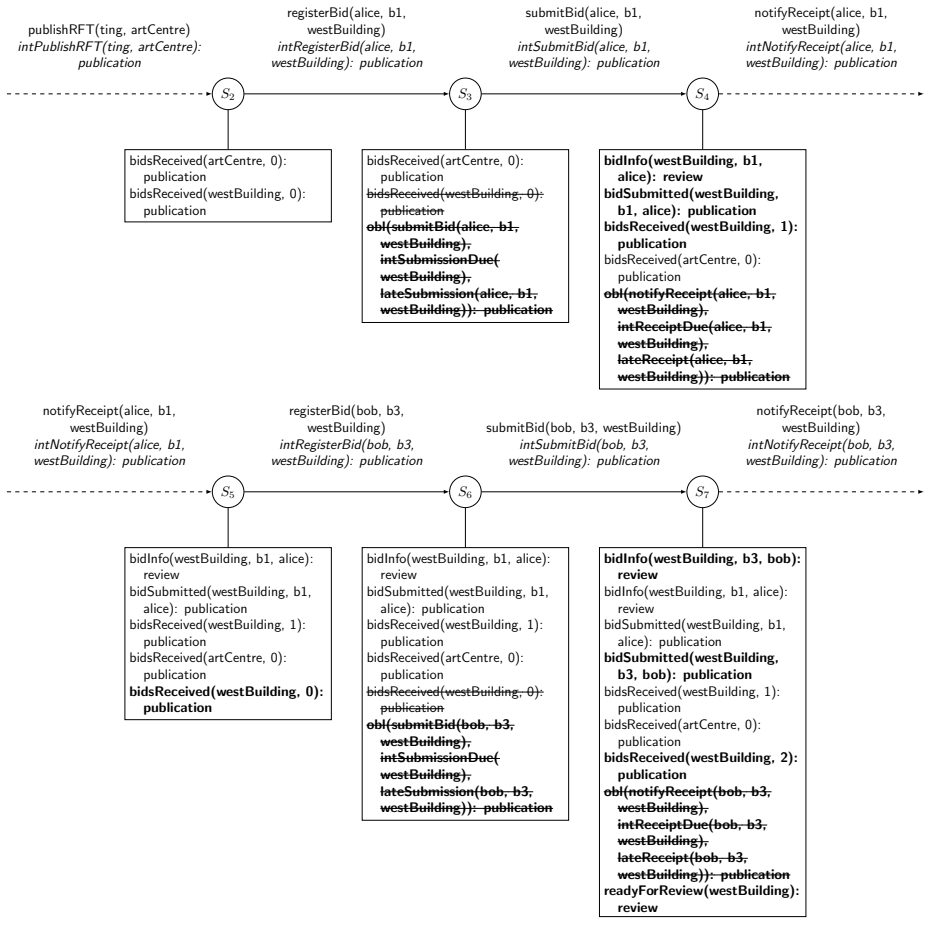

Fig. 5. A partial example event sequence for the start of the tender scenario (events 2–7)

```
observed(submitReview(westBuilding, b3, marina))
observed(reviewDue(westBuilding))
observed(makeDecision(westBuilding, b1, accepted)).
observed(makeDecision(westBuilding, b3, rejected))
observed(decisionDue(westBuilding))
observed(formContract(westBuilding, b1, alice))
```

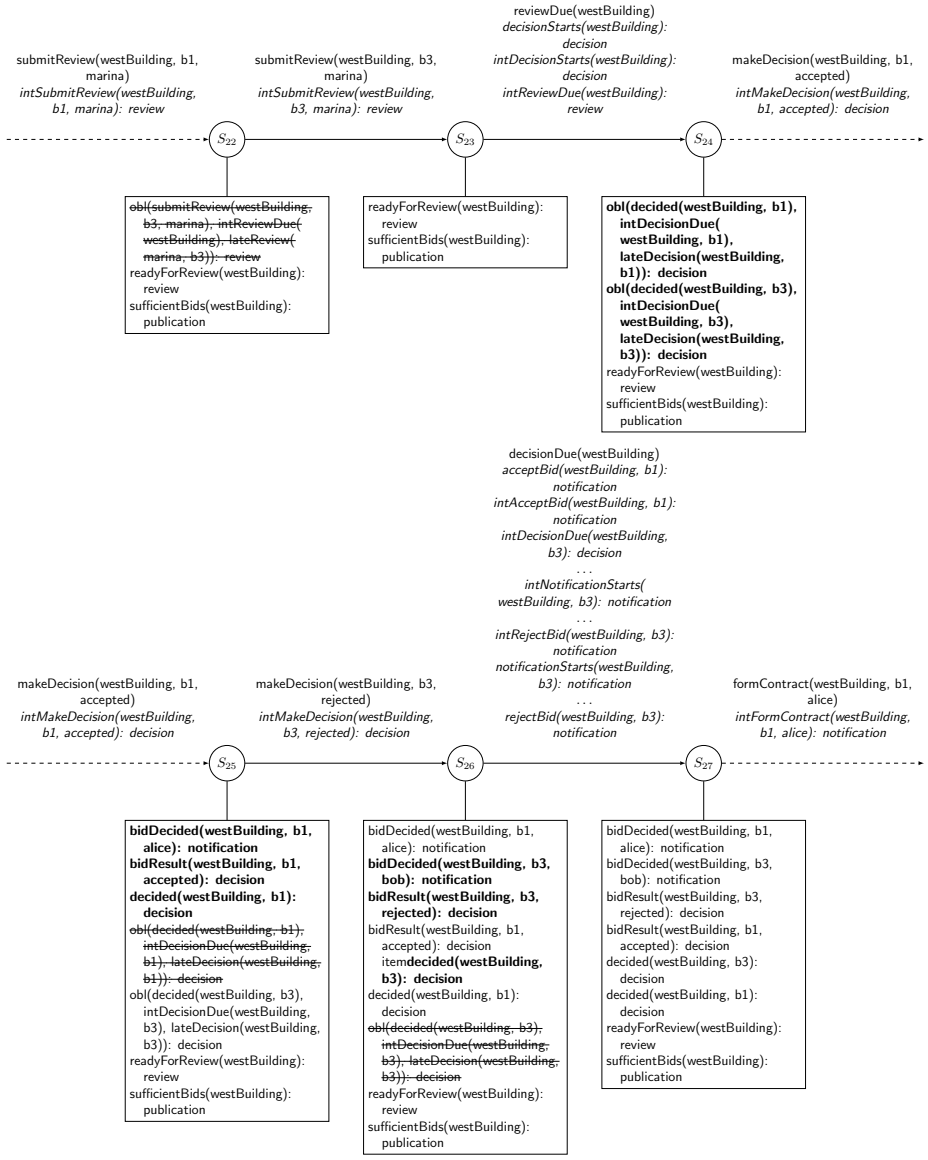

Fig. 6. A partial example event sequence for the end of the tender scenario (events 22–27)

latter usage, being exhaustive is potentially computationally expensive, but does provide the means to prove up to the length of the trace of the correctness of the specification against the events that the model recognizes. It also therefore supports the verification of the model against against all possible event sequences that might occur in practice in a live system.

5 Critical Assessment

The main strength of the framework lies in how it forces the analysis of the situations being modelled to focus on the identification of the significant events – the ones that are deemed to have an impact on the model – and the recognition of critical states brought about by those events. This imposes quite a high level of abstraction on the analysis process, which might create an impression of disconnection from a concrete application, but at the same time, this can potentially be beneficial precisely because of the enforced focus on the essential events (and states) divorced from implementation and technical choices.

A particular aspect of the modelling that can appear problematic is the absence of any explicit notion of advancing time, on the one hand, and the incapacity of the model to accommodate infinite event horizons, on the other. The former forces the designer instead to identify events that signify, for example, the end of a period, rather than saying something lasts 5 seconds, a day or a week. For the modeller, the absence of time units can initially be frustrating, but it eventually has a positive impact because it demands the explicit identification of a corresponding event, leading to a model expressed only in terms of the (asychronous) occurrence of events – which can obviously include externally generated time-dependent events – rather than one that has to account for both synchronous and asynchronous events. Inst*AL* models are not without time, because an event occurs at a given time instant (as in Figures 5–6), but the time period (between events) is elastic and the instants are enumerated (and ordered). Thus, it is possible to specify the duration of an activity, in terms of a number of time instants, but the chronological duration is kept outside the model.

Future work The Inst*AL* language is currently being extended to handle meta-norms [22], but the implications of this for the formal model have yet to be explored. Several issues of a technical nature are identified in [38] and summarised in section 1.1. These will be addressed in the short term.

A substantive extension to the toolset is under development for conflict detection and resolution for coordinated institutions, based on the principles outlined in [13], illustrated by application in [28,29] and described in detail in [27]. These tools allow for the specification of normative positions through use cases comprising a (partial) sequence of events and a (partial) state description and the consequent synthesis using inductive logic programming of a minimal self-consistent rule set. We currently use this to revise an existing rule set to accommodate institutional conflicts, but we propose to investigate how the same technology could be used against an initially null rule set to synthesize norms from examples.

We see two major challenges to be addressed both in Inst*AL* and by the wider norm research community. Both issues involve making normative systems more accessible

to non-specialists, firstly in authoring them and secondly in evaluating the impact of changes:

1. Writing specifications in Inst*AL* requires substantial knowledge of (computational) logic and a good understanding of logic programming and the semantics of Inst*AL*. The approach sketched above is one possible step towards a more user-oriented approach, but is still quite technical. Ghorbani [18] has prototyped a semi-structured web-based authoring environment, which suggests another more accessible approach. Similarly, Behaviour Driven Development [30] and the Relax [39] specification language offer stylised natural language specifications forms. More challenging technically is the question of whether natural language processing could identify events, states and actors in a policy specification and thereby synthesize a first approximation specification. The second challenge could be one way in which to refine and revise the outcome of the linguistic analysis process.
2. Understanding the full range of implications of a specification can be quite hard, since humans do not have high capacity for reasoning over combinatorial orderings and quantified formulae. A more effective approach for human expression of intentions is the use case – which is why we see it adopted in software engineering as a means to capture requirements – that describes a particular situation and how the model should behave given those conditions. However, a use case captures a situation in isolation, whereas in practice the properties that matter (safety, correctness, liveness, depending on the application) are consequential on the *interaction* of several requirements, some of which may conflict with one another, at some time or in some circumstances, or just all the time. The conflict detection and resolution approach outlined above can potentially help to construct conflict-free combinations of specifications, but it still requires a human to choose between the proposed revisions, so an effective way of presenting the effect of a set of norms and to differentiate between them seems essential.

6 Key references

The following four papers document the principal contributions underpinning Inst*AL*:

1. **Formal and computational model:** Cliffe et al [10] first sets out the formal model for an individual institution, identifying and separating out: (i) external and institutional events and (ii) generation (counts-as) and consequence relations, then proceeds to develop a mapping from the formal model to answer set semantics, thus realising the computational model. A more detailed discussion appears in [9].
2. **Multiple institutions and Action Language:** Cliffe et al [12] then extends the single institutional model to multiple institutions, introducing the notions of: (i) the propagation of events from one institution to another (ii) that one institution may have power (in the Jones and Sergot sense) over another (iii) how one institution may initiate or terminate facts in another. To facilitates the design of these multi-institutions, [12] proposes the action language Inst*AL*which can describe the formal model and maps to an answer set program. Again, a more detailed discussion appears in [9].

3. **Norm conflict detection and revision:** Li et al [28] addresses the problem of finding whether norms in different institutions conflict with one another and proposing a revision – by means of inductive logic programming – of one to be compatible with the other. For the purpose of the (automatic) detection process, two notions of conflict are identified: (i) **weak** conflict, in which a fact is be true in one institution and false in the other, and (ii) **strong** conflict, in which an action is not permitted in one institution, but is obliged in the other. Consequently, the model traces containing examples of these conflicts are used as negative examples for the inductive logic program, leading to proposals for the revision of the rules that give rise to the conflicts. A more detailed discussion appears in [27].
4. **Interacting institutions:** Li et al [29] builds on all the above to offer a general solution to institutional interactiwith the notion of the bridge institution which introduces: (i) cross institutional powers (ii) cross generation of events, and (iii) cross initiation and termination of institutional facts. The conflict detection and revision process is then applied to the circumstance of interacting institutions to revise the rules that give rise to the conflicts. A more detailed discussion appears in [27].

References

1. Aldewereld, H. and Dignum, V. Operetta: Organization-oriented development environment. In Dastani, M., Fallah-Seghrouchni, A. E., Hübner, J., and Leite, J., editors, *LADS*, volume 6822 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2010.
2. Athakravi, D., Corapi, D., Russo, A., De Vos, M., Padget, J. A., and Satoh, K. Handling change in normative specifications. In Baldoni, M., Dennis, L. A., Mascardi, V., and Vasconcelos, W. W., editors, *Declarative Agent Languages and Technologies X - 10th International Workshop, DALT 2012, Valencia, Spain, June 4, 2012, Revised Selected Papers*, volume 7784 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2012.
3. Baines, V. and Padget, J. A situational awareness approach to intelligent vehicle agents. In Behrisch, M. and Weber, M., editors, *Modeling Mobility with Open Data*, Lecture Notes in Mobility, pages 77–103. Springer International Publishing, 2015.
4. Balke, T., De Vos, M., and Padget, J. Analysing energy-incentivized cooperation in next generation mobile networks using normative frameworks and an agent-based simulation. *Future Generation Computer Systems*, 27(8):1092–1102, 2011.
5. Balke, T., De Vos, M., and Padget, J. I-ABM: combining institutional frameworks and agent-based modelling for the design of enforcement policies. *Artificial Intelligence and Law*, pages 371–398, 2013.
6. Baral, C. *Knowledge Representation, Reasoning and Declarative Problem Solving*. CUP, 2003.
7. Bibu, G., Yoshioka, N., and Padget, J. System security requirements analysis with answer set programming. In *Requirements Engineering for Systems, Services and Systems-of-Systems (RES4), 2012 IEEE Second Workshop on*, pages 10–13, 9 2012. `http://dx.doi.org/10.1109/RES4.2012.6347689`.
8. Bordini, R., Wooldridge, M., and Hübner, J. *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007.
9. Cliffe, O. *Specifying and Analysing Institutions in Multi-agent Systems Using Answer Set Programming*. PhD thesis, University of Bath, 2007.
10. Cliffe, O., De Vos, M., and Padget, J. Answer set programming for representing and reasoning about virtual institutions. In Inoue, K., Satoh, K., and Toni, F., editors, *CLIMA VII*, volume 4371 of *Lecture Notes in Computer Science*, pages 60–79. Springer, 2006.

11. Cliffe, O., De Vos, M., and Padget, J. Embedding landmarks and scenes in a computational model of institutions. In Sichman, J. S., Padget, J., Ossowski, S., and Noriega, P., editors, *COIN*, volume 4870 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2007.
12. Cliffe, O., De Vos, M., and Padget, J. Specifying and reasoning about multiple institutions. In Vazquez-Salceda, J. and Noriega, P., editors, *COIN 2006*, volume 4386 of *Lecture Notes in Computer Science*, pages 63–81. Springer, 2007. ISBN: 978-3-540-74457-3. Available via `http://dx.doi.org/10.1007/978-3-540-74459-7_5`.
13. Corapi, D., Russo, A., Vos, M. D., Padget, J., and Satoh, K. Normative Design using Inductive Learning. *Theory and Practice of Logic Programming, 27th Int'l. Conference on Logic Programming (ICLP'11) Special Issue*, 11(4–5), 2011.
14. Duan, K., Padget, J., and Kim, H. A. A light-weight framework for bridge-building from desktop to cloud. In Lomuscio, A., Nepal, S., Patrizi, F., Benatallah, B., and Brandic, I., editors, *ICSOC Workshops*, volume 8377 of *Lecture Notes in Computer Science*, pages 308–323. Springer, 2013.
15. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., and Schneider, M. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124, 2011.
16. Gelfond, M. and Lifschitz, V. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365–386, 1991.
17. Gelfond, M. and Lifschitz, V. Action languages. *Electron. Trans. Artif. Intell.*, 2:193–210, 1998.
18. Ghorbani, A. *Structuring Socio-technical Complexity: modelling agent systems using institutional analysis*. PhD thesis, Technical University of Delft, 2013. Available via `http://aminehghorbani.weblog.tudelft.nl/`, retrieved 20140730.
19. Hopton, L., Cliffe, O., Vos, M. D., and Padget, J. A. Inst*ql*: A query language for virtual institutions using answer set programming. In Dix, J., Fisher, M., and Novák, P., editors, *CLIMA*, volume 6214 of *Lecture Notes in Computer Science*, pages 102–121. Springer, 2009.
20. John R. Searle. *The Construction of Social Reality*. Allen Lane, The Penguin Press, 1995.
21. Jones, A. J. I. and Sergot, M. J. A formal characterisation of institutionalised power. *Logic Journal of the IGPL*, 4(3):427–443, 1996.
22. King, T. C., van Riemsdijk, M. B., Dignum, V., and Jonker, C. M. Supporting request acceptance with use policies. Presented at Coordination Organizations Institutions and Norms, 2014 (COIN@AAMAS). Available via `http://homepages.abdn.ac.uk/n.oren/pages/COIN14/papers/p15.pdf`, retrieved 20140730., May 2014.
23. Kowalski, R. A. and Sergot, M. J. A logic-based calculus of events. *New Generation Comput.*, 4(1):67–95, 1986.
24. Krajzewicz, D., Erdmann, J., Behrisch, M., and Bieker, L. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138, December 2012.
25. Lee, J., Baines, V., and Padget, J. Decoupling cognitive agents and virtual environments. In Dignum, F., Brom, C., Hindriks, K. V., Beer, M. D., and Richards, D., editors, *CAVE*, volume 7764 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2012.
26. Lee, J., Li, T., and Padget, J. Towards polite virtual agents using social reasoning techniques. *Computer Animation and Virtual Worlds*, 24(3-4):335–343, 2013.
27. Li, T. *Normative Conflict Detection and Resolution in Cooperating Institutions*. PhD thesis, University of Bath, December 2014.
28. Li, T., Balke, T., De Vos, M., Padget, J. A., and Satoh, K. A model-based approach to the automatic revision of secondary legislation. In Francesconi, E. and Verheij, B., editors, *ICAIL*, pages 202–206. ACM, 2013.

29. Li, T., Balke, T., Vos, M. D., Padget, J., and Satoh, K. Legal conflict detection in interacting legal systems. In Ashley, K. D., editor, *JURIX*, volume 259 of *Frontiers in Artificial Intelligence and Applications*, pages 107–116. IOS Press, 2013.
30. North, D. Introducing BDD. Available via `http://dannorth.net/introducing-bdd/`, retrieved 20140730.
31. Pieters, W., Padget, J., Dechesne, F., Dignum, V., and Aldewereld, H. Effectiveness of qualitative and quantitative security obligations. *Journal of Information Security and Applications*, 22:3–16, 2015.
32. Pinto, J. and Reiter, R. Reasoning about time in the situation calculus. *Ann. Math. Artif. Intell.*, 14(2-4):251–268, 1995.
33. Savarimuthu, B. T. R., Padget, J., and Purvis, M. Social norm recommendation for virtual agent societies. In Boella, G., Elkind, E., Savarimuthu, B. T. R., Dignum, F., and Purvis, M. K., editors, *PRIMA*, volume 8291 of *Lecture Notes in Computer Science*, pages 308–323. Springer, 2013.
34. Searle, J. R. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.
35. Thompson, M., Padget, J., and Battle, S. Governing Narrative Events With Institutional Norms. In Finlayson, M. A., Lieto, A., Miller, B., and Ronfard, R., editors, *2015 Workshop on Computational Models of Narrative*, OpenAccess Series in Informatics (OASIcs), Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. To appear.
36. UK Highways Agency. Traffic flow database system. Accessible via `https:/trads.hatris.co.uk`. retrieved 20160124.
37. von Wright, G. Deontic logic. *Mind*, 60:1–15, 1951.
38. Vos, M. D., Balke, T., and Satoh, K. Combining event-and state-based norms. In Gini, M. L., Shehory, O., Ito, T., and Jonker, C. M., editors, *AAMAS*, pages 1157–1158. IFAAMAS, 2013.
39. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H. C., and Bruel, J.-M. Relax: Incorporating uncertainty into the specification of self-adaptive systems. In *RE*, pages 79–88. IEEE Computer Society, 2009.