On Requirements Representation and Reasoning using Answer Set Programming

Julian Padget and Emad Eldeen Elakehal
Department of Computer Science
University of Bath, UK
jap@cs.bath.ac.uk, emad@bookdepository.co.uk

Ken Satoh
National Institute of Informatics
and Sokendai, Japan
ksatoh@nii.ac.jp

Fuyuki Ishikawa
National Institute of Informatics,
Japan
f-ishikawa@nii.ac.jp

***Abstract*—We describe an approach to the representation of requirements using answer set programming and how this leads to a vision for the role of artificial intelligence techniques in software engineering with a particular focus on adaptive business systems. We outline how the approach has developed over several years through a combination of commercial software development and artificial intelligence research, resulting in: (i) a metamodel that incorporates the notion of runtime requirements, (ii) a formal language for their representation and its supporting computational model (Inst*AL*), and (iii) a software architecture that enables monitoring of distributed systems. The metamodel is the result of several years experience in the development of business systems for e-tailing, while Inst*AL* and the runtime monitor is on-going research to support the specification, verification and application of normative frameworks in distributed intelligent systems. Our approach derives from the view that in order to build agile systems, the components need to be structured more like software that controls robots, in that it is designed to be relatively resilient in the face of a non-deterministic, dynamic, complex environment about which there is incomplete information. Thus, degrees of autonomy become a strength and an opportunity, but must somehow be constrained by informing these autonomous components what should be done in a certain situation or what system state ought to be achieved through norms as expressions of requirements. Because such a system made up of autonomous components is potentially behaviourally complex and not just complicated, it becomes essential to monitor both whether norms/requirements are being fulfilled and if not why not. Finally, because control over the system can be expressed through requirements in the form of data that can be changed, a route is opened to adjustment and dynamic re-direction of running systems.**

I. Introduction

Nearly two decades ago, Fickas and Feather [1] put forward a case for requirements being represented in some way in a live system as a way to provide continuous validation and to help with its modification when re-deployed in an environment different from the original. However, the implementation did depend on the insertion of code into a running system (sic), which is delicate and may not always be acceptable. Furthermore, the approach reported only provides indicators of non-compliance, that require human interpretation, code revision and then patching. Chen et al. [2] demonstrate, in carefully defined circumstances, the use of dynamic patching.

More recently, a research agenda has started emerging in pursuit of self-adaption arising from runtime requirements awareness [3], driven in part by the demands of adaptive systems [4] and illustrated by the possibilities offered by techniques such as argumentation [5]. A key aspect of this theme is the reference to "reasoning" about requirements, to account for different deployments of the same system in diverse environments, possibly even leading to new requirements. This approach is underpinned by a desire for requirements as run-time entities so they may be processed by the system in respect of itself; a notion referred to as requirements reflection [6].

The aim of this paper is to outline a particular approach to requirements representation and reasoning that uses the technology of answer sets (see Section IV) to address aspects (i), (iv) and (v) of the agenda set out in [3], which for the sake of making this paper self-contained, we reproduce here: (i) run-time representations of requirements (ii) evolution of the requirements model and its synchronization with the architecture (iii) dealing with uncertainty (iv) multi-objective decision-making (v) self-explanation.

The central element of our approach is the declarative institutional action language (Inst*AL*) [7], for the specification of collections of interacting normative systems[1], capturing regulations, system states and interactions between different normative sub-models in the form of what should and ought to be so (the deontic modalities). Our contention is that norms are statements about the behaviour of system components, or the state of the system as a whole. As such, they have a clear correspondence with the concept of requirement, which also captures what should and ought to be so with regard to system behaviour, so our objective is to explore the adequacy of the language for this purpose. Inst*AL* has similarities to the Event Calculus [8], but offers two novel features: one theoretical and one technical. The theoretical is the explicit distinction between exogenous events and institutional events, which both makes clear when an external world action "counts-as" [9] as a valid institutional 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 serves as:

[1]We follow the convention in the literature of using institutional and normative framework interchangeably. Note that institution is used to refer to a collection of norms, where a norm is either *regulative*, being a rule that describes how the institutional state changes when an event occurs, or *constitutive*, that describes an institutional state through a condition over the terms of the institutional state.

978-1-4799-6354-6/14/$31.00 © 2014 IEEE

AIRE 2014, Karlskrona, Sweden

(i) a model to capture and validate (exhaustively, over a finite horizon) requirements as part of a design process, (ii) a runtime requirements monitoring mechanism, by evaluating the model one step at a time, and (iii) a source of (re-)direction for amenable software components through the delivery of obligations that inform of the proper action to take.

A complementary and contemporaneous line of research (Section II) begins by exploring the development of adaptive business systems, in which requirements for adaptivity are implemented concretely, providing the lessons from which to abstract, leading to a metamodel [10] (Section III). The purpose of these two sections is to illustrate that the work on adaptation is grounded in actual business needs and then how the patterns observed, combined with concepts from artificial intelligence, lead to a first version metamodel. Here, we have chosen to discuss this before the details of the specification language (Section IV), in order to provide a motivating context for making the connection between the two activities.

The third and final element presented here (Section V) is how to make the institutional framework available through conventional interfaces in the context of distributed (intelligent) systems. The fundamental unit of interaction for the institutional model is the event, hence we take a notification-based approach realized through a publish/subscribe architecture [11] connected to an institution manager.

We conclude with a brief resume of the main points and some lines for future research, while related work, because of the discursive nature of this kind of paper, is distributed throughout, citing and discussing where appropriate, rather than as a tightly focussed devoted section.

II. Adaptive Business Systems

The work we describe here brings together: (i) practical commercial experience from the development of software-based processes in which business agility is a necessity to adapt goods and services to meet changing customer needs and dynamic market conditions, and (ii) research in artificial intelligence on representation and reasoning about rules governing component behaviour and (un)desirable system states.

The first business process example concerns the application of agents to the handling of catalogue and stock-control for the selling of books on the internet [12] at The Book Depository (TBD). TBD's market is characterised by (very) low volumes over a (very) large number of items, so that agility and extremely low overheads are essential to the business model. The requirements are to: (i) provide (and monitor) a continuously available system (was 12,000 orders/day in 2008, now much higher – and running ever since) (ii) handle the variety of data formats used by the different service providers (iii) handle regular catalogue updates. The system design was influenced by notions of normative modelling [13] and self-management [14]. The software components were implemented as reactive agents using the Agentscape platform [15], in which each was given a complete plan for the task and each agent monitored those with which it interacted for signs of anomalous behaviour, that is requirements monitoring. Thus, although the monitoring action was effectively hard-wired, the behaviour at any one time was data-directed, its construction having been factored out into an administrator component. The whole design process was supported by the Prometheus methodology [16]. What we learned from this process was: (i) how normative modelling assists in realizing business agility – although regimentation of components via the communication of task plans does not in general provide sufficient resilience for handling changes without recourse to the administrator component – and (ii) the importance of self-monitoring in early identification of incorrect system behaviour and in reducing the overhead for human operators.

The second example considers the pricing subsystem at TBD [17], which is again a continuously running system (since mid-2008) and spread across many servers (in order to be able to handle several millions of ISBNs in a 24hr period). Its task is to collect market data and apply in real-time one of a set of predefined pricing patterns in response to: (i) the behaviour of other participants in that marketing channel (ie. direct competitors for that specific good) (ii) activity in other channels globally that impact stock levels (and hence the capacity to fulfil an order) (iii) numerous detail factors that affect the 'bottom line', which are discussed in detail in [17]. Beyond such good-specific factors, such dynamic posted-price mechanisms also have to reflect higher level performance indicators, such as maximising sales volume, minimising time from advertisement to sale, maximising profit or increasing the company market share.

As with the first example, the complexity and dynamic nature of the environment – as well as the positive experience from the stock-control system – resulted in the adoption of an agent-based approach, supported by an institutional-style infrastructure, that captures the rules governing component behaviour and the monitoring and enforcement of these rules. This experience reinforced earlier points about business agility and self-monitoring, but also demonstrated scalability and responsiveness. However, the components remain regimented, under the control of the Book Depository pricing engine.

The experience gained from the development of the above lead to a desire to simplify the creation of such systems through high level specification of business processes, governed by normative frameworks to capture business requirements. We outline the results of that work in the next section.

III. A Metamodel with Institutions

The positive experiences of using multiagent systems for business applications reported in the previous section led us to consider how to make the technology more accessible for software developers and business process designers. Although it is not the only interesting property, the capacity to engineer self-management was chosen as the main focus and influenced the supporting metamodel. The three relevant features of the MSMAS methodology are: (i) a formal specification mechanism for system norms, being the means to express both business rules and system integrity constraints (ii) support for the software reflection of the physical organizational structure,

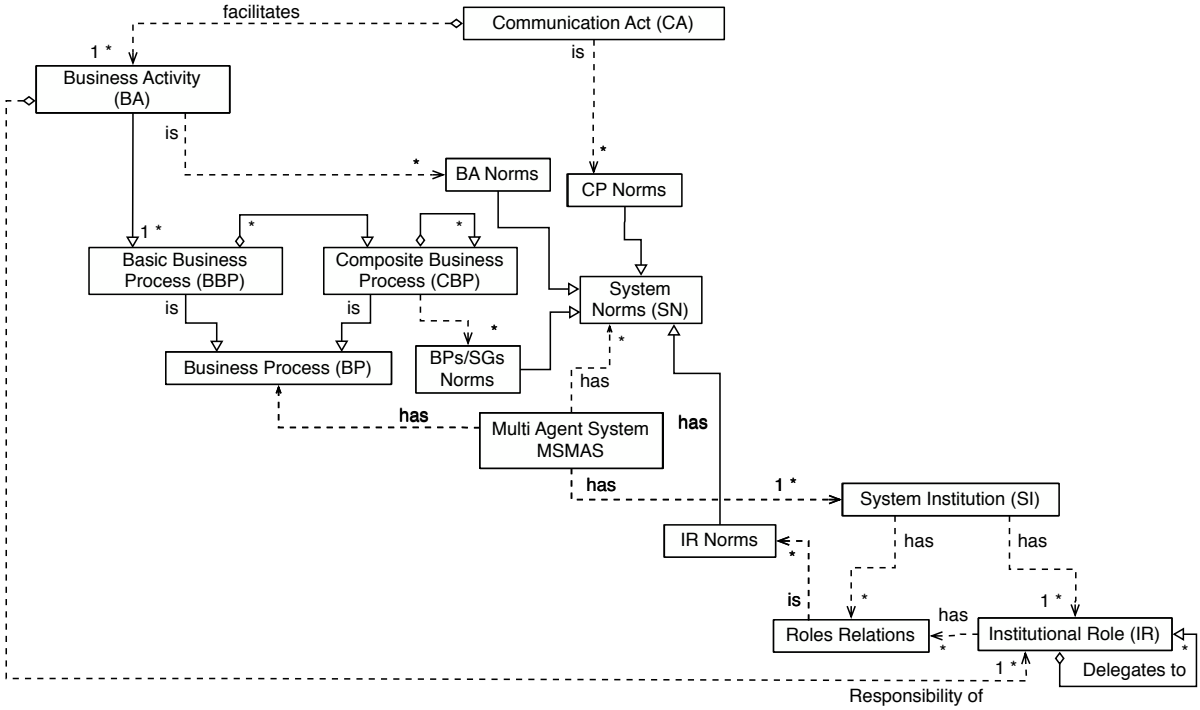

Fig. 1. MSMAS Metamodel: System Norms [18]

through the use of institutions, and (iii) explicit support for self-management through dynamic planning.

Jennings's [19] Commitments and Conventions Hypothesis states that all coordination mechanisms can ultimately be reduced to (join) commitments and their associated (social) conventions. Conventions do not enforce behaviour, but they do introduce an element of constraint, depending on the chances of detection of non-compliance and on the severity of the consequent sanction. How can observation of conventions be added to component behaviour? The straightforward, and from our point of view unacceptable, approach is to hard code them. A slightly less restrictive solution, called regimentation [20], is to enforce them at the protocol level so that violation is impossible. The third mechanism, called regulation [21], relies purely upon norms to provide behavioural direction. This is the true purpose of norms and institutions, operating in conjunction with suitably implemented system components.

MSMAS uses a declarative approach to modelling business processes, borrowing from DecSerFlow [22], which offers an effective way to describe loosely-coupled processes. As a result, unlike conventional graph-oriented workflow languages, which concentrate the designer's attention on "how", the focus shifts up a level to "what", in which constraints are added to the activity model, along with rules to be observed at run-time. The underlying constraint specification language is based on Linear Temporal Logic (LTL), but the ConDec^{++} language [23] provides a visual notation to express the relationships between two (or more) activities and any desired preconditions. A selected comparison with other methodologies appears in [10] and a more comprehensive one in [18].

While the above offers a diagrammatic notation, with an underlying formalization, for the capture of business processes and their coordination, there remains the matter of the superstructure into which those activities fit, namely the MSMAS metamodel. An overview of the main components and their relations is given in Figure 1.

Capturing the designer's intent is one part of the story, but it may be incomplete or incorrect, which is why verification is an increasingly important part of designing and building software systems. In [24], $\mathcal{S}$CIFF is used as the verification tool, checking formal descriptions that are automatically generated from the graphical notation. A forthcoming project, using the metamodel, will implement an export procedure to generate Inst*AL* so that we can interface to the framework and tools described in the next section.

IV. The Inst*AL* Language and Framework

This section provides some context for the Inst*AL* framework, leading into a conceptual description of what institutional modelling is intended to achieve, an overview of the (single institution) formal model, and how AnsProlog, as the computational model, helps to realize those goals. We conclude this section with an example specification in Inst*AL* .

The aim of Inst*AL* is to allow the expression of norms, where the notion of norm derives from the modalities of deontic logic to express what should and ought to be so, as statements of obligations, prohibitions and permissions, that can be associated with actions or states. The Inst*AL* language

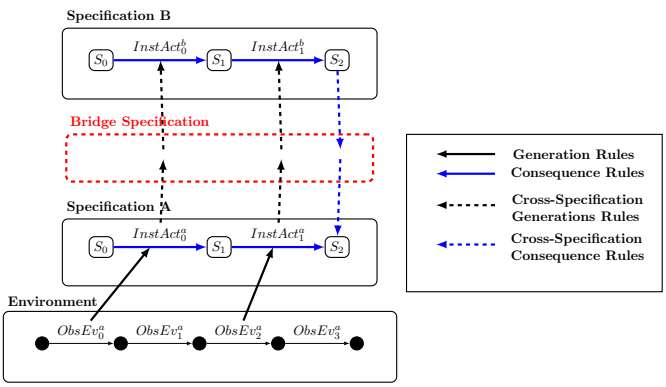

Fig. 2. Cross-Specification Generation and Consequence Functions [25]

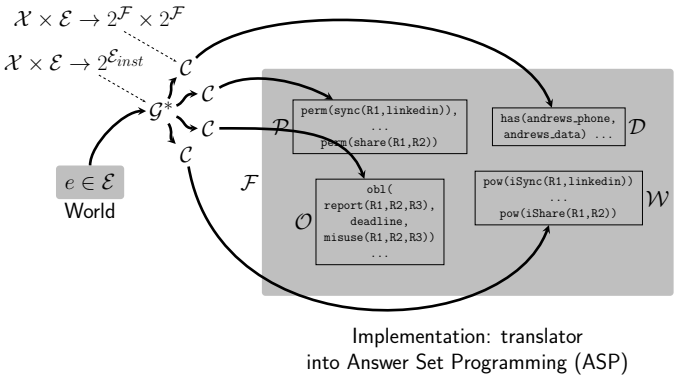

Fig. 3. Outline of formal model for a single institution

supports the description of obligations, such that (i) an event must occur or some (model) state must be achieved (ii) by some deadline, specified either as another event or some other (model) state (iii) or a violation (event) occurs.

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. Benefits of ASP include forward and backward reasoning and planning, all using the same program. On-line reasoning changes very little, because it is a matter of running the model for a single time step, then examining the resulting answer set for obligations and communicating them to the relevant software components (see the example deployment in Figure 7 and discussion in Section V-D).

An Inst*AL* model is a collection of interacting institutions that individually define a collection of event- and state-based norms. Figure 2 helps illustrate the role and situation of the institutional models, in that events are observed in an environment in which the application is running, which may then be recognised by one of the institutional specifications and perhaps propagated to another (the (cross-)$\mathcal{G}$eneration relation), leading directly to changes in that institutional state and perhaps indirectly (via the bridge specification) to changes in another (the (cross-)$\mathcal{C}$onsequence relation). The Inst*AL* formal and computational model supports three forms of institutional collections which are characterised as follows:

1) The institutions function 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 [26].

The formal model has been described at length in several publications of which [27] is representative for the single institutional model. The key elements are outlined in Figure 3, in which there is a set of facts that describe the current state – denoted $\mathcal{X}$ – of the institution and a pair of relations, whose actions are triggered by events ($\mathcal{E}$):

1) We distinguish between external events and institutional events ($\mathcal{E}_{inst}$): the former are not controlled by the institution and act as triggers for institutional action; the latter can have permission ($\mathcal{P}$) and power ($\mathcal{W}$) associated with them to reflect whether an action is allowed at some time and whether the action (by some actor) has any institutional effect[2], respectively.
2) Institutional facts comprise the union of facts about the $\mathcal{D}$omain being modelled, $\mathcal{P}$ermissions and po$\mathcal{W}$ers [9] associated with actions and $\mathcal{O}$bligations arising from actions. If not permitted, an action is implicitly prohibited. The occurrence of an event that is not permitted leads to a violation, while an event that is not empowered simply has no effect (on the institutional state). Obligations indicate some event that should happen or some state that should be reached, a deadline for that event/state, expressed as another event or state and a violation event that should occur if the deadline is not met. We note that an institution is a passive entity and that the detection and handling of violations and obligations are the responsibility of the participants in the institution; the sole purpose of the framework is to observe and maintain (through its progression rules) the model state pertaining to the particular perspective on events for which the institution has been designed.
3) The generation relation $\mathcal{G}$, whose domain is an event ($\mathcal{E}$) and the current state ($\mathcal{X}$) of the institution – so that processing is conditional upon the current state – and range is the set of events defined by the institution; hence, depending on context, a real world action is recognized as an institutional action (counts-as). Initially, $\mathcal{G}$ is invoked with an external event: if that event is recognized by the

[2]For example: it has meaning if the chair of a meeting says that business is finished, but is meaningless if someone other than the chair says so.

institution, it may generate an institutional event, which by the repeated application of $\mathcal{G}$ may in turn generate another and another, all of which are dependent on the current institutional state.

4) The consequence relation $\mathcal{C}$, whose domain is as for $\mathcal{G}$ and range is (i) the set of facts ($\mathcal{F}$) for addition to, and (ii) the set of facts ($\mathcal{F}$) for deletion from the institutional state. $\mathcal{C}$ is invoked for each of the events generated so firing any associated rule for the addition and/or deletion of (so-called inertial) facts from the institutional state.
5) Additionally, there are non-inertial facts, that are true only while some condition over the institutional state is true. These permit the recognition of situations whose constituent facts may be initiated (or terminated) by events at different points in time.

As a result, the single external event and all the institutional events are treated as occurring simultaneously and having a simultaneous effect on the institutional state (meaning the order in which events are generated within a single update has no effect).

The formal model for collections of institutions augments the above with extensions of the generation and consequence relations that cross institutions and adds powers for one institution to create an event in another and for one institution to initiate and terminate fluents[3] in another. Illustrations of collections of institutions and an exposition of the formalization of the revision of specifications through machine learning appears in [26], [25].

The computational model is realised through translation to Answer Set Prolog (referred to AnsProlog or ASP [28]), a logic programming language under the answer set semantics (ASP) [29] and subsequently the use of an answer set solver, such as Clingo [30]. The central idea is that the formal mathematical model is represented as an answer set program, combined with the supporting code and the query program (a set of constraints), resulting in the answer sets.

A. An Example InstAL Specification

The social networking scenario is a small, artificial scenario constructed for the purpose of illustrating the use of the language and the supporting tools[4]. The aim is to show it is possible to analyse the interaction of (smart)phones and social network platforms (SNPs) to explore how it might be used to evaluate risks to personal data.

The specification (see the `initially` section at the bottom of Figure 4) has three phones (Andrew's, Mark's and Roger's), three 'infospheres' (phone, Facebook and LinkedIn) and expresses some 'friend' relationships and some consent attributes. The friend relationship is what permits one handset to send data to another. However, consent moves with the data (like a sticky policy [31]) and expresses whether the data's owner has given access permission to the specified social networking platform. In addition to the `send` event, which is initiated by a handset user, there are two other actions that progress the scenario: `sync`, which is when a handset (autonomously) synchronizes with one of the social networking platforms and `share`, which indicates the sharing of data between social networking platforms (see Figure 5).

The purpose of the specification is to provide a model that, through the application of an answer set solver, can be used to identify situations in which personal data is vulnerable

```
institution sns;

type Resource;

noninertial fluent vulnerable(Resource,Resource);
noninertial fluent warning(Resource,Resource);
noninertial fluent situation1(Resource,Resource);
noninertial fluent situation2(Resource,Resource);

fluent typeOf(Resource,Resource);
fluent subclassOf(Resource,Resource);
fluent has(Resource,Resource);
fluent friend(Resource,Resource);
fluent consent(Resource,Resource);

exogenous event send(Resource,Resource);
inst      event iSend(Resource,Resource);
exogenous event sync(Resource,Resource);
inst      event iSync(Resource,Resource);

perm(send(R1,R2)) when has(R3,R1), friend(R3,R2);
send(R1,R2) generates iSend(R1,R2);
iSend(R1,R2) initiates has(R2,R1);

perm(iSend(R1,R2)) when has(R3,R1), friend(R3,R2);
pow(iSend(R1,R2)) when has(R3,R1), friend(R3,R2);

vulnerable(R,P) when situation1(R,P);
situation1(R2,facebook) when
   typeOf(R1,phone), typeOf(R2,data),
   has(R1,R2), has(R1,facebook_app),
   not consent(R2,facebook);
situation1(R2,linkedin) when
   typeOf(R1,phone), typeOf(R2,data),
   has(R1,R2), has(R1,linkedin_app),
   not consent(R2,linkedin);

warning(R,P) when situation2(R,P);
situation2(R1,facebook) when
   typeOf(R1,data), has(R2,R1),
   friend(R2,R3), has(R3,facebook_app),
   not consent(R1,facebook);
situation2(R1,linkedin) when
   typeOf(R1,data), has(R2,R1),
   friend(R2,R3), has(R3,linkedin_app),
   not consent(R1,linkedin);

initially

   has(andrews_phone,andrews_data),
   has(marks_phone,marks_data),
   has(rodgers_phone,rodgers_data),
   has(marks_phone,facebook_app),
   has(marks_phone,linkedin_app),

   friend(andrews_phone,marks_phone),
   friend(marks_phone,andrews_phone),
   friend(rodgers_phone,marks_phone),

   consent(marks_data,facebook),
   consent(marks_data,linkedin),
   consent(andrews_data,linkedin),
   consent(rodgers_data,facebook)
;
```

Fig. 4. Initial part of specification of the Social Networking Scenario, identifying degrees of risk to personal data on handsets

[3] A fluent is true when present and false when absent.

[4] This scenario was presented at JURISIN 2013, and at the Shonan workshop on privacy and transparency, but has not been published.

```
sync(R1,R2) generates iSync(R1,R2);
iSync(R1,R2) initiates has(R2,R3) if
   typeOf(R1,phone), typeOf(R2,platform),
   has(R1,R3),typeOf(R3,data),
   consent(R3,R2);
noninertial fluent compromised(Resource);

compromised(R1,R2) when
   has(R2,R1), not consent(R1,R2),
   typeOf(R1,data), typeOf(R2,platform);

exogenous event share(Resource,Resource);
inst      event iShare(Resource,Resource);
share(R1,R2) generates iShare(R1,R2);
iShare(R1,R2) initiates has(R2,R3) if
   typeOf(R1,platform), typeOf(R2,platform),
   has(R1,R3),typeOf(R3,data);

noninertial fluent situation3(Resource,Resource);

vulnerable(R,P) when situation3(R,P);
situation3(R1,R3) when
   has(R2,R1), typeOf(R1,data), typeOf(R2,platform),
   consent(R1,R2),
   typeOf(R3,platform), not consent(R1,R3);
```

Fig. 5. Continuation of specification of the Social Networking Scenario, identifying degrees of risk to personal data on handsets

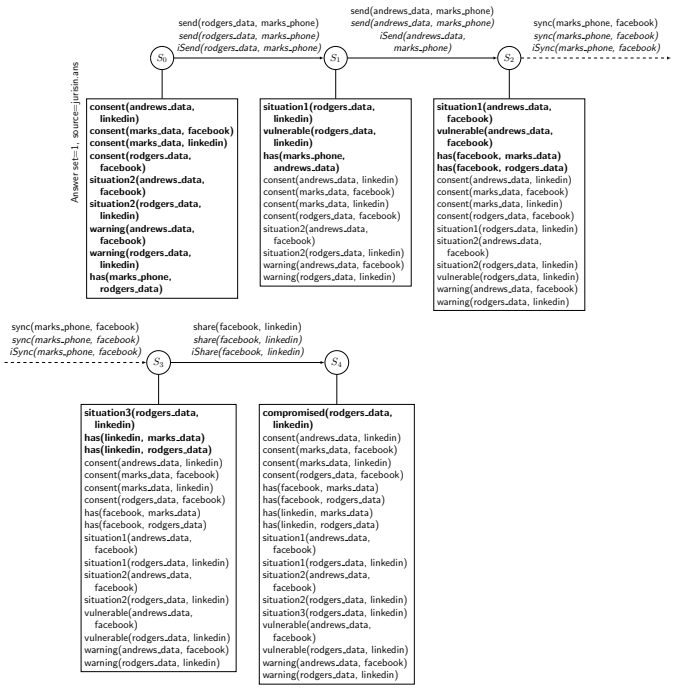

Fig. 6. The phone data model trace

before it is compromised, as well as situations in which it is compromised (in particular in the absence of consent, since this may constitute a breach of contract and consequently a potential legal violation). Consequently, the model has three categories of risk to personal data: (i) `warning` when a friend handset has a social networking app (ii) `vulnerable` when data resides on the same handset as a social networking app for which consent has not been given, and (iii) `compromised` when data is held by a social networking platform for which consent has not been given. These constitute three conditions that the designer would like to be able to monitor. The model is used here by presenting it with a particular trace (the four events that appear in Figure 6 and below) to confirm these conditions are identified. This illustrates a scenario of escalating risk through a sequence of events involving sending, syncing and sharing, that leads to the unconsented holding of data by Facebook and LinkedIn, using this query program:

```
1  observed(send(rodgers_data,marks_phone),0).
2  observed(send(andrews_data,marks_phone),1).
3  observed(sync(marks_phone,facebook),2).
4  observed(share(facebook,linkedin),3).
```

This query has the effect of causing the generation of answer sets with only the above events occurring at time instants 0 through 3. When this is combined with the specification in Figures 4–5, the result is a single answer set (since a specific ordering of events was given), rendered in Figure 6.

This section has served to provide an overview of the Inst*AL* framework, its underlying formal model and illustrate how it can be used as a design tool with a simple example of the visualization of a trace arising from a particular sequence of events. In the next section, we discuss how it can be used for the tasks of monitoring compliance at run-time and providing direction to software components in running systems.

V. Monitoring Requirements

This section discusses three ways in which a specification can be used for different purposes using the Inst*AL* framework, depending on whether the objective is model design or deployment and, if the latter, whether the objective is system monitoring or component (re-)direction. This is followed by a brief example application.

A. Model-Checker

Inst*AL* was firstly developed a language for specifying and model-checking institutional specifications, for which it depends upon the computation of answer sets that represent model traces. Answer set solvers output answer sets in a textual form as sequence of literals, of which there can be lots, depending on the length of the trace and as many kinds as the program defines. In addition, without constraint, the solver will compute answer sets for as many permutations of events as are specified. To make the output easier to interpret, we have also developed a renderer to allow for the visual inspection of the interplay of events and states. As was illustrated in the preceding section, the query program, when combined with the specification and the solver, serves to 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 checking properties such as: (i) whether events occur in certain sequences or not and (ii) the presence or absence of state traces (answer sets) that satisfy certain conditions. Inst*AL* has been used in this way for contract analysis [32], [25], security requirements [33], [34] and wireless network management [35].

B. Requirements Sensor

Here, Inst*AL* acts as a monitor of the system state, using observations of component actions to progress the institutional model(s). Those actions may result in obligations being added to the system state that are then sent to the client to act upon. In this way, components are informed of the consequences of their actions and can decide whether to comply with the obligation or not. For example, Inst*AL* is used (i) to examine the dynamic properties of wireless network management policy and evaluate enforcement costs in [36], (ii) to guide non-player characters in Second Life in [37] and (iii) to explore mixed vehicle environments using SUMO [38] in [39].

C. Oracle

A 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 requirements compliant (ii) whether an action – or a sequence of actions – results in a state satisfying a particular condition, and (iii) what (requirements-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 self-monitoring and self-adaptive systems.

D. Example Application

The Inst*AL* tools are command line programs[5], that facilitates their integration with other software components. Our aim is to support distributed applications with potentially high communication latencies, so we have adopted a notification protocol. In practice, we use the Extensible Messaging and Presence Protocol (XMPP), an open standard communications protocol [40], for which many widely-used languages already have interfaces, making it relatively straightforward to connect up additional software components when needed. Content takes the form of either Resource Description Format (RDF) triples or JavaScript Object Notation (JSON) structures. In this way we can use Inst*AL* in conjunction with agent-based simulation, virtual environments or live systems.

We have explored several scenarios in the context of the system in Figure 7, in which Belief-Desire-Intention agents control vehicles within a population of vehicles provided by the SUMO [38] simulation environment. In one scenario, we define an institution to handle information regarding upcoming traffic lights, and issuing appropriate obligations to ensure the vehicle arrives at that light when it is green, rather than being held at a red light. The traffic light information is received via RDFs from the Area of Interest module, and where the distance to an upcoming light is between 100m to 300m and that light is red, the institution is updated with the event

[5]The translator and visualizer are written in Python and are available from http://www.cs.bath.ac.uk/instal. The clingo answer set solver is available from http://potassco.sourceforge.net/. The Bath Sensor Framework, which provides distributed communications is available from https://code.google.com/p/bsf/.

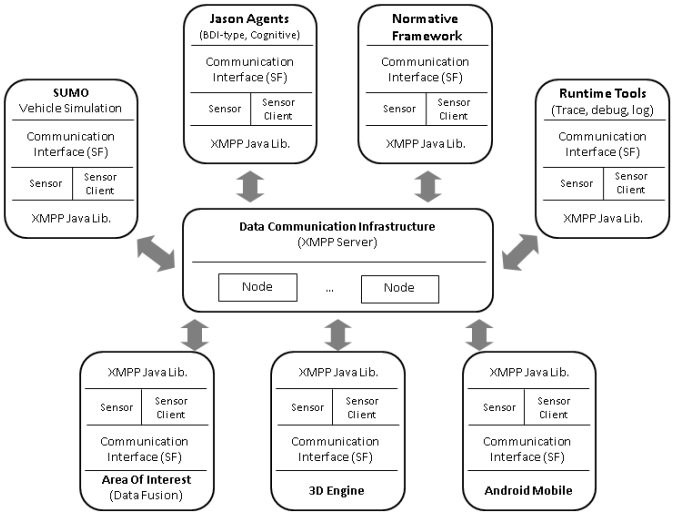

Fig. 7. Bath Sensor Framework (BSF) overview

`upcomingRedLight(Agent)`. This then generates the institutional event `iniOblSlowDown(Agent)`, resulting in the obligation `obl(reduceSpeed(Agent)`, which the Jason agent receives and implements this by reducing its speed for a specified (35 second) period. This allows exploration of the potential impact of such policies on fuel consumption and journey times, as well as on junction congestion [39].

VI. Discussion

Our aim in this paper has been to set out how an approach to knowledge representation and reasoning has the potential to be brought to bear on the challenges raised by "requirements at runtime". Based on experience to date, we believe the approach is suitable for systems comprising numerous autonomous components, whose interactions can be characterised by events in the first instance, but also where potentially complex situations can arise from those interactions, whose detection is a critical part of correct system operation. Furthermore, if the components are appropriately designed, they can be directed through the obligation mechanism, facilitating responses that favour system over individual behaviour. The formalism outlined depends upon a thoroughly researched variety of logic programming that supports both reasoning and planning and which has already received some attention in the RE literature [41]. Complementary to monitoring and reasoning, the concept of obligations arising from institutions, suggests a possible route to adaptive software systems driven by and updating their requirements using formally backed techniques. Finally, we have put forward an initial attempt at a metamodel to connect these ideas, whose requirements are derived from two examples of adaptive business processes.

References

[1] S. Fickas and M. S. Feather, "Requirements monitoring in dynamic environments," in *RE*. IEEE Computer Society, 1995, pp. 140–147.

[2] H. Chen, J. Yu, R. Chen, B. Zang, and P.-C. Yew, "Polus: A powerful live updating system," in *ICSE*. IEEE Computer Society, 2007, pp. 271–281.

[3] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems," in *RE*. IEEE Computer Society, 2010, pp. 95–103.

[4] N. Bencomo, "Requirements for self-adaptation," in *Generative and Transformational Techniques in Software Engineering IV*, ser. Lecture Notes in Computer Science, R. Lmmel, J. Saraiva, and J. Visser, Eds. Springer Berlin Heidelberg, 2013, vol. 7680, pp. 271–296. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35992-7_7

[5] T. T. Tun, A. K. Bandara, B. A. Price, Y. Yu, C. Haley, I. Omoronyia, and B. Nuseibeh, "Privacy arguments: Analysing selective disclosure requirements for mobile applications," *2012 20th IEEE International Requirements Engineering Conference (RE)*, vol. 0, pp. 131–140, 2012.

[6] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier, "Requirements reflection: requirements as runtime entities," in *ICSE (2)*, J. Kramer, J. Bishop, P. T. Devanbu, and S. Uchitel, Eds. ACM, 2010, pp. 199–202.

[7] O. Cliffe, M. De Vos, and J. Padget, "Answer set programming for representing and reasoning about virtual institutions." in *CLIMA VII*, ser. LNCS, K. Inoue, K. Satoh, and F. Toni, Eds., vol. 4371. Springer, 2006, pp. 60–79.

[8] R. A. Kowalski and M. J. Sergot, "A logic-based calculus of events," *New Generation Comput.*, vol. 4, no. 1, pp. 67–95, 1986.

[9] A. J. I. Jones and M. J. Sergot, "A formal characterisation of institutionalised power," *Logic Journal of the IGPL*, vol. 4, no. 3, pp. 427–443, 1996.

[10] E. E. Elakehal and J. Padget, "MSMAS: Modelling Self-managing Multi Agent Systems," *SCPE: Scalable Computing: Practice and Experience*, vol. 13, no. 2, pp. 121–137, 2012.

[11] J. Lee, V. Baines, and J. Padget, "Decoupling cognitive agents and virtual environments," in *CAVE*, ser. LNCS, F. Dignum, C. Brom, K. V. Hindriks, M. D. Beer, and D. Richards, Eds., vol. 7764. Springer, 2012, pp. 17–36.

[12] E. E. Elakehal and J. Padget, "Pan-supplier stock control in a virtual warehouse," in *AAMAS (Industry Track)*, M. Berger, B. Burg, and S. Nishiyama, Eds. IFAAMAS, 2008, pp. 11–18.

[13] J.-A. Rodríguez, P. Noriega, C. Sierra, and J. Padget, "FM96.5 A Java-based Electronic Auction House," in *Proceedings of 2nd Conference on Practical Applications of Intelligent Agents and MultiAgent Technology (PAAM'97)*, London, UK, Apr. 1997, pp. 207–224, iSBN 0-9525554-6-8. [Online]. Available: http://www.iiia.csic.es/Projects/fishmarket/PAAM97.ps.gz

[14] P. V. Roy, "Self management and the future of software design," *Electr. Notes Theor. Comput. Sci.*, vol. 182, pp. 201–217, 2007.

[15] B. J. Overeinder and F. M. T. Brazier, "Scalable middleware environment for agent-based Internet applications," in *Applied Parallel Computing*, ser. LNCS. Berlin: Springer, 2006, vol. 3732, pp. 675–679.

[16] J. Thangarajah, L. Padgham, and M. Winikoff, "Prometheus design tool," in *AAMAS*, F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, Eds. ACM, 2005, pp. 127–128.

[17] E. E. Elakehal and J. Padget, "Market intelligence and price adaptation," in *Proceedings of the 14th Annual International Conference on Electronic Commerce*, ser. ICEC '12. New York, NY, USA: ACM, 2012, pp. 9–16. [Online]. Available: http://doi.acm.org/10.1145/2346536.2346538

[18] E. E. Elakehal, "Modelling self-managing multiagent systems using norms," Ph.D. dissertation, University of Bath, 2014.

[19] N. R. Jennings, "Commitments and conventions: The foundation of coordination in multi-agent systems," *The Knowledge Engineering Review*, vol. 8, no. 3, pp. 223–250, 1993. [Online]. Available: citeseer.ist.psu.edu/jennings93commitments.html

[20] M. Esteva, D. de la Cruz, B. Rosell, J. L. Arcos, J. A. Rodríguez-Aguilar, and G. Cuní, "Engineering open multi-agent systems as electronic institutions," in *AAAI*, D. L. McGuinness and G. Ferguson, Eds. AAAI Press / The MIT Press, 2004, pp. 1010–1011.

[21] V. Dignum and J. Padget, "Multiagent organizations," in *Multiagent Systems*, 2nd ed., G. Weiss, Ed. MIT Press, 2013, pp. 51–98, iSBN 978-0-262-01889-0. http://mitpress.mit.edu/books/multiagent-systems-1 retrieved 20130309.

[22] W. van der Aalst and M. Pesic, "Decserflow: Towards a truly declarative service flow language," in *The Role of Business Processes in Service Oriented Architectures*, ser. Dagstuhl Seminar Proceedings, F. Leymann, W. Reisig, S. R. Thatte, and W. van der Aalst, Eds., no. 06291. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2006/829

[23] M. Montali, *Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach*, ser. LNBIP. Springer, 2010, vol. 56.

[24] E. E. Elakehal, M. Montali, and J. Padget, "Verifying MSMAS model using SCIFF," in *MATES*, ser. LNCS, M. Klusch, M. Thimm, and M. Paprzycki, Eds., vol. 8076. Springer, 2013, pp. 44–58.

[25] T. Li, T. Balke, M. D. Vos, J. Padget, and K. Satoh, "Legal conflict detection in interacting legal systems," in *JURIX*, ser. Frontiers in Artificial Intelligence and Applications, K. D. Ashley, Ed., vol. 259. IOS Press, 2013, pp. 107–116.

[26] T. Li, T. Balke, M. De Vos, J. A. Padget, and K. Satoh, "A model-based approach to the automatic revision of secondary legislation," in *ICAIL*, E. Francesconi and B. Verheij, Eds. ACM, 2013, pp. 202–206.

[27] O. Cliffe, M. D. Vos, and J. A. Padget, "Modelling normative frameworks using answer set programing," in *LPNMR*, ser. LNCS, E. Erdem, F. Lin, and T. Schaub, Eds., vol. 5753. Springer, 2009, pp. 548–553.

[28] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*. CUP, 2003.

[29] M. Gelfond and V. Lifschitz, "Classical negation in logic programs and disjunctive databases," *New Generation Computing*, vol. 9, no. 3-4, pp. 365–386, 1991.

[30] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider, "Potassco: The Potsdam answer set solving collection," *AI Communications*, vol. 24, no. 2, pp. 107–124, 2011.

[31] G. Karjoth, M. Schunter, and M. Waidner, "Platform for enterprise privacy practices: privacy-enabled management of customer data," in *Proceedings of the 2nd international conference on Privacy enhancing technologies*, ser. PET'02. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 69–84. [Online]. Available: http://dl.acm.org/citation.cfm?id=1765299.1765305

[32] M. De Vos, J. A. Padget, and K. Satoh, "Legal modelling and reasoning using institutions," in *JSAI-isAI Workshops*, ser. LNCS, T. Onada, D. Bekki, and E. McCready, Eds., vol. 6797. Springer, 2010, pp. 129–140.

[33] G. D. Bibu, N. Yoshioka, and J. Padget, "System security requirements analysis with answer set programming," in *RESS*. IEEE, 2012, pp. 10–13.

[34] W. Pieters, J. Padget, F. Dechesne, V. Dignum, and H. Aldewereld, "Obligations to enforce prohibitions: On the adequacy of security policies," in *Proceedings of the 6th International Conference on Security of Information and Networks*, ser. SIN '13. New York, NY, USA: ACM, 2013, pp. 54–61. [Online]. Available: http://doi.acm.org/10.1145/2523514.2523526

[35] T. Balke, M. De Vos, and J. Padget, "Analysing energy-incentivized cooperation in next generation mobile networks using normative frameworks and an agent-based simulation," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1092–1102, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X11000574

[36] T. Balke, M. D. Vos, and J. A. Padget, "Evaluating the cost of enforcement by agent-based simulation: A wireless mobile grid example," in *PRIMA*, ser. LNCS, G. Boella, E. Elkind, B. T. R. Savarimuthu, F. Dignum, and M. K. Purvis, Eds., vol. 8291. Springer, 2013, pp. 21–36.

[37] J. Lee, T. Li, and J. Padget, "Towards polite virtual agents using social reasoning techniques," *Computer Animation and Virtual Worlds*, vol. 24, no. 3-4, pp. 335–343, 2013. [Online]. Available: http://dx.doi.org/10.1002/cav.1517

[38] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, December 2012. [Online]. Available: http://elib.dlr.de/80483/

[39] V. Baines and J. Padget, "On the benefit of collective norms for autonomous vehicles," in *Proceedings of 8th International Workshop on Agents in Traffic and Transportation*, 2014, download from http://agents.fel.cvut.cz/att2014/att2014_paper_17.pdf. Retrieved 20140704.

[40] The XMPP Standards Foundation, "Extensible messaging and presence protocol(XMPP): Core, and related other RFCs," http://xmpp.org/rfcs/rfc3920.html. [Online]. Available: http://xmpp.org/rfcs/rfc3920.html

[41] D. Alrajeh, J. Kramer, A. Russo, and S. Uchitel, "Elaborating requirements using model checking and inductive learning," *IEEE Trans. Software Eng.*, vol. 39, no. 3, pp. 361–383, 2013.