An Open System for Social Computation

David Robertson [a,1], Luc Moreau [b], Dave Murray-Rust [a] and Kieron O'Hara [b]
[a] *Informatics, University of Edinburgh*
[b] *Electronics and Computer Science, University of Southampton*

Abstract. Part of the power of social computation comes from using the collective intelligence of humans to tame the aggregate uncertainty of (otherwise) low veracity data obtained from human and automated sources. We have witnessed a surge in development of social computing systems but, ironically, there have been few attempts to generalise across this activity so that creation of the underlying mechanisms themselves can be made more social. We describe a method for achieving this by standardising patterns of social computation via lightweight formal specifications (we call these social artifacts) that can be connected to existing internet architectures via a single model of computation. Upon this framework we build a mechanism for extracting provenance meta-data across social computations.

Keywords. social computation

1. Introduction

We have increased enormously our ability to acquire, store and process data but this comes at the expense of data veracity; we have sophisticated data but often it is subjective, contextualised and prone to misinterpretation. The proportion of this sort of low veracity data increases as sensors, devices and social networks proliferate. We therefore find ourselves in a situation where, in many domains, supply of data far exceeds our ability to perform useful automated tasks with it.

Consider, for example, healthcare where we are rich in data at a variety of levels, from intimately personal data obtained from personal health monitoring devices through to data reflecting population trends and demographics. All of this, potentially, could inter-relate but the technical task of relating such diverse data seems daunting and there is concern over the ethics and motivations of monolithic organisations who operate as integrators. A key issue is whether deeper human engagement with socially derived data can assist technical operations and also help to address issues of computational and data monopoly as the technology grows.

Algorithms that infer useful information from data continue to improve but many tasks and judgements require human effort. For some of these, such as reliably identifying a complex feature in a noisy image, human effort is required because algorithms have not caught up with human skills. For others, such as crowdsourcing views or support, the overall task requires choices that are deeply rooted in subjective human experience. Connectivity over computer networks, complemented by ubiquitous social networking,

[1]Corresponding Author: Informatics, University of Edinburgh; E-mail: dr@inf.ed.ac.uk.

now provides opportunities to tap into human skills and experience at large scale. Innovative applications have followed, each with its own special mixture of accessibility and incentive to participate: some (*e.g.* ReCaptcha[16]) gain human effort by interweaving the effort seamlessly with a task that humans already perform in large numbers; some (*e.g.* Foldit[5]) motivate participants by turning the social task into a game; some (*e.g.* Galaxy Zoo[3]) attract people to the social benefit of the exercise; in others (*e.g.* DARPA Challenge systems[13]) the incentive may be personal payment. All of these systems are effective but each is a unique software system, built to tackle a specific type of problem. Consequently, conventional social computing systems (although solving individual problems that resist conventional methods) are never combined to tackle computing problems of great sophistication. Conversely, conventional computing applications have grown to be more sophisticated (having been developed through the integration of many components via structured design methods) but are unsophisticated in their use of human social structures. This leaves the unexplored area of social computation that we can't yet do, illustrated in Figure 1.

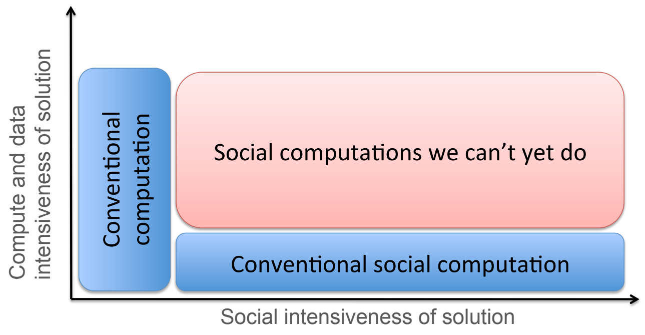
Compute and data intensiveness of solution
Conventional computation
Social computations we can't yet do
Conventional social computation
Social intensiveness of solution

Figure 1. The space of potential social computations

We are, fortunately, unable to inhabit the unexplored space of Figure 1 by fiat - we cannot and should not impose computation on human societies. We do, however, want to provide mechanisms through which more complex social computations may socially be assembled. This paper describes one way of achieving this goal, based on existing methods and computational infrastructure. At the core of our method is the concept of a social artifact which describes (in a formal language independent of the computational environment) a social interaction. We describe this lingua franca for social computation in Section 2. In the remainder of this section we give a simple social computation scenario (to which we return later in the paper) and provide context to our approach in terms of conventional social computation-related languages and architectures.

1.1. An Example Problem

Concepts such as friendship networks or crowdsourcing are common in mainstream social computation. Even such basic concepts, however, resist standardisation. There is no single “best” way to crowdsource and not even an exhaustive set of such methods, since we could always invent new ways of getting together to crowdsource the answer to a question. We do not want to have to write a new social computation system every time we want a different style of crowdsourcing; we would prefer to have a single, generic social computation system to which we could describe different forms of crowdsourcing and have it run them. We can get this far with emerging languages and tools but we want more than this for our broader view of social computation. We would like to inter-relate the outcomes of our social computations so we can build new computations on top of the relevant data generated by others. Furthermore, we want to maintain provenance data that allow us to summarise the context in which key data were derived - thus providing more traction for automated reasoning systems to perform more sophisticated analytics on the mass of data begin derived from our generic system and to supply a foundational layer for accountable information[17].

In what follows, we will return to a specific scenario illustrating this generic problem. To retain precision without wasting space, we make this scenario as simple as possible while retaining all the features of the general problem. Suppose we have someone, x, who wants to crowdsource an answer to some question, q. To do this, x crowdsources the answer indiscriminately to 3 other people (who turn out to be $x1$, $x2$ and $x3$). However, $x2$ and $x3$ already made friends (using a separate social computation) and $x3$ uses a third social computation to crowdsource his answer from his immediate friends (in this case just $x2$). We would like a system that will run all these social computations and allow sharing of data appropriately (such as in the use of the friendship relation from the second computation to establish the crowdsourced answer from the third computation. Furthermore, we would like to be able to extract from these computations their key data accompanied by a representation of sufficient provenance meta-data to be able to determine useful relationships between shared data (such as the fact that the answers from $x2$ and $x3$ in x’s crowdsourced computation were not independent of each other because in a separate computation $x3$’s answer was derived from $x2$’s response.

1.2. Social Computation and Formal Languages

Languages relevant to social computation continue to be developed. The principal groups of such languages are:

Languages oriented to crowdsourcing These languages give engineers an interface to crowdsourcing architectures (*e.g.* Turkit[6] connects to Mechanical Turk) so that one can program a social computation via a high-level language. The process specifications introduced later in this paper serve a similar purpose.

Languages for sharing workflows These are general-purpose process languages that are used to provide a shared view of standard workflows across a community (*e.g.* SCUFL was used as an underlying process language in MyExperiment for sharing designs of scientific experiments[10]). Similarly, the process descriptions we introduce later in this paper are designed to be shared (although the means of sharing is different).

Languages for specifying data provenance The aim of this sort of language is to provide a means to annotate data with information related to its origin (*e.g.* PROV[9] might be used to annotate personal data that is shared socially and, when it is shared, to allow others better to understand its origin). We describe later how one might link provenance-related data to process specification in support of social computation.

Languages for specifying Web services It is, arguably, possible to imagine social computation partly as a service choreography problem where people, as well as computers, provide services (*e.g.* OWL-S[7] might be used to specify human services and automated matchmakers might be built to help identify the humans most appropriate to a given task). This level of specification relates to the discovery phase of computation that we introduce later.

Languages for describing the permitted states of social interactions The purpose of these languages is to characterise standard types of social interaction via state specification models (*e.g.* finite state machines are used in ISLANDER[4] to specify the permitted transitions between roles and scenes in multi-agent interactions). The process language we use later in this paper has been used for a similar purpose (and a translator exists within ISLANDER to our language).

Languages for describing human interaction A wide variety of formal languages have been used to model particular aspects of human behaviour when interacting (*e.g.* Dialogue Games[8] are specification systems in which different aspects of human argument may be represented). Often these specifications are not directly executable but in earlier work we have shown how to provide executable versions of some of these specification systems (such as Dialogue Games) using our process language.

1.3. Social Computation and System Architectures

Many system architectures have been harnessed for social computation - indeed any system architecture that allows people to network could in principle be employed of this purpose. There are, however, a smaller number of overarching approaches to system design:

Driving computation through the social network. This is the architecture that supports the giant social networking systems such as Facebook or LinkedIn. The social network in such systems provides the spine on which other computations are supported.

Making the social element invisible. Social computation sometimes is embedded within some other, larger activity in such a way that people using the larger system may be unaware that there is a social element to it. Systems such as ReCaptcha (where human effort to assist in OCR of historical texts is obtained in the guise of personal identification) are examples of this.

Socially enhancing personal data. Humans are often motivated, with the right tools, to acquire and maintain data about themselves. By performing useful analytics across such data when it is shared across a social network it is sometimes possible to generate a virtuous cycle in which more sharing leads to better analytical results which in turn promotes more sharing. This is a driver for systems such as Patients Like Me.

These approaches have typically been taken separately in different systems but we will demonstrate later in this paper that they can be accommodated within a single, generic architecture. This is important if we wish eventually to broaden engineering of such systems to tackle larger and more sophisticated problems that may need a combination of approaches within a single framework. First, however, we introduce the concept of a social artifact, which will form the basis of specification for all computations in our system.

2. Social Artifacts

A social artifact is a data structure specifying the state of a social process. We take advantage of the normal duality in executable specifications between the view of a social artifact as a document, with a semantics independent of any specific machine, and the view of it as a data structure that can be processed by specific computational mechanisms that interact with the real world (in other words, we supply interpreters for these data structures in a social computation engine). In a completely open social machine, all social artifacts are visible to everyone. In practice, as we later discuss, we may wish to constrain openness to protect the individual from the machine and from other individuals. Social artifacts change, and multiply, as a consequence of the computational mechanisms that act upon them. In Section 3 we describe a general purpose mechanism for updating social artifacts and, as a consequence of this, producing social computations.

The language we use to specify social artifacts is the Lightweight Social Calculus (LSC). This has many features in common with the Lightweight Coordination Calculus (LCC)[15] which has been used in a variety of contexts to describe coordination between peers in distributed systems [14]. It is not intended to be the last word on social computation languages (these will continue to evolve and diversify as the area grows) but it is intended to capture the essence of these sorts of computations while also being executable via the generic operational definition given below. Although we do not dwell on the systems engineering aspects of LSC itself, readers should be aware that an interpreter for the language (written in Prolog) exists and there is also a separate system embedding a version of LSC that operates using Twitter as a communication mechanism.

The syntax for LSC is given in Figure 2. The purpose of the language is to specify the permitted sequences of communication within a social group (that is, a group of people interacting together in some social way). Later, we show how computation within the social group can be understood as manipulations on that specification (and it is in this sense that LSC is an executable specification language). The effort of computation when a LSC specification is used operationally is through the actors operating locally. LSC is agnostic to the nature of these actors (they could be humans or automatons - it is not always possible to distinguish on the open internet) but the anticipated use of LSC is in cases where actors typically are humans. Actors connect to the LSC specification by satisfying conditions associated with elements of the clauses of the LSC specification. Each clause defines a role in the computation, so to enter a computation an actor must choose an appropriate role, and all synchronisation between roles is through communication between clauses. This means that, when used as an executable specification language, each clause can be used independently as the "script" of the particular actor that is engaged with that role.

Social artifacts specify social interactions in two ways. The first of these is to describe **potential interactions** that may in future be performed. For example, a basic op-

$$
\begin{aligned}
Artifact &:= \{Clause, \ldots\} \\
Clause &:= Actor :: Def \\
Actor &:= a(Role, Id) \\
Def &:= E \mid Def\ then\ Def \mid Def\ or\ Def \\
E &:= Event \mid Event \leftarrow C \mid C \leftarrow Event \mid C \leftarrow Event \leftarrow C \\
Event &:= Actor \mid Communication \mid null \\
Communication &:= Content \Rightarrow Actor \mid Content \Leftarrow Actor \\
C &:= \langle \ldots Item \ldots \rangle \\
Item &:= k(Term) \mid e(Term) \mid i(Term) \mid k(Term)\ \leftarrow\ e(Term) \\
Role &:= Term \\
Content &:= Term
\end{aligned}
$$

Where *null* denotes an event which does not involve communication; *Term* is a structured term in Prolog syntax and *Id* is either a variable or a unique identifier for the actor. $M \Rightarrow A$ denotes that content, M, is communicated to actor A. $M \Leftarrow A$ denotes that content, M, is received from actor A. Events can be associated with pre and post conditions, so $C_2 \leftarrow E \leftarrow C_1$, says that event E, can take place if C_1 is true and that C_2 must hold after E has occurred. All conditions are understood within the context of the actor following the clause in which the condition appears. Conditions of the form $k(Term)$ denote that data item*Term* is assumed to be believed by the appropriate actor. Conditions of the form $e(Term)$ describe how the appropriate actor engages with the social interaction. Conditions of the form $i(Term)$ are internal computations performed by the LSC interpreter. The *then* operator represents sequence. The *or* operator represents committed choice.

Figure 2. Syntax of LSC

eration in social computation systems is to develop social links. In LSC we can specify the simplest form of this type of social interaction using the single clause given in definition 1. Informally, this definition says that an actor, X, can play the role of a friend of actor Y if X offers an *invite* to Y (who accepts the role of a friend of X) and receives an *accept* or (conversely) if X receives an *invite* from Y and offers an *accept*. The communication of an *invite* by X is conditional on X actually wanting to invite Y (which is denoted by the condition $invites(Y)$). Similarly the communication of an *accept* by X is conditional on X actually wanting to accept an invitation from Y (which is denoted by the condition $accepts(Y)$).

$$
\begin{array}{l}
a(friend, X) :: \\
\quad \left(\begin{array}{l} invite \Rightarrow a(friend, Y)\ \leftarrow\ \langle e(invites(Y))\rangle\ then \\ \langle k(friends(X,Y))\rangle\ \leftarrow\ accept \Leftarrow a(friend, Y) \end{array} \right) \\
\quad or \\
\quad \left(\begin{array}{l} invite \Leftarrow a(friend, Y)\ then \\ \langle k(friends(X,Y))\rangle\ \leftarrow\ accept \Rightarrow a(friend, Y)\ \leftarrow\ \langle e(accepts(Y))\rangle \end{array} \right)
\end{array}
\tag{1}
$$

The second way in which we use specification is to describe **actual interactions** that have been performed by specific collections of actors. For example, if (using the interaction specification in definition 1) actors $p1$ and $p2$ established a friendship link then the resulting specification of their specific interaction might be as shown in definition 2. This is a ground version of definition 1 in which the variables have been replaced by constants identifying the actors involved. Notice that we have two clauses in definition 2, each an instance of definition 1 used by the appropriate actor. In Section 3.2 we explain social computation as a derivation of actual from prospective clauses (*i.e.* deriving what has been done from what might be done).

$$\begin{array}{l} a(friend, p1) :: \\ \quad invite \Rightarrow a(friend, p2) \ \leftarrow \ \langle e(invites(p2))\rangle \ then \\ \quad \langle k(friends(p1, p2))\rangle \ \leftarrow \ accept \Leftarrow a(friend, p2) \\ \\ a(friend, p2) :: \\ \quad invite \Leftarrow a(friend, p1) \ then \\ \quad \langle k(friends(p2, p1))\rangle \ \leftarrow \ accept \Rightarrow a(friend, p1) \ \leftarrow \ \langle e(accepts(p1))\rangle \end{array} \tag{2}$$

From actual interactions we can infer from each clause the knowledge associated with the corresponding actor. For example, from definition 2 we can infer that $p1$ knows $friends(p1, p2)$ and that $p2$ knows $friends(p2, p1)$. Depending on the privacy policies applying to each clause, this knowledge might be common knowledge (shared by $p1$, $p2$ and perhaps third parties). In Section 3.4 we look at this in more detail.

3. Core Cycle of Computation

The core cycle of computation is depicted in Figure 3. At the centre of the computation is the repository of social artifacts. All of the basic operations of computation are transformations on individual artifacts within this repository, achieved through the following cycle of actions:

Discover: An actor, A, locates a relevant social artifact, $(\mathscr{P}, M)$, in the repository where $\mathscr{P}$ is the state of the interaction and M is the set of open communication events (the messages produced by actors that have not yet been read by their intended recipients). We write this as $\mathbb{D}_A((\mathscr{P}, M))$.

Engage: Actor A is motivated to choose a role in the interaction and chooses an appropriate clause, S, from $\mathscr{P}$. We write this as $\mathscr{P} \ni^A S$.

Act: The actor interacts according to the social norms expressed in S, producing the new interaction state, S_n. In doing so, the communication events associated with $\mathscr{P}$ may change (through consumption and/or generation of communications by A) to give the new event set M_n. This creates a new social artifact, $(\mathscr{P}_n, M_n)$ that is recorded in the repository. We write this as $S \xrightarrow{A,\mathscr{P},M,M_n} S_n, \mathscr{P} \overset{S,S_n}{\rightarrowtail} \mathscr{P}_n$

A full cycle of social computation, starting with the discovery of artifact $(\mathscr{P}_1, M_1)$ and concluding with new artifact $(\mathscr{P}_2, M_2)$ is then the sequence:

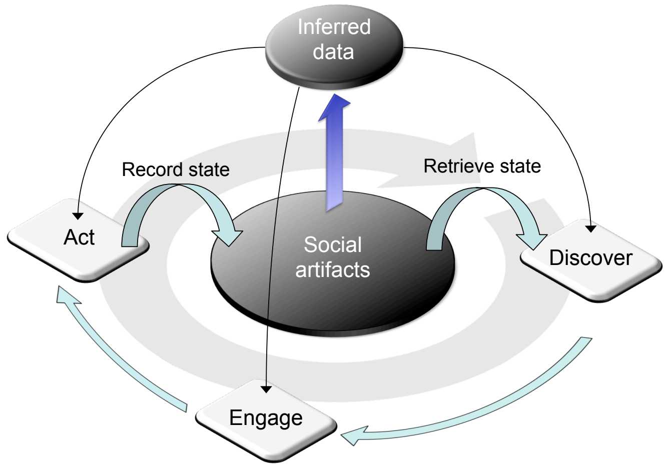

Figure 3. Basic cycle of computation in LSC

$$\mathbb{D}_A((\mathscr{P}_1, M_1)),\ \mathscr{P}_1 \ni^A S_1,\ S_1 \xrightarrow{A, \mathscr{P}_1, M_1, M_2} S_2, \mathscr{P}_1 \overset{S_1, S_2}{\rightarrowtail} \mathscr{P}_2$$

Repeating this cycle for $(\mathscr{P}_1, M_1), \ldots, (\mathscr{P}_i, M_i)$ gives a sequence of computations for the social artifact.

While individual people are interacting via this discover-engage-act cycle, automated mechanisms are used to infer additional data from the interaction states. These derived data are fed into each step of the cycle with the aim of informing and assisting individual actors. We describe this cycle (and its relationship to data provenance) in Section 3.4 but first we describe in more detail the discover-engage-act cycle.

3.1. Discovery

A key advantage of high-level specification of social artifacts is that these specifications may be accessed within a variety of architectures. For example:

Via a Web browser where artifacts are annotated to increase the likelihood of their being discovered by semantic web search engine and an action+engagement mechanism is built into the browser. An example of this is the OKeilidh system[1].

Via peer to peer recommendation where artifacts are held on peers and query routing mechanisms used to search for them across the peer network. An example of this is the OpenKnowledge system[14].

Via interaction with existing social media systems where an interpreter for LSC specifications posts adverts to engage with them to the media stream and listens for responses. An example of this is our use of Twitter as a media stream for LSC, using hashtags to advertise and respond.

Although the specific choice of architecture (and artifact annotation) may vary, there is a major difference, across architectures, in the way actors discover artifacts. The simplest form of introduction (and the one most commonly practiced) is for actors to be invited directly. The following example demonstrates this.

Having established friend-of-friend links via use of the computation specified in definition 1, we might want to use this network to crowdsource the answer to some question. Definition 3 defines one form of basic crowdsourcing, using immediate friends. Here the social interaction is initiated by an actor in the role of *coordinator* who polls, in turn, each of his/her friends for the answer to question, Q, and then decides the answer, A, based on the set, S of replies received. The role of each *participant* is simply to be receive $request(Q)$, from the coordinator and offer $reply(A)$, if it is able to respond.

$$
\begin{array}{l}
a(coordinator, C) :: \\
\quad \langle k(answer(Q,A)) \leftarrow e(decide(S,A)) \rangle \leftarrow a(collector(Q,F,S),C) \leftarrow \\
\qquad \langle set(Y,k(friends(C,Y)),F), e(choose\_question(Q)) \rangle \\
\\
a(collector(Q,F,S),C) :: \\
\quad \left(\begin{array}{l} request(Q) \Rightarrow a(participant,Y) \leftarrow \langle i(F=[Y|F_r]) \rangle \; then \\ \langle i(S=[A|S_r]) \rangle \leftarrow reply(Q,A) \Leftarrow a(participant,Y) \; then \\ a(collector(Q,F_r,S_r),C) \end{array} \right) \\
\quad or \\
\quad null \leftarrow \langle i(F=[]), i(S=[]) \rangle \\
\\
a(participant,Y) :: \\
\quad request(Q) \Leftarrow a(collector,C) \; then \\
\quad \langle k(answer(Q,A)) \rangle \leftarrow reply(Q,A) \Rightarrow a(collector,C) \leftarrow \langle e(respond(Q,A)) \rangle
\end{array}
\tag{3}
$$

The approach of Definition 3 is not the only way to crowdsource. We could use a less discriminating approach such as the one in Definition 4. This interaction does not reference the friend relations known to the coordinator. Instead, the coordinator accepts replies from any actor that responds (collecting the first N responses as its sample).

$$
\begin{array}{l}
a(coordinator, C) :: \\
\quad \langle k(answer(Q,A)) \leftarrow e(decide(S,A)) \rangle \leftarrow a(collator(Q,N,S),C) \leftarrow \\
\qquad \langle e(choose\_question(Q)), e(sample\_size(N)) \rangle \\
\\
a(collator(Q,N,S),C) :: \\
\quad \left(\begin{array}{l} \langle i(S=[A|S_R]) \rangle \leftarrow reply(Q,A) \Leftarrow a(responder(Q),Y) \leftarrow \langle i(N>0) \rangle \; then \\ a(collator(Q,N_r,S_r),C) \leftarrow \langle i(N_r = N-1) \rangle \end{array} \right) \\
\quad or \\
\quad null \leftarrow \langle i(N=0), i(S=[]) \rangle \\
\\
a(responder(Q),Y) :: \\
\quad reply(Q,A) \Rightarrow a(collator,C) \leftarrow \langle k(answer(Q,A)) \leftarrow e(respond(Q,A)) \rangle
\end{array}
\tag{4}
$$

As in conventional specification, a wide range of different LSC definitions could be constructed, even for a comparatively simple task like crowdsourcing a question. We are unlikely to standardise on a single, canonical specification of a social activity. On the other hand, we do want to take advantage of data sharing between social activities (so that, for example, the specification of Definition 3 relies on friendship data that can be produced by Definition 1). This is the reason why we distinguish (syntactically, using the term $k(X)$) items of knowledge acquired by actors via interaction with social artifacts.

3.2. Acting

Recall from the start of Section 3 that a social artifact is a set, $\mathscr{P}$, of LSC clauses plus a set of open communication events, M_i, initiated by previous interaction with $\mathscr{P}$ (see Figure 2 for syntax). A change in state of $\mathscr{P}$ takes place when an actor, A, selects a clause, S_1, from $\mathscr{P}$ and translates it to the new state S_2. In performing this update, some communication events in M_i may be removed (because they are received by A) and others may be added (because A initiates them), generating the new communication set M_o. We write this as $S_1 \xrightarrow{A,\mathscr{P},M_i,M_o} S_2$ in the formal definition of state transition in Figure 4. This basic notion of state transition supplies us not only with the basis to support primary social interactions (for the task in hand) but also with a basis for secondary support for engagement (Section 3.3) and data inference (Section 3.4).

3.3. Engagement

Engagement of an actor, A, with a social artifact, $\mathscr{P}$, occurs when the actor commits to a role in $\mathscr{P}$. In Section 3 we wrote this as $\mathscr{P} \ni^A S$ where S is the clause of the interaction with which A engages. That engagement is likely to depend on a variety of things: the ability of the actor to identify the role as being appropriate in his/her current context; the incentive to engage with the role; and trust that other actors engaging with the artifact will play their roles appropriately. This is complex and extensively studied from many human perspectives (economic, social, psychological) but we are concerned with providing a broad infrastructure for the (computer based) social interaction that may underpin these human processes.

The actor must at least be able to form a view of what the social artifact is intended to achieve, and this will inform the decision to join, as well as further questions about the quantity of resources to commit. The artifact will in general focus on a particular goal G (which may be designed in, or may have emerged via interaction between its constituent members). Hence one obvious case would be when the actor shares the goal G; his interests are therefore served by making a positive contribution to the artifact. But he has to be sure that his contribution would be positive before he took steps to join (a football supporter wants his team to win, but does not for that reason join the team).

Yet this is not the only possibility. ReCAPTCHA is a social artifact whose goal is to help Google transcribe books, which is not the goal of very many of its users. The actors immediate goal is to prove that he or she is human, in pursuit of some wider goal (*e.g.* of gaining admittance to a website), and hence ReCAPTCHA is designed to achieve its own goal by presenting itself as a solution to actors immediate access problems. Other social artifacts, such as the winning DARPA balloon challenge, offer financial or other incentives to a public which would otherwise be indifferent. Where an artifact defines a community, the goal of the actor may simply be to join the community, and to serve that wider goal the actor may simply adopt the goal of the artifact (just as the goal of a golfer is not to place this ball in that hole a quarter of a mile away — golf would be a

$S_1 \xrightarrow{A,\mathscr{P},M_i,M_o} S_2$ denotes a transition in the social computation from the state represented by clause S_1, taken from social artifact, P, to the state represented by clause S_2 via the actions of actor A. M_i and M_o are sets of communication events remaining to be processed at the start and end of the transition. Definition:

$$\begin{array}{ll}
A_n :: D \xrightarrow{A,\mathscr{P},M_i,M_o} A_n :: E & \text{if } D \xrightarrow{A_n,\mathscr{P},M_i,M_o} E \\
T_1 \ or \ T_2 \xrightarrow{A,\mathscr{P},M_i,M_o} E & \text{if } T_1 \xrightarrow{A,\mathscr{P},M_i,M_o} E \\
T_1 \ or \ T_2 \xrightarrow{A,\mathscr{P},M_i,M_o} E & \text{if } T_2 \xrightarrow{A,\mathscr{P},M_i,M_o} E \\
T_1 \ then \ T_2 \xrightarrow{A,\mathscr{P},M_i,M_o} E \ then \ T_2 & \text{if } \neg\odot(T_1),\ T_1 \xrightarrow{A,\mathscr{P},M_i,M_o} E \\
T_1 \ then \ T_2 \xrightarrow{A,\mathscr{P},M_i,M_o} T_1 \ then \ E & \text{if } \odot(T_1),\ T_2 \xrightarrow{A,\mathscr{P},M_i,M_o} E \\
X \Leftarrow A_1 \xrightarrow{A,\mathscr{P},M_i,M_o} c(X \Leftarrow A_1, \psi) & \text{if } M_o = M_i \setminus \{m(A, X \Leftarrow A_1, \psi)\} \\
M \Rightarrow A_1 \xrightarrow{A,\mathscr{P},M_i,M_o} c(M \Rightarrow A_1, \psi) & \text{if } \mathbb{P}(M,\psi),\ M_o = M_i \cup \{m(A_1, X \Leftarrow A, \psi)\} \\
a(R,I) \xrightarrow{A,\mathscr{P},M_i,M_o} a(R,I) :: E & \text{if } \mathscr{P} \looparrowright a(R,I) :: D,\ D \xrightarrow{A,\mathscr{P},M_i,M_o} E \\
T \leftarrow C \xrightarrow{A,\mathscr{P},M_i,M_o} E \leftarrow c(C,\psi) & \text{if } \mathbb{K}(A,C,\psi),\ T \xrightarrow{A,\mathscr{P},M_i,M_o} E \\
C \leftarrow T \xrightarrow{A,\mathscr{P},M_i,M_o} c(C,\psi) \leftarrow E & \text{if } T \xrightarrow{A,\mathscr{P},M_i,M_o} E,\ \mathbb{K}(A,C,\psi)
\end{array}$$

$\odot(T)$ denotes that an interaction term, T has been covered by the preceding interaction (we say that it is closed). Definition:

$$\begin{array}{ll}
\odot(c(X,\psi)) & \\
\odot(A \leftarrow B) & \text{if } \odot(A) \text{ and } \odot(B) \\
\odot(A \ then \ B) & \text{if } \odot(A) \text{ and } \odot(B) \\
\odot(X :: D) & \text{if } \odot(D)
\end{array}$$

$\mathscr{P} \looparrowright X$ is true if clause X appears in the interaction framework $\mathscr{P}$.
$\mathbb{P}(X,\psi)$ is true when the actor assigns context description ψ to X.
$\mathbb{K}(A,C,\psi)$ is true if C can be satisfied from the actor's current state of knowledge, given whatever computational system the actor deploys internally. Definition:

$$\begin{array}{ll}
\mathbb{K}(A,k(C),\psi) & \text{if } C \text{ is known by } A \text{ and } \mathbb{P}(C,\psi) \\
\mathbb{K}(A,e(C),\psi) & \text{if } C \text{ is satisfied through interaction with } A \text{ and } \mathbb{P}(C,\psi) \\
\mathbb{K}(A,C,\emptyset) & \text{if } C \text{ is a predefined function that the system can perform} \\
\mathbb{K}(A,C_1 \wedge C_2,\psi_1 \cup \psi_2) & \text{if } \mathbb{K}(A,C_1,\psi_1) \text{ and } \mathbb{K}(A,C_2,\psi_2)
\end{array}$$

Figure 4. State transition for a clause of a social artifact

very inefficient way of doing that — but rather she adopts that goal because of her desire to play a game of golf). The Zooniverse provides examples of this type of artifact, where participants who have no intrinsic interest in galaxy formations or the sex life of worms have the wider goal of taking part in a scientific project, promoting science and hanging out in a community of citizen scientists.

Hence the artifact needs to communicate to potential users, in order to attract (the right) actors, but communicating its ultimate goal G may on occasion be counterproductive. The content of the communication must depend on whether the actor is attracted by the aims of the artifact itself, or instead by what the artifact will do for the actor.

Although part of an actor's decision to engage with a role may be internal to his/her own thoughts, the choice may also be influenced by external interactions with other actors. These interactions can also be enacted via the state transition mechanism of Figure 4, thus bring them also within the social computation. For example, an actor, X, may wish to decide whether to engage a clause, S (so $\mathscr{P} \ni^X S$) by requesting approval from some authority, Y, by playing the role of $a(assessor(S,R),X)$, as defined in interaction 5, to obtain a recommendation, R, on whether or not to engage with the interaction defined by S. The choice of whether or not to engage might then be based on the actor's (internal) judgement based on (socially derived) R.

$$\begin{array}{l}
a(assessor(C,R),X) :: \\
\quad request(recommendation(C)) \Rightarrow a(authority,Y) \ \leftarrow \ \langle e(choose\_authority(Y))\ then \\
\quad recommend(R) \Leftarrow a(authority,Y) \\
\\
a(authority,Y) :: \\
\quad request(recommendation(C)) \Leftarrow a(assessor,X)\ then \\
\quad recommend(accept) \Rightarrow a(assessor,X) \ \leftarrow \ \langle e(recommends(C,R))\rangle
\end{array} \tag{5}$$

Having moved this part of the decision process (partially) into the social computation realm, we have the flexibility (as before) to define a wide variety of social mechanisms for engagement. For example, definition 6 provides an alternative definition for *assessor* by re-using the crowd sourcing interaction from definition 8.

$$\begin{array}{l}
a(assessor(C,R),X) :: \\
\quad \langle e(decide(S,R))\rangle \ \leftarrow \ a(collator(C,N,S),X) \ \leftarrow \ \langle e(sample\_size(N))\rangle
\end{array} \tag{6}$$

This move has particular value in describing the mechanisms which support the decision of the actor to trust (or not) the other elements of a social artifact to deliver their promised contributions. Trust is essentially the belief that the would-be trustee is sufficiently trustworthy in a given context, and so is an internal decision (or involuntary attitude) of the actor. However, there are many ways of socialising trust, to increase the likelihood that trust effectively correlates with trustworthiness. Reputations store accessible records of past performance; recommendations are the opinions of others who have been in the role that the actor contemplates occupying; sanctions give the actor access to redress which incentivises the would-be trustee not to defect; transparency allows the actor to monitor the would-be trustees performance.

Given that the notion of an actor in LSC is agnostic to whether the actor is a human or an automaton, there is an issue of what to do when trust relies on the actor being human (because some uniquely human measure of trust needs to be applied). Although LSC itself cannot verify whether or not actors are human, it could (via the constraints attached to roles) attach to a separate claims systems that aim to provide such verification.

As well as trusting human actors in LSC-based systems it is necessary to trust the social artifacts themselves. Earlier work [11] supplies a system for checking social artifacts against specifications made by an actor of the commitments and obligations he/she requires of the interactions they contain. This contrasts with systems for propagating preference conditions for use of shared data along with data being shared (see, for example [12]). Importantly, social artifacts expressed in LSC are themselves data so they, themselves, constitute personal data to be managed, with the architectural issues discussed in (for example) [2].

3.4. Inferring Structured Common Knowledge

We now demonstrate by example how our style of social computation allows us to infer useful information about the social structure of knowledge acquired via interaction between actors. We return to the scenario introduced in Section 1 but this time enact it using the social artifacts given in definitions 1, 3 and 4.

The first social interaction establishes $x2$ and $x3$ as friends via Definition 1. This interaction constructs the ground social artifact shown in Definition 7.

$$\left(\begin{array}{l} a(friend,x2) :: \\ \quad invite \Rightarrow a(friend,x3) \ \leftarrow \ \langle e(invites(x3))\rangle \ then \\ \quad \langle k(friends(x2,x3))\rangle \ \leftarrow \ accept \Leftarrow a(friend,x3) \\ a(friend,x3) :: \\ \quad invite \Leftarrow a(friend,x2) \ then \\ \quad \langle k(friends(x3,x2))\rangle \ \leftarrow \ accept \Rightarrow a(friend,x2) \ \leftarrow \ \langle e(accepts(x2))\rangle \end{array}\right) \tag{7}$$

Meanwhile, actor x decides to crowdsource the answer to question q via Definition 4. This involves collecting three responses from which x then decides the answer ($a2$) based on the responses from $x1$, $x2$ and $x3$. This interaction constructs the ground social artifact shown in Definition 8.

$$\left(\begin{array}{l} a(coordinator,x) :: \\ \quad \langle k(answer(q,a2)) \ \leftarrow \ e(decide([a1,a2,a2],a2))\rangle \ \leftarrow \\ \qquad \left(\begin{array}{l} a(collator(q,3,[a1,a2,a2]),x) :: \\ \quad reply(q,a1) \Leftarrow a(responder(q),x1) \ then \\ \quad a(collator(q,2,[a2,a3]),x) :: \\ \qquad reply(q,a2) \Leftarrow a(responder(q),x2) \ then \\ \qquad a(collator(q,1,[a3]),x) :: \\ \qquad\quad reply(q,a2) \Leftarrow a(responder(q),x3) \ then \\ \qquad\quad a(collator(q,0,[]),x) \end{array}\right) \\ \qquad \leftarrow \ \langle e(choose\_question(q)), e(sample\_size(3))\rangle \\ \\ a(responder(q),x1) :: \ reply(q,a1) \Rightarrow a(collator,x) \ \leftarrow \ \langle k(answer(q,a1)) \ \leftarrow \ e(respond(q,a1))\rangle \\ a(responder(q),x2) :: \ reply(q,a2) \Rightarrow a(collator,x) \ \leftarrow \ \langle k(answer(q,a2)) \ \leftarrow \ e(respond(q,a2))\rangle \\ a(responder(q),x3) :: \ reply(q,a2) \Rightarrow a(collator,x) \ \leftarrow \ \langle k(answer(q,a2)) \ \leftarrow \ e(respond(q,a2))\rangle \end{array}\right) \tag{8}$$

However, actor $x3$ had also crowdsourced an answer to question q using Definition 3 to contact his immediate friends. This interaction constructs the ground social artifact shown in Definition 9.

$$\left(\begin{array}{l}
a(coordinator, x3) :: \\
\quad \langle k(answer(q,a2)) \leftarrow e(decide([a2],a2))\rangle \leftarrow \\
\qquad \left(\begin{array}{l}
a(collector(q,[x2],[a2]),x3) :: \\
\qquad request(q) \Rightarrow a(participant, x2)\ then \\
\qquad reply(q,a2) \Leftarrow a(participant, x2)\ then \\
\qquad a(collector(q,[],[]),x3)
\end{array}\right) \\
\quad \leftarrow \langle k(friends(x3,x2)), e(choose\_question(q))\rangle \\
\\
a(participant, x2) :: \\
\quad request(q) \Leftarrow a(collector, x3)\ then \\
\quad \langle k(answer(q,a2))\rangle \leftarrow reply(q,a2) \Rightarrow a(collector, x3) \leftarrow \langle e(respond(q,a2))\rangle
\end{array}\right) \tag{9}$$

The social artifacts we have constructed through interaction in Definitions 7, 8 and 9 allow us to extract data and meta-data created during the interaction (illustrated in the basic computation cycle of Figure 3). We can directly extract data believed to be known by actors as a consequence of completed interactions; for example, from Definition 7 we can infer that actor $x2$ knows $friends(x2,x3)$ and that actor $x3$ knows $friends(x3,x2)$. This direct data is, however, of limited value unless accompanied by meta-data describing the context in which it was obtained; for example, the data that actor x knows $answer(q,a2)$ is not of much use without knowing how x acquired this answer. It is therefore necessary to extract this sort of meta-data from social artifacts.

An important aspect of social data is its provenance. Various languages exist for describing data provenance but we have chosen to use the PROV language as an exemplar in this paper because it was developed with the Web in mind, so if we have meta-data expressed in PROV we have access to the efforts already made to make PROV expressions accessible on the Web in a standard way. We can automatically infer PROV specifications (in a subset of the PROV language) from LSC specifications as summarised below. This is a work in progress, so the PROV relations described here are not all that we could infer from the LSC specifications (in particular, we are currently working on a representation of entities that is more faithful to the PROV method) but the example given here demonstrates the principle, which is the limit of our intention in this paper.

At the core of PROV are three classes of entity and eight relations, as shown in Table 1. Of the core types and relations of Table 1, two are difficult to infer from LSC specifications: wasInvalidatedBy(Entity, Activity) is hard to infer because in LSC there is no specific invalidation construct (one can only access data or create it); actedOnBehalfOf(Agent1, Agent2) is hard to infer because in LSC there is no specific delegation construct (although sophisticated interactions between actors can be defined which might involve delegation, but it is hard to know when that occurs without an additional model of delegation in terms of interaction). The remainder of the types and relations can be inferred automatically, as described in Table 2.

As a demonstration of this method, we can use Table 2 to convert the social artifacts from Definitions 7, 8 and 9 into the PROV graphs shown in Figures 5, 6 and 7 respectively. In these visual depictions of PROV relations, the yellow ovals represent entities;

Table 1. Core PROV types and relations

PROV construct	Informal meaning
entity(Entity)	Something being created, adapted or manipulated.
activity(Activity)	Something that occurs over time and acts upon or with entities.
agent(Agent)	Something that bears responsibility for an activity.
wasDerivedFrom(Entity2, Entity1)	Transformation or constriction of Entity2 from Entity1.
wasGeneratedBy(Entity, Activity)	Production of an Entity by an Activity.
used(Activity, Entity)	Utilisation of an Entity by an Activity.
wasInformedBy(Activity2, Activity1)	The use of information in Activity1 to inform Activity2.
wasInvalidatedBy(Activity, Entity)	Making an Entity unavailable for further use through an Activity.
wasAttributedTo(Entity, Agent)	Ascribing an Entity to an Agent.
wasAssociatedWith(Activity, Agent)	Assigning responsibility for an Activity to an Agent.
actedOnBehalfOf(Agent2, Agent1)	Delegating responsibility to Agent2 by Agent1.

Table 2. Core PROV types and relations

PROV construct	Means of inference from LSC specification
entity(Entity)	Any instance of $Entity$ in a term $k(Entity)$
activity(Activity)	Any Activity subterm of the form $M \Rightarrow A$, $M \Leftarrow A$ or $e(X)$.
agent(Agent)	Any instance of $Agent$ in a term $a(Role,Agent)$
wasGeneratedBy(Entity, Activity)	Any subterm of the form $k(Entity) \leftarrow Activity$ or of the form $\langle \ldots k(Entity) \ldots \rangle \leftarrow Activity$
used(Activity, Entity)	Any subterm of the form $Activity \leftarrow k(Entity)$ or of the form $Activity \leftarrow \langle \ldots k(Entity) \ldots \rangle$
wasInformedBy(Activity2, Activity1)	Any subterm of the form $Activity1$ *then* $Activity2$ or of the form $Activity1 \leftarrow \langle \ldots e(Activity2) \ldots \rangle$ or of the form $\langle \ldots e(Activity1) \ldots \rangle \leftarrow Activity2$
wasAttributedTo(Entity, Agent)	Any $Agent$ appearing as the principal actor in a clause of the form $a(R,Agent) :: Def$, where Def contains a subterm for which $entity(Entity)$ can be derived.
wasAssociatedWith(Activity, Agent)	Any $Agent$ appearing as the principal actor in a clause of the form $a(R,Agent) :: Def$, where Def contains a subterm for which $activity(Activity)$ can be derived

the purple rectangles are activities; the orange pentagons are agents. For completeness in terms of LSC we have also included in the diagram green lozenges representing the role associated with an agent as part of its attribution or association relation. This an also be derived automatically from an LSC specification because all occurrences of an Agent appear within a term of the form $a(Role,Agent)$ (see third line in Table 2).

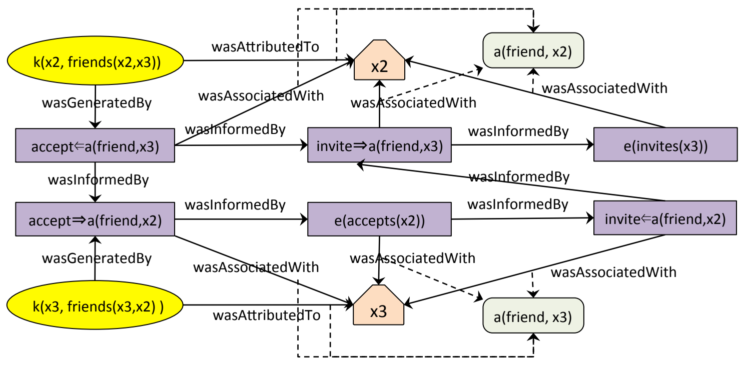

Figure 5. Provenance graph derived from Definition 7

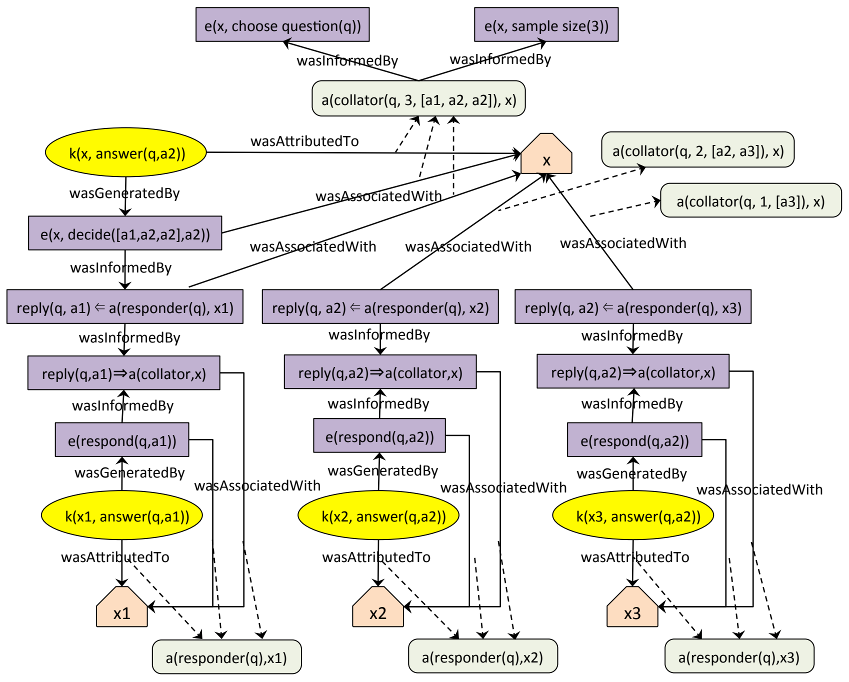

Figure 6. Provenance graph derived from Definition 8

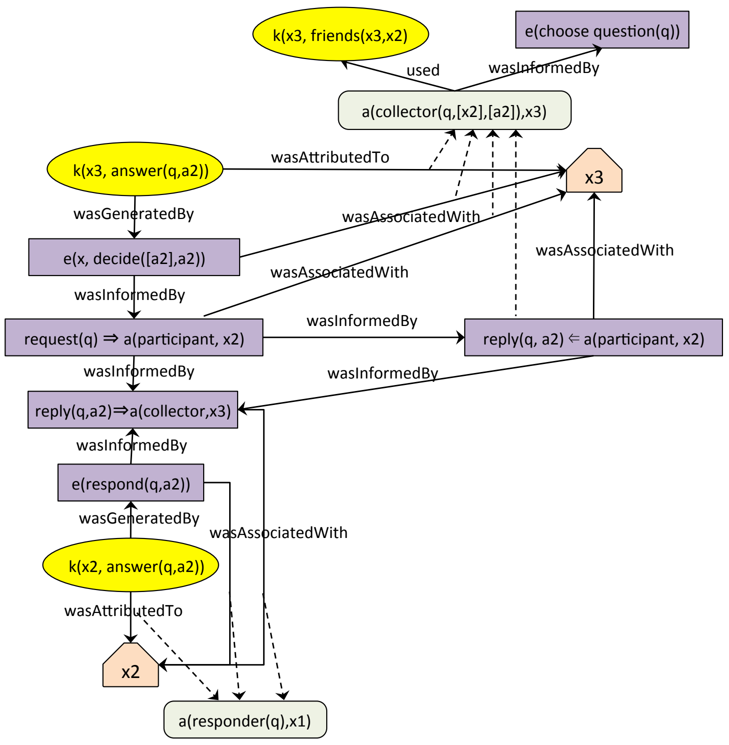

Figure 7. Provenance graph derived from Definition 9

4. Conclusions

The contribution of this paper is to demonstrate how, with the aid of comparatively well understood formal methods assembled for the new engineering setting of social computation, we can produce an infrastructure for a wide class of social computation using mechanisms originally developed for knowledge sharing in open, peer-to-peer environments. This is important for two reasons: to give a uniform technical basis for supporting social computations, so that both computations and data can more readily be shared, and to disrupt the current trend for social computations to be centralised within institutions with resulting loss of control over personal data.

We have tackled these issues by giving (in Section 2) a generic language in which specifications for social computations may be described. These "social artifacts" may then be used computationally (via the mechanisms summarised in Section 3) to provide a core cycle of computation that allows people to discover, engage and act with social artifacts while, in the process, deriving data established through the social interaction. These data can be expressed (through automatic translation) within standard formats that allow them to be combined with other "semantic web" data and used to infer further knowledge about the social computation. In our example of Section 3.4 the data we derived was provenance-related, so we were interested in inferring dependencies between data and the events (and actors) creating data. We chose PROV because a community of engineers exists around the PROV language and by conforming to it we increase the chances that tools developed by that community to analyse PROV could be used for the our data.

The mechanisms described in this paper do not provide the last word on generic social computation - this will take the combined efforts of many researchers across the wide spread of relevant disciplines. We have already seen rapid growth in the number and variety of systems that rely on human social systems for their operation (the lower block of Figure 1). The key issue of scale is whether we can build larger and more complicated systems of social computation (the upper-right block of Figure 1) by growing the complexity of one or more individual, closed systems or whether the route to growth is through greater interoperability between systems through shared data. Our paper pushes, as far as we are able, toward the shared data option - to the extent that the framework for social interaction is itself shared data (and can be used to derive data for systems outside the immediate social computation). We have not demonstrated, however, that our solution actually scales in practice to large problems. Technically, it allows an open community of actors to engage with as many different forms of social interaction as can be described in the LSC language so, in principle, this can operate at large scale. In practice, we need additional mechanisms to discover, engage and act in concert with these LSC specifications. Our earlier work in ontology matching, reputation management and distributed execution with these sorts of specifications provides prototypical mechanisms for these tasks but much work remains to be done in assembling these in a new social context.

Acknowledgements

This work is supported by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant number EP/J017728/1 (SOCIAM: The Theory and Practice of Social Machines) and by the European Communitys Seventh Framework Programme under grant agreement 600854 (Smart-Society: Hybrid and Diversity-Aware Collective Adaptive Systems). We have benefited from discussions with colleagues on both these projects.

References

[1] X. Bai. *Peer-to-Peer, Multi-Agent Interaction Adapted to a Web Architecture*. PhD thesis, School of Informatics, University of Edinburgh, 2013.

[2] J. Bus and C. Nguyen. *Digital Enlightenment Yearbook*, chapter Personal Data Management A Structured Discussion. IOS Press, 2013.

[3] D. Clery. Galaxy zoo volunteers share pain and glory of research. *Science*, 333(6039), 2011.

[4] M. Esteva, D. de la Cruz, and C. Sierra. Islander: an electronic institutions editor. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems*, pages 1045–1052, 2002.

[5] F. Khatib, F. Dimaio, S. Cooper, M. Kazmierczyk, M. Gilski, S. Krzywda, H. Zabranska, I. Pichova, and J. Thompson. Crystal structure of a monomeric retroviral protease solved by protein folding game players. *Nature Structural and Molecular Biology*, 18(10), 2011.

[6] G. Little, L.B. Chilton, M. Goldman, and R.L Miller. Turkit: Human computation algorithms on mechanical turk. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*, pages 57–66. ACM Press, 2010.

[7] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S 1.1, 2004.

[8] P. McBurney and S. Parsons. Games that agents play: A formal framework for dialogues between autonomous agents. *Journal of Logic, Language and Information*, 11(3):315–334, 2002.

[9] Luc Moreau and Paul Groth. *Provenance: An Introduction to PROV*. Morgan and Claypool, September 2013.

[10] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[11] N. Osman and D. Robertson. Dynamic verification of trust in distributed open systems. In *Twentieth International Joint Conference on Artificial Intelligence*, 2007.

[12] S. Pearson and M. Casassa Mont. Sticky policies: An approach for managing privacy across multiple parties. *IEEE Computer*, September, 2011.

[13] G. Pickard, W. Pan, I. Rahwan, M. Cebrian, R. Crane, A. Madan, and A. Pentland. Time critical social mobilisation. *Science*, 334(6055), 2011.

[14] D. Robertson, C. Walton, P. Barker, A. Besana, Y. Chen-Burger, F. Hassan, D. Lambert, G. Li, J. McGinnis, N. Osman, A. Bundy, F. McNeill, F. van Harmelen, C. Sierra, and F. Giunchiglia. Models of interaction as a grounding for peer to peer knowledge sharing. In E Chang, T. Dillon, R. Meersman, and K Sycara, editors, *Advances in Web Semantics, vol 1*. Springer-Verlag, LNCS-4891, 2008.

[15] David Robertson. Multi-agent coordination as distributed logic programming. In *International Conference on Logic Programming*, volume 3132 of *Lecture Notes in Computer Science*, pages 416–430, Sant-Malo, France, 2004. Springer.

[16] L. von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

[17] D. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. Hendler, and G. Sussman. Information accountability. *Communications of the ACM*, 51(6):82–87, 2008.