Inst*AL*: An Institutional Action Language and Framework for the Specification, Verification and Monitoring of Norm-governed Systems

tbd

Dept. of Computer Science, University of Bath, UK

1 Introduction

A few paragraphs that describe the framework in general terms and give the reader a good picture of what is the framework for and how mature and dynamic its development.

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 AnsProlog[7], 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. The language derives from the Situation calculus [31], the Event calculus [22] and the action language $\mathcal{A}$ [17] and consequently has several features in common with the above, such as the use of negation as failure, default (non-monotonic) reasoning, the use of inertia to handle the frame problem and circumscription. All of these are well-known issues – and solutions – in the context of knowledge representation and reasoning and we depend on the broader literature to support these choices.

There are two novel features that Inst*AL* offers: one theoretical and one technical. The theoretical is the explicit treatment of exogenous 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 (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. Going via the event calculus, typically implemented in Prolog, adds an extra translation step and due Prolog's sensitiveness to the order of rules and the use of cuts it can halt or return incorrect results.

Inst*AL* has been in use and under development since 2007. The original implementation was replaced in 2011 and has since then been extended in order to meet user's modelling requirements as the language has been applied in new domains. The most significant of these features are: (i) non-inertial fluents, which allow the creation of institutional fluents that hold for as long as some condition over existing institutional facts is true, (ii) interacting institutions, which allow for events in one institution to affect another and support the modularization of institutional specifications, and (iii) durations, which allow a fluent to hold at some number of time steps in the future.

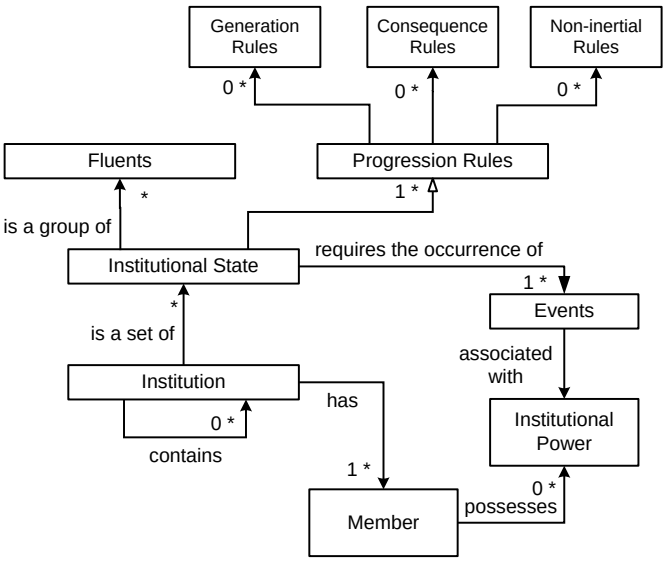

Fig. 1. Inst*AL* metamodel in UML notation

Main points not to miss: what is the main use of the framework (model, describe, specify, test; model and implement)

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) [**?**], along with the institutional notion of power [20], as appears in the event-calculus [22]. 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 agents' normative positions as part of a simulation [25,6] or a live system.

what are the main components of the "conceptual framework" (metamodel)

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, discussed in Section 2.1, along with a set of rules to translate the elements of the formal model into AnsProlog.

1.1 Brief history

Brief historical account of the Framework: where and when was this framework developed

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

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 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]. The two essential contributions of Cliffe's work were: (i) the realization that the computational properties of ASP could be applied to normative systems, and (ii) the construction of a formal model and its computational counterpart to effect the modelling of institutional fluents and how they might change under the influence of exogenous events. Subsequently, the model was extended to account for a first attempt at interacting institutions [12], then called multi-institutions. The revised multiple institution model is outlined in [26]

The first implementation of the Inst*AL* to ASP translator was written in Perl, and for several years this supported the development and application of Inst*AL* models in a variety of domains, some of which are discussed in the Applications section (1.2). This version was designed to work with the LPARSE grounder and the SMODELS solver[30].The current version works with the Clingo solver [15], so the translation now generates an unground program to work with an efficient grounding solver. This version also includes the new multiple institution model, and a re-implementation of the trace visualizer, all of which are now written in Python.

Variant models

How mature is the system? what version of the framework you are talking about. Mention if there are other variants.

Landmarks and non-inertial fluents The notion of landmark originates from [34,24] where the essence is of a condition on a state in order for an action in some protocol to have effect. Consequently, the concept forms an integral part of the graphical notation of scenes in Operetta [2], although it is worth noting that Operetta specifications are expressed entirely in terms of normative positions (that is, states) and it is purposely left undefined how the transition from one normative position to another may be achieved. In [11], we identified a similar effect that arises naturally from the formulation of the progression rules of the institutional specification, in that as a result of the interaction of the event generation and the consequence relations, fluents are initiated and terminated in respect of an event and some conditions on the state of the institution. This corresponds precisely with the notion of landmark, in that an event takes the institution into a new state but this is predicated on the current state – that is, a condition. However, a criticism of this approach is that the identification of the state is tied to the conditions on an initiation or termination rule that can only be triggered by an (institutional) event. From a specification point of view, this means that in the case where several unordered events might preface the normative position, the consequence rules would be both complicated and hard to maintain. Non-inertial fluents (in the current version of Inst*AL*) offer a more elegant solution, whereby each fact (fluent) that contributes to the

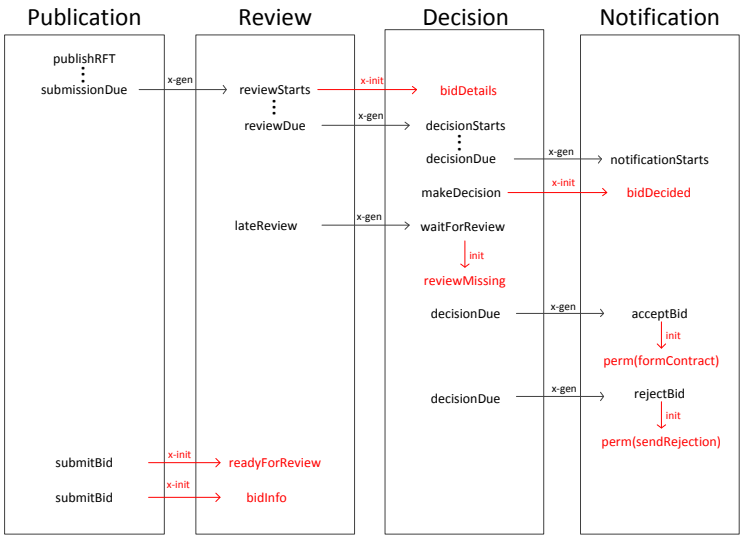

Fig. 2. High level view of the interacting institutions of the tendering scenario

normative position can be part of the condition for a non-inertial fluent expression, such that the fluent is true for as long as the condition is satisfied.

Later, [35] offers a formal perspective on both non-inertial fluents and landmarks. For the latter they introduce specific fluents, in the style of Operetta and specific rules that indicate when the landmarks should be initiated and what are the consequences of reaching such a landmark.

Empowerment The original Inst*AL* model regards external events as not permittedby default, but empowered (since they cannot be otherwise),whereas institutional events are not permitted and not empowered by default and such normative fluents must be initiated if an institutional event is not to result in a violation or simply ignored. An alternative view of empowerment is put forward in [35], where it is associated with external events as an indicator of whether the institution should acknowledge the event or not.Institutional events, since they are part of the institution, are automatically acknowledged and therefore empowered.

Violation Fluents and negative obligations Given the event and state-based nature of the Inst*AL* formalism, [35] introduces violation fluents to indicate that a norm was violated. This has the added benefit that violations can now be tracked in the state rather than as institutional events that need to be monitored. A further introduction is the negative obligation, indicating that a landmark should not be achieved or that an event should not occur.

Tendering Use Case

What variants and parts art of the framework you are describing and you used in the modelling of the usecase.

We use the current standard version of the framework (ie. with interacting institutions, with non-inertial fluents, without landmarks and with the original notion of empowerment) to illustrate the tendering use case (Figure 2):

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

Give a good enough indication of where the framework has been used and why (domains, types of systems, features of the systems where the framework is best fitted to be used.

Inst*AL* has been used for (i) modelling systems in advance of development, in order to improve designers' understanding of the properties of the system and (ii) providing a social reasoning component that governs agents both in simulation and in virtual environments – where agents interact with human actors.

Balke et al [5] shows how a normative framework can be used to govern ... and to enable to the prototyping of components that do not yet exist

Lee et al [25] illustrates how a normative framework can be used to guide the behaviour of intelligent-agent controlled avatars in a virtual environment...

Bibu et al [8]... security

Balke et al [6]... policy

Li et al [29] applies Inst*AL* to the modelling of legal frameworks (building on [27]) to consider conflict detection between different jurisdictions...

Baines and Padget [4] applies normative guidance to (Jason) agents that control vehicles in a traffic simulation environment that combines SUMO [23] with Open Street Map and actual traffic data [1] to be able to explore...

2 Metamodel

2.1 Overview

Describe succinctly what are the components of a systems that is developed with the framework in terms of the main conceptual constructs of the framework.
For example a JaCaMO system consists of a workspace where agents do... using artefacts ... structural specification ... normative view,...
Or, an electronic institution is an interrelated scenes where... transitions, speech acts, ... protocols...
Illustrate these comments with the use case.

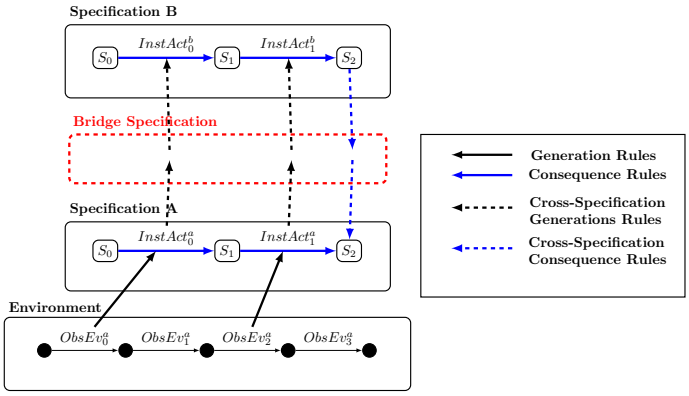

Fig. 3. Cross-Specification Generation and Consequence Functions

An Inst*AL* model is a collection of interacting institutions that individually define a collection of event- and state-based norms. The Inst*AL* formal and computational model supports 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 [].

Does it have a set of tools or a full platform associated? How are these computational components related to the conceptual framework

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).

Formal Model The formal model is a simple set-theoretic formulation based around sets of events and sets of facts. In the terminology of Inst*AL*, the facts are called fluents and are true if present and unknown if absent. There are two kinds of fluents: (i) inertial: these are initiated subject to an event and some condition over the state, and persist from state to state until terminated, again contingent on an event and some condition over the state (ii) non-inertial: these are asserted subject to some condition over the state, but only persist for as long as the condition holds.

JP: update figure **??** to include non-inertials

Mathematical model for a single institution:

$\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{G}, \mathcal{C}, \Delta \rangle$, where

1. $\mathcal{F} = \mathcal{W} \cup \mathcal{P} \cup \mathcal{O} \cup \mathcal{D}$
2. $\mathcal{G} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{E}_{inst}}$
3. $\mathcal{C} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{F}} \times 2^{\mathcal{F}}$ where $C(\phi, e) = (\mathcal{C}^{\uparrow}(\phi, e), \mathcal{C}^{\downarrow}(\phi, e))$ in which
   (i) $\mathcal{C}^{\uparrow}(\phi, e)$ initiates fluents
   (ii) $\mathcal{C}^{\downarrow}(\phi, e)$ terminates fluents
4. $\mathcal{E} = \mathcal{E}_{ex} \cup \mathcal{E}_{inst}$ where $\mathcal{E}_{inst} = \mathcal{E}_{act} \cup \mathcal{E}_{viol}$
5. Δ
6. State Formula: $\mathcal{X} = 2^{\mathcal{F} \cup \neg \mathcal{F}}$

Translation rules for a single institution:

$$
\begin{aligned}
p \in \mathcal{F} &\Leftrightarrow \texttt{fluent(p, In)}. \\
e \in \mathcal{E} &\Leftrightarrow \texttt{event(e)}. \\
e \in \mathcal{E}_{ex} &\Leftrightarrow \texttt{evtype(e, In, obs)}. \\
e \in \mathcal{E}_{act} &\Leftrightarrow \texttt{evtype(e, In, act)}. \\
\in \mathcal{E}_{viol} &\Leftrightarrow \texttt{evtype(e, In, viol)}. \\
\mathcal{C}^{\uparrow}(\phi, e) = P &\Leftrightarrow \forall p \in P \cdot \texttt{initiated(p, In, I)} \leftarrow \texttt{occurred(e, In, I)}, EX(\phi, I). \\
\mathcal{C}^{\downarrow}(\phi, e) = P &\Leftrightarrow \forall p \in P \cdot \texttt{terminated(p, In, I)} \leftarrow \texttt{occurred(e, In, I)}, EX(\phi, I). \\
\mathcal{G}(\phi, e) = E &\Leftrightarrow g \in E, \texttt{occurred(g, In, I)} \leftarrow \texttt{occurred(e, In, I)}, \\
&\qquad \texttt{holdsat(pow(p), In, I)}, EX(\phi, I). \\
p \in \Delta &\Leftrightarrow \texttt{holdsat(p, In, i00)}.
\end{aligned}
$$

Mathematical model for interacting institutions:

$C_I = \langle \bigcup_{i=1}^{n} \mathcal{I}^i,\ \mathcal{G}^x, \mathcal{C}^x, \Delta^x, \delta^x \rangle$:

1. $\mathcal{I}^i = \langle \mathcal{E}^i, \mathcal{F}^i, \mathcal{G}^i, \mathcal{C}^i, \Delta^i \rangle$
2. $\mathcal{F}^x = (\mathcal{W}_g \cup \mathcal{W}_i \cup \mathcal{W}_t)$, $\mathcal{F}^x \subset (\mathcal{F}^d \cap \mathcal{F}^s)$: $\begin{cases} \mathcal{W}_g = \{gpow(s, e, d)\}, e \in \mathcal{E}^d \\ \mathcal{W}_i = \{ipow(s, f, d)\}, f \in \mathcal{F}^d \\ \mathcal{W}_t = \{tpow(s, f, d)\}, f \in \mathcal{F}^d \end{cases}$
3. $\mathcal{F}^m = \bigcup_{i=1}^{n} \mathcal{F}^i, \mathcal{E}^m = \bigcup_{i=1}^{n} \mathcal{E}^i$
4. $\mathcal{G}^x : \mathcal{X}^m \times \mathcal{E}^m \rightarrow \langle 2^{\mathcal{E}_1}, \ldots, 2^{\mathcal{E}_n}, 2^{\mathcal{E}^m} \rangle$
5. $\mathcal{C}^x : \mathcal{X}^m \times \mathcal{E}^m \rightarrow \langle 2^{\mathcal{F}_1}, \ldots, 2^{\mathcal{F}_n}, 2^{\mathcal{F}^m} \rangle \times \langle 2^{\mathcal{E}_1}, \ldots, 2^{\mathcal{F}_n}, 2^{\mathcal{F}^m} \rangle$
6. $\Delta^x = \bigcup_{i=1}^{n} \Delta_i$
7. $\delta^x \subseteq \mathcal{F}^x$
8. $\mathcal{X}^m = 2^{\mathcal{F}^m \cup \neg \mathcal{F}^m}$

Translation rules for interacting institutions:

$$
\begin{aligned}
\mathcal{G}_x(S^m, e) = E &\Leftrightarrow g \in E, S^m \models \mathcal{X}^m \\
&\texttt{occurred(g, D, T)} \leftarrow \texttt{occurred(e, S, T)}, \texttt{holdsat(gpow(S, g, D), I)}, \\
&\qquad \texttt{inst(S; D)}, EX(\mathcal{X}^m, T), \texttt{instant(T)}. \\
\mathcal{C}_x^{\uparrow}(S^m, e) = P &\Leftrightarrow p \in P, S^m \models \mathcal{X}^m \\
&\texttt{initiated(p, D, T)} \leftarrow \texttt{occurred(e, S, T)}, \texttt{holdsat(ipow(S, p, D), T)}, \\
&\qquad \texttt{inst(S; D)}, EX(\mathcal{X}^m, T), \texttt{instant(T)}. \\
\mathcal{C}_x^{\downarrow}(S^m, e) = P &\Leftrightarrow p \in P, S^m \models \mathcal{X}^m \\
&\texttt{terminated(p, D, T)} \leftarrow \texttt{occurred(e, S, T)}, \texttt{holdsat(tpow(S, p, D), T)}, \\
&\qquad \texttt{inst(S; D)}, EX(\mathcal{X}^m, T), \texttt{instant(T)}.
\end{aligned}
$$

Fig. 4. Mathematical models and AnsProlog translation rules for single and interacting institutions.

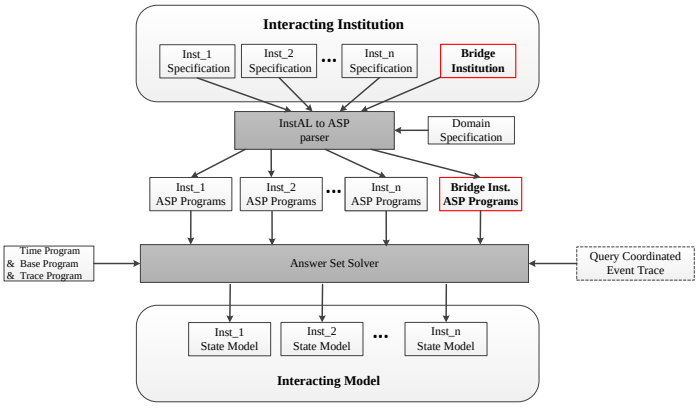

Fig. 5. The computational process for the progression of interacting institutions

Computational Model The computational model is realised through translation to Answer Set Prolog (referred to AnsProlog or ASP) and subsequently the use of an answer set solver, such as Clingo [15]. The central idea is that the formal mathematical model is represented as an answer set program – as shown in Figure **??** – this is then combined with some supporting code: (i) the time program, which specifies the time instants over which the solver will run (ii) the base program, which handles fluent inertia, and (iii) the trace program, which specifies the literals to in/exclude from the generated answer set.

JP: check this for consistency with Tingting

2.2 Assumptions

What is the theoretical or conceptual background upon which your framework rests?

Inst*AL* is inspired by the Situation Calculus [31] and the Event Calculus [22]...

assumptions about the systems that may be modelled with the framework: Mention what is specific to the framework (as opposed to open, heterogeneous, self-motivated, autonomous ....). assumptions about the types of systems that the metamodel is not interested or capable to model

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.

What are the functions that the metamodel is intended to serve (to facilitate understanding of systems, to describe existing systems, to design new systems, to design and achieve a formal specification, ..., , to specify, validate and run,...

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, such as that illustrated by [5]...

2.3 Main Constructs

Try to comment on all the features (and affordances) of the table.
Describe the building blocks of the metamodel (agents, artefacts, normative specification, scene, protocol, speech act, event, role, performance indicators, agents, procedural norms, governance,.....). Illustrate with the use case.

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.

MDV: I assume this is a fixed template but some introductory sentences would be nice if possible

JP: added

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 [33,19]. 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.

MDV: why not permission and power, the designer needs to know about them as well although they get automatically generated

JP: added

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 choose to ignore it, depending on the conditions on 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 exogenous events. The progression of each institution can take place concurrently and activities can overlap, but in the context of any single institution, (exogenous) events are considered to be totally ordered, which is a necessary condition to compute a stable model.

MDV: I did not follow when I first went over it. Needs some more explanation.

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 [35] and such properties can readily be modelled as arbitrary relations/facts in Inst*AL*, e.g. `plays(agent1, 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.

JP: not the right place?

MDV: I think this section should be called Verification and Compliance and the first sentence rewritten

Implementation and Deployment As noted in Section **??**, and in Figure 5, implementation 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] [**?**] which allows for the connection to an agent platform, such as Jason[**?**]. The institution

JP: what ought it to say?

JP: There is supposed to bve a section on interaction with the external environment

[2] `https://code.google.com/p/bsf/`

manager can also be deployed as a RESTful web service, using the software developed in [14].

MDV: I think these sentences should go elsewhere

Regulation and Governance The two constraints on the action afforded by the model are permission and power [20]. 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 second. These language elements, along with domain fluents, allow for the expression of event-based (subject to state) regulatory norms and state-based norms.

JP: revised

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 for example [27,28,3,13]). Separately, on-going too are the related tasks of norm recognition and norm evolution [32].

2.4 Operations

What are the main functions or operations that the framework supports: atomic actions, proof, model check, open a scene,....

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 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 [18] (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 [5,25]. 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-hysical systems.

2.5 Languages

How is the modelling expressed? Is there a description language? a specification language? Are norms deontic formulas?, ...

JP: name change here: needs following through

Models are expressed in Inst*AL*. We illustrate the language in two ways: by a full description of syntax and by reference to fragments of the publication institution (constituent) and the tender institution (bridge) from the tendering scenario (see Figure 6). From a syntactic point of view, an Inst*AL* description starts by declaring the name of the institution; thereafter declarations can be in whichever order suits the designer with the constraint that declaration precedes use.

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`)

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. 6. Example Inst*AL* specification for the publication phase

and constants, e.g. `requester`. The reference semantics are established by the formal translation given in Figure 4.

The rest of this section is divided into two parts, where the first describes and illustrates the features of the specification language for an individual institution and the second does the same for institutional interaction, which is how Inst*AL* supports the communication of events and states *between* institutions.

The institution specification language The objective of the analysis of a scenario is firstly, the identification of the actors, artifacts, events and facts that appear to be critical for building a sufficiently accurate model and secondly, the identification of any self-contained groups of activities that might constitute separate institutions. This latter aspect is addressed in the following section about interacting institutions. Thus, the first part of the language description focusses on (i) the elements that make up the institutional state – institutional facts or fluents (ii) the events that trigger the progression rules that recognize external events and subsequently revise the institutional state, and (iii) the types of the constituents of events and fluents – used to identify classes of actors and artifacts.

JP: what do you feel about the term artifact?

Events, states and types We begin the description of the language from the point of view of what a designer might wish to express as a result of their analysis of the problem, namely events, states and types:

```
type P1;
type Q1;
...
exogenous event ename1(P1,...);
inst event iename1(P1,...);
violation event vename1(Q2,...);
...
fluent fname1(Q1,...);
```

JP: revised

These declare: (i) the types `P1` and `Q1`, describing a set of atoms which may be applied to the parameters of fluents and events in rule descriptions. (ii) an external (exogenous) event `ename`, whose parameter types are `P1` etc., which identifies an observable event in the environment in which the institution is situated, (iii) an internal (institutional) event `iename`, whose parameter types are `P1` etc., identifying an event that can occur in the institution, (iv) a violation event `vename1`, with parameter types `Q2`, etc., which is a special kind of institutional event, used to signal a non-compliant action, and (v) an institutional domain fact `fname`, with parameter types `Q1` etc., that captures a salient feature of the model space. The types parameterising events and those parameterising fluents can be freely mixed. The use of P and Q above does not indicate any constraint. The typing system is monomorphic.

Generation rules and "counts-as" The next step is the recognition of external events as institutional events,[3] corresponding with the notion of "counts-as", described by Jones and Sergot [20]. At its simplest, this can just be an automatic one-to-one mapping, but normally, this mapping needs to be expressed in a conditional form, in order to capture the (institutional) circumstances in which the event is observed *and* recognized:

```
ename1(P1,...) generates iename1(Q1,...), iename2(R1,...)
   if fname1(...), not fname2(...), X==Y;
```

in which the institutional events `iename1` and `iename2` only occur if `fname1(...)` unifies in the current institutional state, `fname2(...)` does not, i.e. is not present in the current state, and `X` is equal to `Y`. If the conditions are not met, then the institutional event is not generated. See for example, lines 15–16 of Figure 6, where the `intPublishRFT` event is only generated if `Agent` is playing the role `requester`.

Consequence rules The recognition of an event can be used in two ways: (i) to generate a corresponding institutional event as discussed above, and (ii) to effect changes directly in the institutional state. Similarly, an institutional event can also be used to effect state changes and in this way, the counts-as mechanism functions as a gatekeeper for the state. There are two kinds of consequences arising from an (institutional) event: (i) initiates, where fresh fluents are added to the state, subject to some conditions, and (ii) terminates, where existing fluents are removed, again subject to some conditions:

```
ename1(P1,...) initiates fname1(Q1,...), fname2(R1,...)
   if fname3(S1,...) ...
iename2(P1,...) initiates fname1(Q1,...), fname2(R1,...)
   if fname3(S1,...) ...
```

where `fname1` and `fname2` are added to the state if `fname3` unifies in the current state, and the rest of the condition is satisfied, when either `ename1` is observed or `iename2` occurrs. Lines 17–21 of Figure 6 illustrate the adding of fluents for permission and power to register a bid for any `Agent1` playing the role of `bidder`.

```
iename2(P1,...) terminates fname1(Q1,...), fname2(R1,...)
   if fname3(S1,...) ...
```

where `fname1` and `fname2` are removed from the state if `fname3` unifies in the current state, and the rest of the condition is satisfied.

Obligations One common form of requirement is that a particular sequence of events occur. That is, if an agent carries out one action, then another has to follow within some period, and if not that circumstance must be detected: this is captured by the combination of obligation fluents and violation events:

```
violation event ename5(A,B);
obligation fluent obl(ename1(A,B),  % event
```

[3] Note: we adopt the naming convention of prefixing institutional events with an 'i' to avoid the need to remember or refer back to the declaration, but this denotation has no significance for the semantics of the language.

```
                        iename2(A,B),  % deadline event
                        ename5(A,B)); % violation

obligation fluent obl(ename2(A,B),  % event
                        fname3(A,B),  % deadline state
                        ename5(A,B)); % violation
```

where the event `ename1` must occur before the deadline event `iename2` (or before a state containing `fname3` is reached), otherwise a violation event `ename5` occurs. Lines 22-25 of Figure 6 express the link between the obligation for `submitBid` to take place before the institutional close of bidding, denoted by the event `intSubmissionDue`, otherwise the violation event `lateSubmission` occurs.

In addition to the requirement for a certain event to occur – as in two cases above – Inst*AL* also supports a form of obligation with regard to the achievement of a certain institutional state before a deadline. Thus, rather than specifying that an agent is obliged to perform a particular action, it is instead required that the institutional state satisfy a certain condition before some deadline occurs or some state condition is satisfied, otherwise a violation event occurs:

```
violation event ename5(A,B);
obligation fluent obl(fname1(A,B),  % state
                        iename2(A,B),  % deadline state
                        ename5(A,B)); % violation

obligation fluent obl(fname1(A,B),  % state
                        fname3(A,B),  % deadline state
                        ename5(A,B)); % violation
```

where the fluent `fname1` characterises the required state to reach before the deadline event `iename2` (or the certain state `fname3`), otherwise a violation event `ename5` occurs. In all cases, where a state is specified, this can be an arbitrary boolean expression over the institutional state.

Normative positions Another class of requirement is the recognition of a particular configuration of the institutional state, e.g. arising from the arbitrarily ordered occurrence of several actions:

```
noninertial fluent fname4(T1,...);
fname4(T1,...) when fname1(P1,...), not fname2(Q1,...);
```

where `fname4` is true iff `fname1` unifies in the current state and `fname2` does not. Lines 26–28 in Figure 6 show the use of this feature to express a constraint on the separation of roles, such that a bidder may not at the same time also be a requester.

Initial configuration Finally, there is the matter of setting up the initial state of the institution by defining some facts that should be present in the state at the start. The elements of an `initially` clause can be any fluent defined in the institutional specification, possibly decorated with permission or power:

```
initially perm(iename1(A,B)),
```

```
pow(iename1(A,B)),
fname1(alice,charlie);
```

Notice that parameters here (as elsewhere) can be variables or constants. In the case of the permission and power fluents above, this means that `iename1` is permitted and empowered for all `A` and for all `B`. On the other hand, `fname1` is a grounded fact relating `alice` and `charline`. Lines 10–14 of Figure 6 illustrate both such uses of `initially`, establishing permission for any agent and any request for tender, while the latter statement declares the roles associated with the given names.

The institution interaction language The second part of the language section address how events and states in one institution can be used to affect another institution. The individual institutions are completely independent of one another and contain no specification dependencies: the connections are established entirely through a bridge institution, which is the sole repository of dependency information in the form of the names of the events and fluents to which the bridge specification refers.

Dependencies take two forms: events and fluents. In the first case, the purpose of inter-institutional interaction is for an institutionally recognized event in one institution to cause an exogenous event in another (or several others), subject to some condition:

```
iename1(A,B) xgenerates ename5(A,B) if fname1(A,B);
```

where `iename1` is an institutional event in one institution and `ename5` is an exogenous event defined in another, while `fname1` is present in the state of one of the constituent institutions. The condition part obviously generalizes to permit the combination of fluents in any of the constituent institutions. Lines 1–8 in Figure 7 show how events in one phase of the tender scenario generate events in other phases, effecting a form of synchronization in this case.

In the second case, an (institutional) event in one institution is used to add to or remove facts from the state of another (or several others), subject to some condition:

```
iename1(A,B) xinitiates fname2(A,B) if fname3(A,B);
iename1(A,B) xterminates fname2(A,B) if fname3(A,B);
```

where `iename1` is an (institutional) event in one institution and `fname2` is created/deleted in another, subject to the presence of `fname3` in another institution – or indeed a condition over facts in the state of several others. Lines 9–17 in Figure 7 show how the presence of facts in an institution modelling one phase of the tendering scenario initiates facts in other consituent institutions, subject to a range of conditions.

The third aspect of institutional interaction are the "cross" power fluents which declare that (i) an exogenous event may be generated in one institution for recognition by another (ii) a fluent may be initiated/terminated in one institution subject to an action in another. The `gpow` and `ipow` fluents are used to establish a relationship between two institutions with respect to a given event or fluent, respectively:

```
cross fluent gpow(Inst, ename(A), Inst);
initially gpow(instA, ename(A), instB);
cross fluent ipow(Inst, fname(A), Inst);
```

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 exogenous event in another. Here we see several such rules from the tendering scenario that link the four constituent institutions (see also Figure 2):

```
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 coordinate institutions is achieved by specifying cross-consequence rules, linking the presense 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. 7. Example Inst*AL* bridge specification for the tendering scenario

```
initially ipow(instA, fname(A), instB);
```

where the `gpow` declaration establishes that `ename` is subject to cross generation and the term on the following line instantiates the relation for `ename` between the institutions `instA` and `instB`. Similarly, the `ipow` declaration does the same for cross initiation/termination, which is then instantiated for two specific institutions in the `initially` statement.

JP: 20140616: got here

3 Tools and Platform

Describe the different tools that your framework contains (those developed for or as part of the framework, as well as those that are assumed to be in place for your models to work (e.g FIPA, JADE, JASON, ...)

- Specification language: ???
- Model checker: clingo
- Syntactic validator: pyinstal
- Monitoring tools: clingo
- Run-time interpreter: clingo
- instQL
- JASON, Second Life, SUMO

JP: restructure: this is not right for us

And those technologies or platforms that may be used or have to be used in order to run your models:

- Agent communication languages
- Agent programming languages
- Environments

discuss BSF?

4 This Framework in use

Make a brief description of how the use case is approached with your framework showing what are the characteristic features of your framework used to model and implement the use case.

Event identification, state identification

4.1 Modelling and Implementation

You may want to include the following:

- describe the process you follow when modelling a system
- how the main components of the conceptual framework are involved in the model
- How is the model implemented

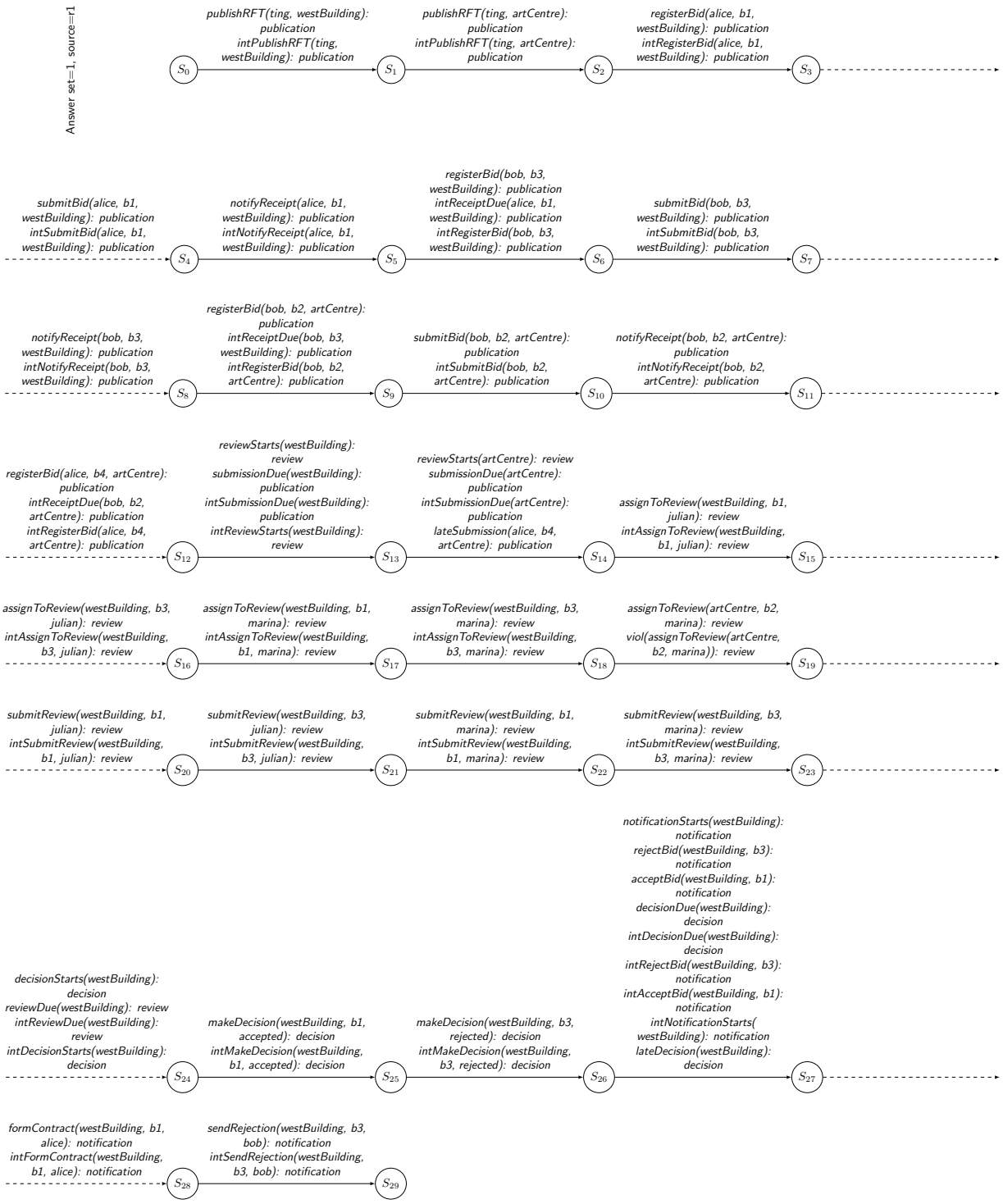

Fig. 8. First cut at trace: needs trimming/selecting for legibility

4.2 Discussion

You may want to include the following:

- Underline those aspects of the use case that you decided to stress in order to show the better qualities of your framework .
- what are the aspects of the use case that you find inadequate or insufficient to demonstrate your framework or how you have elaborated the use case described in the appendix in order to better demonstrate your framework.
- Discuss any variants of the framework you decided to use or alternative modelling that could illustrate your framework better.

5 Critical Assessment

Maybe:

- What is good about your framework
- What you find to be its strongest points
- what is difficult to do with your framework and why.
- What tricks you do in order to do those things you came to realise that your framework should do but you hadn't thought about.
- What types of systems is your framework better-suited to be applied (again).
- Future work

Future work conflict detection and revision
norms by example

6 Key references

An annotated bibliography
[10] formal and computational model
[12] extension to multiple institutions

References

1. UK Highways Agency. Traffic flow database system. Accessible via `https:/trads.hatris.co.uk`. accessed 26th Jan 2014.
2. Huib Aldewereld and Virginia Dignum. Operetta: Organization-oriented development environment. In Mehdi Dastani, Amal El Fallah-Seghrouchni, Jomi Hübner, and João Leite, editors, *LADS*, volume 6822 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2010.
3. Duangtida Athakravi, Domenico Corapi, Alessandra Russo, Marina De Vos, Julian A. Padget, and Ken Satoh. Handling change in normative specifications. In Matteo Baldoni, Louise A. Dennis, Viviana Mascardi, and Wamberto Vasconcelos, editors, *DALT*, volume 7784 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2012.
4. Vincent Baines and Julian Padget. On the benefit of collective norms for autonomous vehicles. Presented at Agents in Traffic and Transportation workshop, AAMAS, May 2014.

5. Tina Balke, Marina De Vos, and Julian Padget. 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.
6. Tina Balke, Marina De Vos, and Julian Padget. I-ABM: combining institutional frameworks and agent-based modelling for the design of enforcement policies. *Artificial Intelligence and Law*, pages 371–398, 2013.
7. Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. CUP, 2003.
8. Gideon Bibu, Nobkazu Yoshioka, and Julian Padget. 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.
9. Owen Cliffe. *Specifying and Analysing Institutions in Multi-agent Systems Using Answer Set Programming*. PhD thesis, University of Bath, 2007.
10. Owen Cliffe, Marina De Vos, and Julian Padget. Answer set programming for representing and reasoning about virtual institutions. In Katsumi Inoue, Ken Satoh, and Francesca Toni, editors, *CLIMA VII*, volume 4371 of *Lecture Notes in Computer Science*, pages 60–79. Springer, 2006.
11. Owen Cliffe, Marina De Vos, and Julian Padget. Embedding landmarks and scenes in a computational model of institutions. In Jaime Simão Sichman, Julian Padget, Sascha Ossowski, and Pablo Noriega, editors, *COIN*, volume 4870 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2007.
12. Owen Cliffe, Marina De Vos, and Julian Padget. Specifying and reasoning about multiple institutions. In Javier Vazquez-Salceda and Pablo Noriega, 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. Domenico Corapi, Alessandra Russo, Marina De Vos, Julian Padget, and Ken Satoh. 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. Kewei Duan, Julian Padget, and H. Alicia Kim. A light-weight framework for bridge-building from desktop to cloud. In Alessio Lomuscio, Surya Nepal, Fabio Patrizi, Boualem Benatallah, and Ivona Brandic, editors, *ICSOC Workshops*, volume 8377 of *Lecture Notes in Computer Science*, pages 308–323. Springer, 2013.
15. M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124, 2011.
16. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365–386, 1991.
17. Michael Gelfond and Vladimir Lifschitz. Action languages. *Electron. Trans. Artif. Intell.*, 2:193–210, 1998.
18. Luke Hopton, Owen Cliffe, Marina De Vos, and Julian A. Padget. Inst*ql*: A query language for virtual institutions using answer set programming. In Jürgen Dix, Michael Fisher, and Peter Novák, editors, *CLIMA*, volume 6214 of *Lecture Notes in Computer Science*, pages 102–121. Springer, 2009.
19. John R. Searle. *The Construction of Social Reality*. Allen Lane, The Penguin Press, 1995.
20. A.J.I. Jones and M. Sergot. A formal characterisation of institutionalised power. *Logic Journal of IGPL*, 4(3):427–443, 1996.
21. Andrew J. I. Jones and Marek J. Sergot. A formal characterisation of institutionalised power. *Logic Journal of the IGPL*, 4(3):427–443, 1996.
22. Robert A. Kowalski and Marek J. Sergot. A logic-based calculus of events. *New Generation Comput.*, 4(1):67–95, 1986.

23. Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. 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.
24. Sanjeev Kumar, Marcus J Huber, Philip R Cohen, and David R McGee. Toward a formalism for conversation protocols using joint inte ntion theory. *Computational Intelligence*, 18(2):174–228, 2002. doi:10.1111/1467-8640.00187.
25. JeeHang Lee, Tingting Li, and Julian Padget. Towards polite virtual agents using social reasoning techniques. *Computer Animation and Virtual Worlds*, 24(3-4):335–343, 2013.
26. Tingting Li, Tina Balke, Marina De Vos, Julian Padget, and Ken Satoh. Legal conflict detection in interacting legal systems. In *Legal Knowledge and Information Systems - JURIX 2013*, volume 259 of *Frontiers in Artificial Intelligence and Applications*, pages 107–116. IOS Press, 2013.
27. Tingting Li, Tina Balke, Marina De Vos, Julian A. Padget, and Ken Satoh. A model-based approach to the automatic revision of secondary legislation. In Enrico Francesconi and Bart Verheij, editors, *ICAIL*, pages 202–206. ACM, 2013.
28. Tingting Li, Tina Balke, Marina De Vos, Julian Padget, and Ken Satoh. Legal conflict detection in interacting legal systems. In Kevin D. Ashley, editor, *JURIX*, volume 259 of *Frontiers in Artificial Intelligence and Applications*, pages 107–116. IOS Press, 2013.
29. Tingting Li, Tina Balke, Marina De Vos, Julian Padget, and Ken Satoh. Legal conflict detection in interacting legal systems (jurix 2013). In Kevin D. Ashley, editor, *Legal Knowledge and Information Systems*, volume 259 of *Frontiers in Artificial Intelligence and Applications*, pages 107–116. IOS Press, 2013. DOI 10.3233/978-1-61499-359-9-107, ISBN 978-1-61499-358-2 (print) — 978-1-61499-359-9 (online).
30. I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal LP. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programing and Nonmonotonic Reasoning*, volume 1265 of *LNAI*, pages 420–429, Berlin, July 28–31 1997. Springer.
31. Javier Pinto and Raymond Reiter. Reasoning about time in the situation calculus. *Ann. Math. Artif. Intell.*, 14(2-4):251–268, 1995.
32. Bastin Tony Roy Savarimuthu, Julian Padget, and Maryam Purvis. Social norm recommendation for virtual agent societies. In Guido Boella, Edith Elkind, Bastin Tony Roy Savarimuthu, Frank Dignum, and Martin K. Purvis, editors, *PRIMA*, volume 8291 of *Lecture Notes in Computer Science*, pages 308–323. Springer, 2013.
33. John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.
34. I.A. Smith, P.R. Cohen, J.M. Bradshaw, M. Greaves, and H. Holmback. Designing conversation policies using joint intention theory. In *Proceedings of International Conference on Multi Agent Systems*, pages 269–276, 1998. doi:10.1109/ICMAS.1998.699064.
35. Marina De Vos, Tina Balke, and Ken Satoh. Combining event-and state-based norms. In Maria L. Gini, Onn Shehory, Takayuki Ito, and Catholijn M. Jonker, editors, *AAMAS*, pages 1157–1158. IFAAMAS, 2013.