Situating COIN in the Cloud (Invited Paper)

Julian Padget

University of Bath, Dept. of Computer Science
jap@cs.bath.ac.uk

Abstract. We start from the view that the central theme of the research at the core of coordination, organization, institutions and norms, is whether the social structures and mechanisms, that have emerged over time, can be adapted and applied to artificial societies of programs and perhaps more significantly, to mixed societies of humans *and* programs –and how The means by which the social constraints that guide and regulate behaviour are acquired and represented remains an open problem. If recent experiences in information retrieval and natural language processing are plausible indicators, the statistical may yet oust the logical. Technology aside, it is clear that for socio-technical systems, that integrate human and software components, we may expect the adoption of, or the illusion of observation of, and support for human social conventions. The growing migration to cloud computing of the services that make up current pervasive, social applications suggests near-term developments emerging from the same platform(s). Thus, the question considered here is, what pathways, opportunities and challenges exist for the development, wider use and validation of COIN technologies to help realize socio-technical systems that better meet human requirements. As examples of specific enabling technologies, we review current developments in resource-oriented architecture, complex event processing and stream reasoning and observe how COIN technologies might integrate with them.

1 Introduction

At the first meeting of the COIN series in 2005, Noriega [36] spoke of "fencing the open fields" as an analogy for the need to provide suitable regulation of participant interactions in open systems governed by electronic institutions. However, the risk is that designers, in their desire to ensure the "right" outcomes, may lock actors into a protocol strait-jacket, simply because it is then possible to prove through brute-force search that compliance is assured and only good results follow. In seeking to formalize electronic institutions, this writer fell into the trap of over-specification, attempting to use the π-calculus to describe elements of the FishMarket [19]. We may observe similar trends in both policy- and law-making, where in some cases the political debate appears to exhibit a strong desire to circumscribe individual actions for the sake of the goal of achieving high levels of compliance. Paradoxically, such approaches would appear to be counter to the underlying principles, at the formal end, of game theory, or at the informal end, of economics and the notion of free markets and the 'invisible hand'. In each case, it is precisely freedom of action that imbues the arena with the flexibility to

adapt to changing circumstances and to provide the essential space – 'wiggle room', if you will – to explore as well as exploit [30]. Thus, optimization – interpreted as a high-level of compliance, regardless of cost – may be preferred to satisficing [45]. Costs here may be reflected in a lack of agility to respond to change as much as in the actual cost of policing [9].

We may also consider the freedom to move from a genuinely architectural perspective – a theme we pick up later with reference to patterns – by recalling that Alexander [3] sought what he called "quality without a name" in the design of habitable spaces. Effective, by some definition of the word, social institutions, whether they capture policies or laws, or even social conventions, can equally be viewed as habitable spaces: desirable properties they may typically exhibit are flow, ease of use and low overheads (action, cognition). There has inevitably been a reaction[1] to the up-take of design patterns and the consequences of it, because while Alexander's vision was that inhabitants would design and build their own environment, that is largely unrealistic for most users of software. The situation is potentially different however in respect of normative frameworks – if we are prepared to give participants – both human and software agents – the power to change the rules that govern them and prepared to build the tools to accommodate that change while maintaining system integrity [7, 8].

The challenge is how to regulate loosely enough to provide sufficient autonomy and flexibility, but tightly enough to meet system objectives, where most, rather than all, participants behave correctly. Indeed, such a notion of optimization may be illusory, in that all it ensures is behaviour as foreseen by the designer in advance of use. The software engineering literature and the security literature provide many anecdotes on the subject of how system integrity can be undermined by actual usage, not for any malicious motive, but simply because practice discovers more efficient – again, by some definition of the term – ways to achieve the same goals. This observation is central to the argument that underpins process mining, which seeks to discover the so-called 'desire lines' (also called more popularly elephant path, social trail, goat track or bootleg trail) that indicate the sequences of events that occur in practice. This is in contrast to the view of business process engineering, which sees such sequences as almost certainly non-optimal from a business point of view (cost, speed, reliability) and sees instead meanderings of little consequence which they call 'cow paths' [1]. Thus, whether the path is due to an elephant – and is worth preserving – or a cow – and should be eradicated – seems to depend on point of view.

The consensus in normative systems research might now appear to be in favour of regulation (good?) rather than regimentation (bad?). But even that may not be a permanently tenable position, because while the balance between the two can be established through *a priori* design choices, it may be desirable to adjust the degree of autonomy subject to prevailing conditions. These conditions may be reflected in system metrics, leading to a shift from regulation towards regimentation or vice versa, as circumstances dictate.

Let us now consider some more pragmatic issues, such as: (i) where shall the many software components execute that comprise such systems, (ii) how shall data sent between them be represented, and (iii) how shall communications between them

[1] There are many reports on, but no definitive link to, the Gang of Four trial at OOPSLA'99.

be achieved? These are not new questions for the agent community, although unfortunately our attempts to reach a conclusion on at least the latter two have been unsuccessful. These questions are not new, either, for the wider computer science community. The momentum that is forming around cloud computing not only suggests an answer to the first issue, but also that we should consider establishing common ground with the accompanying technologies of web services, publish/subscribe communications protocols and semantic annotation. Consequently, COIN technologies can be part of, rather than apart from, the cloud environment.

To return to the issue cited at the opening of fencing the open fields, the challenge left for the community then was "how to put the bell on the cat". Put another way, if we believe normative specifications and norm-aware actors are the answer, how are we to get these features into open systems, the designers of which may well see no need for such bells[2]. The purpose of this somewhat wide-ranging introduction has been to make connections with a variety of research and practice, inside and outside computer science to try to provide some form of backdrop for the current state of COIN technology research. We review the state of COIN research in the next section and put forward some of the significant tasks that we believe face us. Then in section 3, we examine the features of cloud computing that together offer an excellent experimentation and evaluation environment. Using these we can show what COIN technologies might achieve, so that we may indeed "bell the cat". We conclude with some suggestions for actions that the COIN community might take to initiate the transition to cloud computing and at the same time create greater synergy across the community.

2 The state of COIN technologies

The aim of this section is to examine from a high level what has been achieved in COIN technologies over the last decade and a half, but also to identify the various issues that stand in the way of the wider recognition of the utility it offers and, more significantly, what needs to be done to provide externally acceptable validation of the technologies. Several factors are put forward as problems relating to COIN technologies, but the overriding issues are connectivity – how to join our tools and components with the wider software world – and usability – how to join our concepts and approaches with the wider software community. The citations are intended as representative of relevant work rather than as an exhaustive survey. A complete list of the COIN volumes can be found at `http://www.pcs.usp.br/~coin/`.

Over a period of more than 15 years, COIN technologies have evolved from static normative frameworks, encoded implicitly in the control logic of software components, through explicit representations regimenting agent behaviour in trading platforms [43] or informing agent choices in agent-based simulation [34], to guiding agent and human actions in complex mixed environments [35]. But while this progression demonstrates real advances in norm representation and reasoning, those demonstrations are largely limited to small, carefully constructed illustrative cases – not necessarily helped by the publication format of the conferences and journals that the community uses, as well as

[2] or whistles.

other academic environment factors – rather than showing impact on large scale systems driven by real, rather than synthetic, data.

There is little originality in the following observations on the criticisms that can be levelled at COIN technologies and could equally be applied to other areas of computer science, or even science in general:

1. **Plausibility:** this is the first barrier to up-take. We have no demonstrators that make the case in practice for COIN technologies; there is only the potential and the case is not compelling because existing systems work – or appear to – on small and carefully selected illustrative scenarios.
2. **Scalability:** part of the lack of plausibility stems from a lack of evidence either in the form of deployed systems or from theoretical analysis of the capacity for the technologies to scale; we need to demonstrate solutions at scale. But this is a chicken-and-egg situation, because we also need host systems that can be augmented post-facto by COIN technologies in order to be able to make that case.
3. **Visibility:** the benefits of the use of COIN technologies need to be clear and to offer substantive improvement, possibly in several ways, over conventional technologies. But, frustratingly, the best indicator of effective application of COIN technologies may be that they are barely noticeable.
4. **Packaging:** a practical barrier to up-take depends upon how COIN technologies are delivered. New technologies that either require re-training, new interfaces or discarding (part of) the existing software base inevitably face greater resistance to up-take than something that integrates by means of widely-used interfaces and which enhance rather than replace. Not least, the capacity for turning enhancements on and off may provide a valuable way to demonstrate their impact. We need to deliver COIN technologies in packaging that not only helps us as a community to integrate and evaluate what we do, but also to integrate with minimal overhead or impact with legacy systems.

COIN technologies have the potential to contribute to the creation of systems that are all of open, distributed, intelligent and adaptable – even if those terms might require an essay each of their own to circumscribe expectations and establish the connections between them. But the software we have built so far typically has limited *interoperability*: a Java library can be a useful component and might with some effort be deployed as a web service, but lack of systems experience means it is hard to say whether an API is a suitable interface or whether a richer communication language is desirable. A further artefact of the development process is the difficulty of *re-usability*: although the technologies aim to be and often are quite general purpose in nature, the supporting software can be quite sensitive to deployment environment and hard to maintain even in the short to medium term. *Performance* is also often over-looked, not least because the development scenarios that illustrate the properties we wish to demonstrate (for academic purposes) are quite small, but also because it is hard to identify useful metrics and because the decision procedures in play do not, or cannot, have well-defined performance profiles. Much COIN technology software is written to demonstrate that a particular behaviour or envelope of behaviours can be realized, but there is little culture as yet in the practice of *patterns* for COIN technologies: this may percolate through from

the underlying software base, but that seems more likely to address how some function is realized, rather than the function itself. Finally in this tally of criticisms, there is the matter of *resilience*: given that flexibility and adaptation in the face of changing circumstances are among the benefits that should follow from COIN technologies, it seems essential that it should be possible to demonstrate such properties in our own software – that is, reflection – and not just in the systems to which it is applied.

2.1 The Agent View

Previously in intelligent agent research, agent architecture was a major topic from the earliest days of agent-based simulation (ROSS, SWIRL and TWIRL [32]) and subsequently with the more complex layered architectures such as Touring Machines [20] and InteRRap [33]. However, these are now largely forgotten and either regimented agents, such as in e-Institutions [43] and $\mathcal{M}$OISE$^+$ [26], or BDI and variants appear to be the common choices. In consequence, a number of those variants of BDI have sought to address the matter of how an agent could and should avail itself of normative reasoning. This has lead to a dichotomy between the internal approach, which further divides between full incorporation [4] and separation of BDI and normative knowledge [14] and the external approach, where normative positions are communicated to agents in the form of obligations [2], determined by normative reasoning components [13].

The convergence of architectural choice on BDI makes system comparison somewhat simpler, but has also had the effect of pushing processing into the agent that is not necessarily so easily handled at that level. Specifically, complex trigger formulae for BDI actions are both hard to test, inasmuch as BDI testing is feasible [48], and to maintain. Furthermore, this hard-codes plan triggers into agents, reducing scope for resilience, since they cannot easily be adjusted in response to changing circumstances.

2.2 The Organization View

In parallel with the evolution of agent architecture, the twin notions of institution and organization have also undergone significant development. In the first instance, there was the FishMarket [44], in which the agents cede control to a governor that directs which actions they shall take in order that each agent be fully compliant with the rules of the institution. Although the approach is somewhat different, $\mathcal{M}$OISE$^+$ achieves similar goals, in which agents are constrained by the role they play within a group, so that agents are in effect regimented. A second group of approaches to institutional specification have favoured agent autonomy over guaranteed compliance, using a variety of formalisms [22, 12, 13, 46], in which the common trait is an external entity that reasons about agent actions in respect of the governing norms and identifies normative positions and obligations acquired by agents in consequence.

Dignum and Padget [15] put forward a view that brings together organization and institution. Here, the latter capture the regulations pertaining to specific contexts, and the former expresses the combination of the many institutions that together describe the processes that make up an organization together with the roles of the actors that participate in those institutions.

3 COIN in the Cloud

The preceding sections have summarised a view on the current state of COIN technologies and also put forward some shortcomings that we believe must be addressed in order to raise awareness of the technologies outside the immediate agent community. The purpose of this section is to assess how COIN and cloud technologies might fit together and what actions we might take as a community to bring that about.

The most significant and attractive feature of cloud computing is that it offers provision on demand. Furthermore, that provision can be configured through virtual machines to exactly the combination of operating system and resources that a particular program requires, facilitating the deployment of legacy codes so they can be accessible from anywhere. Cloud computing is also unavoidably distributed, which makes it essential that we establish some conventions on how to interact with COIN deployments – some approaches are discussed below. Distribution offers the opportunity not only for the community to share software (as a service), but also to take advantage of the mechanisms such as enterprise bus architectures, distributed messaging systems and different forms of web services to connect our components with one another and with a huge range of other services. This could potentially significantly reduce our development burden by re-using rather than re-inventing.

Cloud computing can also be seen as the product of an evolutionary process in computing systems. This began with closed systems in which all the components are the product of a single team running on one computer, moved through the (re-)use of libraries and components connected by CORBA running on several computers (on an intranet) and now seeks the creation of increasingly open systems. In this last, components may be replaced/upgraded in live deployments over a range of platforms and edge devices across the internet.

Clearly, from a systems management perspective, the complexity increases significantly in the transition from one computer, to many on a LAN, to many – where devices join and leave the system over time – on a WAN/internet. The governance of such systems has to be distributed, with many components making decisions on the basis of local circumstances, but also, crucially, informed by guidelines for the perceived correct running of the system, which is where COIN technologies are key. Recognition that both much data is being created all the time and that its rapid interpretation is necessary (until we establish what is actually worth collecting) forms the motivation for the SHINE project[3], which brings together sensors, social media and intelligent agents to answer complex information retrieval requests in real-time. But not only are such systems conduits for data and information for delivery to human users, they are also huge data sources in their own right, as each component can be viewed as a monitoring element that feeds into an over-arching process that can observe system health, identify the early stages of anomalous or undesirable situations and alert system control agents of the need to consider whether action need be taken and what it might be [16, 17].

Another inevitability concerns the relationship between the producer and consumer of such data. Up to a certain level, a request-response communication model, as has conventionally been used in agent systems. This follows the tradition of procedural

[3] `http://direct.tudelft.nl/shine-117.html`

programming, through remote procedure call to SOAP-based web services – and can work, as long as network latency is low enough. However, asynchronous communications in the form of event notifications, publish/subscribe and RESTful web services are seeing increasing adoption as mechanisms that, while imposing constraints on the programming model, manage to insulate components from the innate and unpredictable latencies in operating across wide area networks. This perspective leads to a view of software components – of all kinds – as a rich variety of event processors – both as consumers and producers of events – connected together in loose and dynamic opportunistic networks.

Unfortunately, although by design, producers know nothing of the requirements or capabilities of consumers and the data volumes can be hard to process and assimilate, if they are not at the information level and frequency commensurate with the consumer's reasoning cycle. For example, a BDI reasoning cycle in one Jason application (driving autonomous vehicles in simulation) [31] appears to be able to cope with percept updates every few hundreds of milliseconds, although this frequency will depend upon number of plans, the complexity of their triggers and the duration of their actions. The critical issue here is the rate at which a consumer can process the event streams it is receiving in order for it to be able to achieve its intended level of situational awareness. This in turn can be exacerbated by the level of the event information being incompatible with the plan triggers (for example) – typically by being at a lower level – so that patterns comprising several events, and possibly several event sources must be recognized [41] before a plan can be triggered. Such processing is typically difficult to develop, debug and maintain *within* an agent framework and would be better out-sourced to a component that can deliver a single percept covering a set of events that characterize a given situation. This leads to the creation of more independent software components in an increasingly complex communication network and a consequent need for further system health monitoring and appropriate presentation of that data, if there is to be any chance of identifying anomalous behaviours.

Looking back at the above, there appear to be several aspects of cloud computing that have features that we need both to demonstrate the benefits of COIN technologies, and also to provide the facilities we need to demonstrate those benefits, amongst which:

1. Sources of data both about the system and the application domain
2. The capacity to create (on demand) the computational facilities to interpret it
3. Means to deploy legacy code in bespoke environments through cloud-based VMs
4. On demand creation of new services to filter, aggregate and summarize data comprising multiple events, even from multiple sources.

3.1 Understanding the Situation

If all the above can be realized, it puts us on a path towards the creation of system that can construct degrees of situational awareness, both about the state of their own system (self-awareness) and awareness of situations as they evolve in the application domain of the system. The outcome should be the means to monitor and to take action on both system health and actor health. Eventually, it should even become feasible to discern the desire lines [1] and, following some collective decision procedure, implement a consistent revision of the governing norms [8].

We want to enable software agents to make good and timely action choices in a variety of situations, thinking specifically about those created by: (i) virtual environments, where there may be a mix of software- and human-controlled avatars and (ii) (imagined) socio-technical systems, such as in search and rescue, military, emergency-response and medical domains.

Intelligent behaviour, or behaviour that is perceived as intelligent, may be attributable to many factors. The one of concern here is how an agent uses information about its situation to make choices of actions. The right choice may be regarded as intelligent. The wrong choice and at worst the agent or its collaboration partners, or even the humans for which it is working, may be exposed to some risk. The situation may not be as black-and-white as just described, but rather there may be better choices, amongst which a dominating choice may not be readily apparent, and worse choices. Optimisation is not required, but satisficing is. Seemingly less serious, at first sight, is that by making a worse choice, trust and plausibility is forfeited and the agent is seen as just another program. However, this too can be critical, not because some pretence need be maintained, but because the loss of reputation and believability induces distraction in a human participant, since they are now looking for the next "mistake", as well as everything else they are doing. As a result, the whole collaborative activity may no longer achieve its goals.

Although there are many factors affecting the effectiveness of such mixed system, we highlight two that we believe have a significant influence on everything else: speed of response and breadth of knowledge. A third aspect is a corollary to these two: the matter the communication of data between components.

Speed of response Early agent architectures, such as those mentioned earlier [20, 33] sought to reflect then current AI thinking, influenced by research in robotics, by proposing a layered architecture comprising reactive, deliberative and generative components, reflected in the apotheosis of this line of development by Soar [29]. Meanwhile, as noted earlier, the agent community has largely converged on the BDI architecture to fulfil the function of the deliberative and generative layers. In consequence, the reactive layer is effectively subsumed into BDI as events lead to the addition of percepts that in turn cause actions. This usefully simplifies the agent structure and programming because control is all expressed through some variant of the AgentSpeak language. However, the BDI architecture is not designed for the rapid assimilation and assessment of high frequency data and even if higher performance implementations were available, the relative sophistication of the architecture is at odds with the task asked of it. Since the purpose of BDI is to support the deliberative component of an agent, this suggests that: (i) high frequency data needs to be processed and somehow summarized to provide lower frequency data with an implied higher informational value, and (ii) such processing must use a lower overhead computational model to accommodate the high frequency data rate. Out-sourcing this task to a process situated between the data source and the agent, therefore seems sensible. Connectivity can be addressed through some of the distributed computing technologies identified earlier.

Breadth of Knowledge Understanding the situation should allow us to choose an appropriate action both for the situation and with respect to individual and group goals. Programming an agent to be prepared for any and every situation is clearly infeasible. Thus, while decision-making must remain the responsibility of the agent, the information upon which that decision is the result of the assessment of sensed data from a variety of sources across a range of time frames and representations. Again, it is likely to be infeasible and undesirable to integrate all such assessment processes within an agent. Thus, although the BDI agent has the capacity for deliberation, not all deliberations are within its capabilities and in common with both software engineering and societal structures, such deliberation could be delegated, leading to recommendations from which to choose. From this, again two conclusions follow: (i) specialized domains may be better reasoned about externally, producing summary recommendations for the agent to choose between and (ii) such processing can use representations and resources appropriate to the domain, or indeed re-use existing reasoning systems. Once again, the conclusion from this is the desirability of out-sourcing the task to a process that may function more like a service, collecting inputs from several sources and publishing results either when ready or on request. A particular case in point are the institutions that comprise an organization, for example, where it is the institutions that act as repositories of social state(context) and provide the function of social reasoning to identify violations and obligations. It nonetheless remains the responsibility of agents to decide what actions to take, in the light of information received from the institutions.

Making Connections There are two issues to address under the heading of connectivity: (i) how to pass data between the components – representation and protocol – and (ii) how to package existing software to operate in such an environment, and each will affect the other. Our aim is for a low overhead, low maintenance connection fabric, preferably that can be relied upon for support in the wider internet community for some time. Network speeds continue to increase, but latency, as a relative factor, does not change significantly as a proportion of the delay. Thus, although it is traditional to speak of synchronous and asynchronous systems, the practice reflecting the physical constraints of the internet, is either for loosely synchronizing systems, where one side may pause or continue working while waiting for a response from the other side, or asynchronous systems in which components push out data without concern for the receiver, while other components pull in data as it suits them.

Much effort has been expended in the agent community on trying to reach agreement on what and how to communicate between agents. The results have been inconclusive and furthermore are unlikely to see up-take, or indeed support, outside the agents community. Therefore, we suggest that the pragmatic solution is to adopt widely used, maintained and developed standards, so that the task for the community is just to track those standards and build components that utilise them. Specifically, for protocols, this suggests something based on HTTP, to ensure traffic across firewalls, such as the eXtended Messaging and Presence Protocol (XMPP) and for content, something based on RDF, but possibly defined in terms of OWL in order to put constraints on the RDF.

Having discussed broadly the form of some requirements, the following two sections present brief introductions and some indicative references to technologies that

may offer some solutions. These are resource-oriented architecture and event processing. The rationale for highlighting these two is firstly, their fit with the intrinsic computational properties of the internet as a distributed computing environment and secondly, as emerging maturing technologies and frameworks, into which we can embed and package COIN technologies, in order to deliver reach beyond individual research groups and beyond the COIN community.

4 Resource Oriented Architecture

It is perhaps an oversimplification to say that ROA is for RESTful web services [47] what Service Oriented Architecture (SOA) is for RPC-style (SOAP) web services, but it does capture the sense of the relationship, in that REST should be seen as one way of achieving resource orientation [38]. In practice, ROA, as do events, enables decoupling of components based on stateless message exchange. Stateless however does not mean state cannot be modelled, but rather that each state, if so required, is a new resource, identified by a new, unique URI. Furthermore, message exchange does not mean request-response in the RPC sense, rather an operation (the request) typically results in a new resource, identified by URI (the response). From this perspective, it has much in common with functional programming.

Although Resource Oriented Architecture denotes a general purpose set of principles in respect of web application design, it is fair to say that its synthesis has been driven by the concrete aspects of RESTful web services and the principles of addressability, statelessness, connectedness, and a uniform interface [42]. Consequently,

1. Resources can be universally identified by unique addresses (addressability),
2. Every request to a resource should contain all the information needed for further processing (statelessness),
3. A resource representation should contain the addresses of all related resources (connectedness) and
4. All resources can be manipulated through uniform methods (uniform interface).

REST-compliant Web services differ from RPC-style Web services in the protocol employed between client and server, in that the latter requires each application designer to define a new and arbitrary protocol comprising vocabulary of nouns and verbs that is usually overlaid on the HTTP protocol[39]. In contrast, REST services work directly with the HTTP verbs of POST, PUT, GET and DELETE which map to the four basic functions of data storage, namely Create, Update, Retrieve and Delete.

As with adopting a purely event-oriented approach, ROA/REST puts constraints on design (and implementation), which can initially be tiresome. But, as with functional, or perhaps a better analogy would be with dataflow/single assignment languages, there are potentially significant benefits, when operating in a distributed environment and especially one with unpredictable levels of latency. Specifically cited benefits [21] include simpler protocols, better synergy with underlying web components, and lower application design costs, but it is hard to find studies that substantiate these largely qualitative claims.

5 Event Processors

This section discusses how to carry out data analysis on behalf of agents, based on the twin criteria of speed of response and breadth of knowledge, identified above. We review some of the work in this area over the past few decades and evaluate its suitability for incorporation into the connection fabric outlined earlier.

5.1 Real-time expert systems

The 1970s and 80s saw the development and refinement of expert systems and the shells used to author them. Two directions emerged from this activity that are still in use today: (i) rule-based systems utilising the RETE algorithm, and (ii) the Prolog language, which although not solely used for building expert systems, nevertheless provides a conceptually similar environment, where facts drive inferencing, expressed through rules. A handful of tools now represent the first group, with some commercial (Ilog Rules) and a few free/open-source (JBoss Rules, JESS) examples. Most recent publications are domain-oriented, focussing on how an expert system has been applied in a particular control context. This underlines the maturity and stability of the underlying technology, which is either RETE (the net algorithm) or Selective Linear Definite clause resolution (logic languages).

A naive implementation of a rule selection process is linear in the number of rules. Any given rule set is finite and so has an upper bound on the matching time required, but is not an adequate guarantee of real-time response, because it means that the number of rules and the complexity of the rule conditions that determine which are applicable must be constrained in order to meet performance requirements. RETE effectively compiles the left hand sides of the rules to build a decision network that identifies the conflict set (of rules that match the current state). While, this and subsequent improvements, are better than the naive approach, complexity is still $O(n)$. The story is broadly the same in logic languages, but with semantic changes (committed choice) and the use of parallel resolution algorithms helping to improve performance.

The key requirement is to be able to know how long the match process will take. This in turn can be affected by placing limitations on the left-hand side conditions resulting in the network having particular time-based properties. The need for real-time response was recognized decades ago and some of the issues surrounding the delivery of such performance are addressed in [28], while a notable contemporary example of the application of the technology was IBMs YES/MVS system [18]. A current example is GENSYMs G2[4], which appears to have significant up-take in process control settings.

RTES provide general programmability through a declarative framework as well as the means to accumulate data over arbitrary long time windows. The programmability also affects how real-time a particular system can be in practice. Prediction of match time can be computed off-line (static analysis of the rules). However, apart from providing an upper bound on the cycle time, there appears to be no way either to guide or to constrain the programmer to produce a match procedure with guaranteed performance. It may be possible to apply syntactic restrictions to the rules so that the (RETE)

[4] Retrieved from `http://www.gensym.com/`, 20130111

networks generated are of limited propagation depth. But either way – syntactic restriction or cycle time bound – is not a solution if it means expressing the reasoning logic becomes either contorted, creating a maintenance problem, or it is just impossible to express what is wanted. Furthermore, the match procedure is very fragile in respect of the system as a whole, because it can be pushed outside its performance envelope either by accommodating new requirements or by an increase in the data rate of one of its subscription feeds.

5.2 Complex event processing (CEP)

Like many concepts in Computer Science, event processing is not new and could, terminologically, be traced back to the earliest operating systems and soon after with the development of discrete event simulation frameworks such as GPSS [24] and Simula [37]. But there, events were the drivers (of simulation), rather than the subject of analysis themselves. The emergence of the topic of verification as a topic within the design process of various kinds of systems, at both hardware and software levels, focussed on the scrutiny of event traces to detect desirable or undesirable patterns of behaviour. Model checking languages and the tools associated with the various calculi of concurrent systems are some of the computational approaches that have resulted from the objective of understanding large collections of traces in their entirety. Such systems analyse all possible systems states – and the paths between them – in pursuit of the establishment of system invariants. In contrast, ad-hoc solutions to analysing such things as packets on networks, transactions in distributed systems and intrusion detection systems laid the foundations for looking at fragments of traces for significant or anomalous activity. Financial markets were early adopters of the conceptual model, using it to process real-time market feeds for events of significance.

The languages for expressing CEP have become more sophisticated and now offer functions to filter, correlate and aggregate data. The practical question remains however, of how quickly can or must such operations be carried out. Esper [5] and Drools Fusion[6] are typical examples of event processing engines – which notably have a strong commercial orientation, coupled with up-take in the financial sector. The conceptual model is that the user registers queries with the event processor, which will then invoke the query when its conditions match. In Esper, the matching conditions can express durations, the composition of several different streams, filtering, aggregation and sorting. An important abstraction feature is the means to glue together statements using "followed by" conditions. The summary of features that follows is abstracted from the Esper tutorial[7]. Espers programming language shares some syntactic features with Structured Query Language (SQL) for accessing relational databases, in particular in its `select` and `where` clauses, but the operations are carried out on views – finite length fragments – over streams rather than tables. Views can represent windows over a stream

[5] Retrieved from `http://esper.codehaus.org`, 20130111.

[6] Retreieved from `http://www.jboss.org/drools/drools-fusion.html`, 20130111

[7] Retrieved `http://esper.codehaus.org/tutorials/tutorial/tutorial.html` 20130110

of events, specified by time or count. Views can also sort events, derive statistics from event properties, group events or handle unique event property values. Furthermore, a window can be used to preserve data, in effect by defining an internal table that can be used for input or output by other queries. It is not clear how performance is guaranteed, but the finiteness of the windows and the relative simplicity of the queries that can be performed would certainly limit demand for processing time.

5.3 Stream Reasoning

The term 'stream reasoning' has emerged in the last few years. Initially, this has applied to the means to process streams of RDF triples, using SPARQL extended with concepts similar to those discussed above [10].From a technical point of view, there would appear to be very little difference between the two, since both rely upon finite state machines to recognise patterns in the input stream in order to trigger some action. The important advance offered by C-SPARQL, however, is that the event data may also contain reference to ontologies, and not just literals. As a result, data from different sources may be associated semantically, rather than through the syntactic structure of the stream records, thus reducing the coupling (in the software engineering sense) between producer and consumer.

Alternative, but different, logic-based approaches are put forward by Gebser et al [23] and Anicic-et-al [5]. The difference is that the former is based on answer set semantics, while that latter uses Prolog, but both have to face technical challenges to ensure that the tree synthesis approach both works efficiently and does not consume unbounded resources. The essential idea is that the head of a logic programming term denotes the recognition of a complex event, subject to the satisfaction of the right hand side of the clause. As such, the programmer is presented with a language that has much in common with real-time expert systems, but the implementations must provide temporal performance guarantees as well as a means to 'forget' facts in order to recover memory. As a result, both approaches incorporate a substantial amount of theoretical work that establishes the correctness of their mechanisms to recover memory and to compute results under real-time constraints.

6 Closing remarks

We have put forward a personal survey of the situation in COIN and how its strengths and weaknesses fit with cloud computing. In particular, we believe that the resource-oriented architecture and stream processing have much to offer in conjunction with COIN technologies. In conclusion, we make some suggestions for next steps:

1. Make our components deployable as Software as a Service (SaaS), possibly through RESTful [39] interfaces and informed by a ROA perspective,
2. Out-source complex situation analysis to software built for the purposezip, rather than trying to embed it in an agent architecture that was not designed for the task,
3. Collect datasets from cloud systems for testing and training of decision-making components, but not forgetting the value of synthetic datasets as a means to test boundary conditions,

4. Utilise communication protocols and data representations that enable interoperability, such as enterprise bus architectures [6, 27, 40] or distributed communication architectures [11] with semantically-annotated messaging for further decoupling of producer and consumer, and
5. Take advantage of free cloud services to prototype and share ideas and services.

Acknowledgements

I am grateful to the organizers of COIN 2012 at AAMAS for inviting me to speak at the meeting and to have the opportunity to follow that up with this invited paper. Some of ideas presented here have been developed in part with the support of The Royal Society (UK), the University of Otago and the COST action on Agreement Technologies.

References

1. van der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011), ISBN: 978-3-642-19344-6
2. Alechina, N., Dastani, M., Logan, B.: Programming norm-aware agents. In: van der Hoek et al. [25], pp. 1057–1064
3. Alexander, C.: A Timeless Way of Building. Center for Environmental Structure, Oxford University Press Inc, USA (1980), ISBN-13: 978-0195024029
4. Andrighetto, G., Villatoro, D., Conte, R.: Norm internalization in artificial societies. AI Communications 23(4), 325–339 (2010)
5. Anicic, D., Fodor, P., Rudolph, S., Stühmer, R., Stojanovic, N., Studer, R.: A rule-based language for complex event processing and reasoning. In: Hitzler, P., Lukasiewicz, T. (eds.) RR. Lecture Notes in Computer Science, vol. 6333, pp. 42–57. Springer (2010)
6. Apache Camel. Website, retrieved from `http://camel.apache.org/`, 20130104
7. Artikis, A.: Dynamic protocols for open agent systems. In: Sierra, C., Castelfranchi, C., Decker, K.S., Sichman, J.S. (eds.) AAMAS (1). pp. 97–104. IFAAMAS (2009)
8. Athakravi, D., Corapi, D., Russo, A., Vos, M.D., Padget, J.A., Satoh, K.: Handling change in normative specifications. In: van der Hoek et al. [25], pp. 1369–1370
9. Balke, T., Vos, M.D., Padget, J.: Normative run-time reasoning for institutionally-situated BDI agents. In: Coordination, Organizations, Institutions, and Norms in Agent Systems VI - COIN 2012 International Workshop, COIN@AAMAS 2012, Valencia, Spain, June 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. This volume, p. tbd. Springer (2013)
10. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: Querying RDF streams with C-SPARQL. SIGMOD Record 39(1), 20–26 (2010)
11. Bernstein, D., Vij, D.: Using XMPP as a transport in intercloud protocols. In: CloudComp, 2010 the 2nd International Conference on Cloud Computing (2010)
12. Cardoso, H.L., Oliveira, E.C.: Institutional reality and norms: Specifying and monitoring agent organizations. Int. J. Cooperative Inf. Syst. 16(1), 67–95 (2007)
13. Cliffe, O., De Vos, M., Padget, J.: Modelling normative frameworks using answer set programing. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR. Lecture Notes in Computer Science, vol. 5753, pp. 548–553. Springer (2009)
14. Criado, N., Argente, E., Botti, V.: A BDI architecture for normative decision making. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1. pp. 1383–1384. International Foundation for Autonomous Agents and Multiagent Systems (2010)

15. Dignum, V., Padget, J.: Multiagent organizations. In: Weiss, G. (ed.) Multiagent Systems. MIT Press, 2^{nd} edn. (2012), in press
16. El-Akehal, E.E.D., Padget, J.A.: Pan-supplier stock control in a virtual warehouse. In: Berger, M., Burg, B., Nishiyama, S. (eds.) AAMAS (Industry Track). pp. 11–18. IFAAMAS (2008)
17. Elakehal, E.E., Padget, J.: Market intelligence and price adaptation. In: Proceedings of the 14th Annual International Conference on Electronic Commerce. pp. 9–16. ICEC '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2346536.2346538
18. Ennis, R.L., Griesmer, J.H., Hong, S.J., Karnaugh, M., Kastner, J.K., Klein, D.A., Milliken, K.R., Schor, M.I., Van Woerkom, H.M.: A continuous real-time expert system for computer operations. IBM J. Res. Dev. 30(1), 14–28 (Jan 1986), http://dx.doi.org/10.1147/rd.301.0014
19. Esteva, M., Padget, J.: Auctions without auctioneers: distributed auction protocols. In: Moukas, A., Sierra, C., Ygge, F. (eds.) Agent-mediated Electronic Commerce II. Lecture Notes in Artificial Intelligence, vol. 1788, pp. 20–38. Springer Verlag (2000), available via http://dx.doi.org/10.1007/10720026_12
20. Ferguson, I.A.: Touring machines: Autonomous agents with attitudes. Computer 25(5), 51–55 (May 1992), http://dx.doi.org/10.1109/2.144395
21. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine, Irvine, California (2000)
22. Fornara, N., Viganò, F., Verdicchio, M., Colombetti, M.: Artificial institutions: a model of institutional reality for open multiagent systems. Artif. Intell. Law 16(1), 89–105 (2008)
23. Gebser, M., Grote, T., Kaminski, R., Obermeier, P., Sabuncu, O., Schaub, T.: Stream reasoning with answer set programming: Preliminary report. In: Brewka, G., Eiter, T., McIlraith, S.A. (eds.) KR. AAAI Press (2012)
24. Gordon, G.: The development of the general purpose simulation system (gpss). In: Wexelblat, R.L. (ed.) History of programming languages I, pp. 403–426. ACM, New York, NY, USA (1981), http://doi.acm.org/10.1145/800025.1198386
25. van der Hoek, W., Padgham, L., Conitzer, V., Winikoff, M. (eds.): International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes). IFAAMAS (2012)
26. Hübner, J.F., Sichman, J.S., Boissier, O.: A model for the structural, functional, and deontic specification of organizations in multiagent systems. In: Bittencourt, G., Ramalho, G. (eds.) Advances in Artificial Intelligence, Lecture Notes in Computer Science, vol. 2507, pp. 118–128. Springer Berlin Heidelberg (2002), http://dx.doi.org/10.1007/3-540-36127-8_12
27. Ibsen, C., Anstey, J.: Camel in Action. Manning (2010), iSBN-13: 978-1935182368
28. Laffey, T.J., Cox, P.A., Schmidt, J.L., Kao, S.M., Read, J.Y.: Real-time knowledge-based systems. AI Mag. 9(1), 27–45 (Mar 1988), http://dl.acm.org/citation.cfm?id=44132.44133
29. Laird, J.E., Newell, A., Rosenbloom, P.S.: Soar: An architecture for general intelligence. Artificial Intelligence 33(1), 1 – 64 (1987), http://www.sciencedirect.com/science/article/pii/0004370287900506
30. Lazer, D., Friedman, A.: The dark side of the small world: how efficient information diffusion drives out diversity and lowers collective problem solving ability. Program on Networked Governance (PNG) Working paper 06-001, Harvard University (2006), retrieved from http://www.hks.harvard.edu/netgov/files/png_workingpaper_series/PNG06-001_WorkingPaper_LazerFriedman.pdf, 20130106
31. Lee, J., Baines, V., Padget, J.: Decoupling cognitive agents and virtual environments. In: Dignum, F., Beer, M., Brom, C., Hindriks, K., Richards, D. (eds.) First International Work-

shop on Cognitive Agents for Virtual Environments. LNAI, vol. 7764, pp. 17–36. Springer (2013)
32. McFall, M.E., Klahr, P.: Simulation with rules and objects. In: Proceedings of the 18th conference on Winter simulation. pp. 470–473. WSC '86, ACM, New York, NY, USA (1986), http://doi.acm.org/10.1145/318242.318479
33. Müller, J.: The Design of Intelligent Agents: A Layered Approach, LNCS, vol. 1177. Springer (1996), DOI 10.1007/BFb0017806
34. Neville, B., Pitt, J.: Presage: A programming environment for the simulation of agent societies. In: Hindriks, K.V., Pokahr, A., Sardiña, S. (eds.) ProMAS. Lecture Notes in Computer Science, vol. 5442, pp. 88–103. Springer (2008)
35. Nieves, J.C., Padget, J., Vasconcelos, W., Staikopoulos, A., Cliffe, O., Dignum, F., Vázquez-Salceda, J., Clarke, S., Reed, C.: Coordination, organisation and model driven approaches for dynamic, flexible, robust software and services engineering. In: Schahram, D., Li, F. (eds.) Service Engineering, pp. 85–115. Springer (2011), http://dx.doi.org/10.1007/978-3-7091-0415-6_4, ISBN: 978-3-7091-0414-9
36. Noriega, P.: Fencing the open fields: Empirical concerns on electronic institutions (invited paper). In: Boissier, O., Padget, J.A., Dignum, V., Lindemann, G., Matson, E.T., Ossowski, S., Sichman, J.S., Vázquez-Salceda, J. (eds.) AAMAS Workshops. Lecture Notes in Computer Science, vol. 3913, pp. 81–98. Springer (2005)
37. Nygaard, K., Dahl, O.J.: The development of the simula languages. In: Wexelblat, R.L. (ed.) History of programming languages I, pp. 439–480. ACM, New York, NY, USA (1981), http://doi.acm.org/10.1145/800025.1198392
38. Overdick, H.: The Resource-Oriented Architecture. 2007 IEEE Congress on Services Services 2007 0, 340–347 (2007), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4278816
39. Pautasso, C., Zimmermann, O., Leymann, F.: RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision. In: 17th International World Wide Web Conference (WWW2008). pp. 805–814. Beijing, China (April 2008 2008), http://www2008.org/
40. Ranathunga, S., Cranefield, S.: Embedding BDI agents in business applications using enterprise integration patterns (extended abstract). In: Ito, Jonker, Gini, Shehory (eds.) Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013). IFAAMAS (2013), to appear.
41. Ranathunga, S., Cranefield, S., Purvis, M.K.: Identifying events taking place in second life virtual environments. Applied Artificial Intelligence 26(1-2), 137–181 (2012)
42. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly Media, Inc., 1 edn. (May 2007)
43. Rodriguez, J.: On the design and construction of agent-mediated electronic institutions. IIIA Monographs 14 (2001)
44. Rodríguez, J.A., Noriega, P., Sierra, C., Padget, J.: FM96.5 A Java-based Electronic Auction House. In: Proceedings of 2nd Conference on Practical Applications of Intelligent Agents and MultiAgent Technology (PAAM'97). pp. 207–224. London, UK (Apr 1997), http://www.iiia.csic.es/Projects/fishmarket/PAAM97.ps.gz, iSBN 0-9525554-6-8
45. Simon, H.A.: Rational choice and the structure of the environment. Psychological Review 63(2), 129–138 (1956)
46. Vázquez-Salceda, J., Dignum, V., Dignum, F.: Organizing multiagent systems. Autonomous Agents and Multi-Agent Systems 11(3), 307–360 (2005)
47. Web Services Architecture. http://www.w3.org/TR/ws-arch/\#relwwwrest, http://www.w3.org/TR/ws-arch/\#relwwwrest, retrieved 20110808
48. Winikoff, M., Cranefield, S.: On the testability of BDI agent systems. Information Science Discussion Papers Series 2008/03, University of Otago (2008), retrieved from http://hdl.handle.net/10523/1063, 20130104