LSCitter: Building Social Machines by Augmenting Existing Social Networks with Interaction Models

Dave Murray-Rust
Centre for Intelligent Systems and Applications
School of Informatics
University of Edinburgh, Scotland
d.murray-rust@ed.ac.uk

Dave Robertson
Centre for Intelligent Systems and Applications
School of Informatics
University of Edinburgh, Scotland
dr@inf.ed.ac.uk

ABSTRACT

We present LSCitter, an implemented framework for supporting human interaction on social networks with formal models of interaction, designed as a generic tool for creating social machines on existing infrastructure. Interaction models can be used to choreograph distributed systems, providing points of coordination and communication between multiple interacting actors. While existing social networks specify how interactions happen—who messages go to and when, the effects of carrying out actions—these are typically implicit, opaque and non user-editable. Treating interaction models as first class objects allows the creation of electronic institutions, on which users can then choose the kinds of interaction they wish to engage in, with protocols which are explicit, visible and modifiable. However, there is typically a cost to users to engage with these institutions. In this paper we introduce the notion of "shadow institutions", where actions on existing social networks are mapped onto formal interaction protocols, allowing participants access to computational intelligence in a seamless, zero-cost manner to carry out computation and store information.

Categories and Subject Descriptors

H.1 [**Information Systems**]: Models and Principles

General Terms

social machines; coordination; socio-technical systems; social networks

1. INTRODUCTION

The concept of *social machines* denotes an increasingly widespread category of socio-technical systems which support purposeful human interaction [5]. Central to the theory of social machines is an integrated analysis of both the human and computational components, as opposed to a traditional bipartite approach.

Copyright is held by the International World Wide Web Conference Committee (IW3C2). IW3C2 reserves the right to provide a hyperlink to the author's site if the Material is used in electronic media.
WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2744-2/14/04.
http://dx.doi.org/10.1145/2567948.2578832.

Social machines take many forms, and have some overlap with domains such as Social machines can be seen as a polyarchy, where the infrastructure of general social machines is used to build more specific machines for particular purposes [24]. An example of this can be seen in "`#icanhazpdf`"; this is a simple social machine built on Twitter, concerned with the liberation of the scientific literature. Using the hashtag `#icanhazpdf`, participants have formed an ad-hoc community [7], where requests are made for PDFs of scientific papers. The coordination here is carried out entirely by humans—partly due to the subversive and legally questionable nature of the activity—but despite the noise in the channel, it has significant usage[14].

In general, any social machine will embody some kind of *interaction model*: a specification of who can do what, when, and what the effects should be. These are often: i) *informal*, as with `#icanhazpdf`; the interaction structures arise over time through community engagement. This can lead to tensions, as newcomers need to entrain themselves with a complex system[12]; ii) *opaque*, since they are the net effect of many pieces of code interacting together; iii) *non-user editable* as generally users within the system are given very limited opportunities to customise the way that interactions are carried out.

There is generally a permeable boundary between social and technical programming. For example, hashtags were not built in to twitter as a coordination mechanism when it was launched; rather, they were proposed by a user, and functioned as a social structure, a way to contextualise and group tweets together[16], and it was only after widespread community adoption that Twitter created computational architecture to support their use.

Electronic institutions [10] offer an alternative approach. Interaction models are explicitly specified, with roles, messages, actions and side effects, and actors can select an institution with which they desire to engage; by treating protocols as first-class objects [17] they can be shared, inspected, modified, executed and composed. This gives participants in an interaction far more power to understand and control their interactions. However, these institutions are rigorously formal, making their adaptation to human behaviour and adoption by large communities problematic.

The approach outlined in this paper binds electronic institutions onto existing social networks, providing a way to create new social machines which utilise existing socio-technical infrastructure to ease adoption. All code is available online[1].

[1] `https://bitbucket.org/mo_seph/scalsc-fuzzy`

2. SHADOW INSTITUTIONS AND AGENTS

The central theme of this paper is the use of shadow institutions and shadow agents[2] to support natural human interaction on the web. Building on the idea of coordination artifacts to facilitate interaction among computational agents [20], and drawing on ideas from the Ambient Intelligence community, [8] introduced the idea of a digital *mirror world*, where computational artifacts create *digital shadows* of real-world objects. This mechanism projects real-world state into the digital domain, allowing interaction between physical and computational entities and objects. Our notion of *shadow agents* is broadly similar: they are the digital shadows of people, projecting human interaction into the domain of formal systems. Shadow agents act out workflows on behalf of their human counterparts, interpreting their actions and making accessible the fruits computational processes: coordination, calculation, inference, persistence and so on.

Shadow institutions are designed around the idea that a limiting force on the adoption of electronic institutions (and computational coordination in general) is the cost to humans of understanding and entraining themselves with complex structures, and the brittleness with which they operate: the need for complete compliance.

If we take a set of agents $a_i \in A$ who want to carry out a complex task T, then the cost[3] might be given by the sum of the costs to all the agents involved:

$$C_{solo}(T) = \sum_{a_i \in A} C(t_{a_i})$$

If the agents engaged in an institution, the cost then becomes the cost to join the institution plus the cost of doing the task with the support of that institution:

$$C_{inst}(T) = \sum_{a_i \in A} C(J_{a_i,inst}) + C_{inst}(t_{a_i})$$

In many cases, the cost of joining the institution may outweigh the reduction in cost for its support—particularly for one-off tasks.

The motivation behind shadow institutions is to reduce the cost of joining (ideally to zero), while still reducing the cost of carrying out the task. An *ideal shadow institution* would hence have:

$$C(J_{a_i,isi}) = 0, \quad C_{isi}(t_{a_i}) \leq C(t_{a_i}) \quad \forall t, a_i$$

The main route towards achieving this reduction in joining cost is an inversion of control; rather than passively waiting for participants to learn how an institution works to the point where they can take advantage of it, a shadow institution actively looks for patterns of interaction which it understands, and attempts to support them. Some implications of this are:

- Participants may not know that a shadow institution exists until it does something for them; a truly zero-cost institution could be completely invisible until it collates, computes and comes back with a recommendation, a summary or some other useful piece of information.
- Existing infrastructure where people are *already* interacting can, and indeed must, be re-used or parasitized upon. There is no necessity to convince people to use a new platform, as the institution joins in with what they are already doing.
- At the same time, as the institution sits outside the social network infrastructure, it can enable cross-network interactions by scanning multiple networks simultaneously.
- Since participants have not had to go out of their way to invoke or join an institution, failure is more of an option. Any system exposed to the vagaries of human behaviour will break down at some point; in traditional institutions this failure must be repaired or the institution stops. With shadow institutions, failure of the institution is more likely to be a benefit which is not accrued, rather than a loss of effort or a waste of people's time.

[2]We use shadow here not to signify the crepuscular watchers of three letter agencies, but rather to relate to the projection of an object into another space—Plato's cave wall—or shadowing, where one person follows and observes another to improve understanding of how they carry out tasks.

[3]Cost could be simply effort, or include money, time, safety etc.

3. SYSTEM DESCRIPTION

3.1 Motivating Examples

In order to develop the framework, we have created a number of example interactions, which are available online[4]. Here we provide short descriptions of the motivations behind two examples, and the machinery which makes it possible. Other interactions which are not discussed here include collaborative hypothesising based on personal data stores and adding semantic markup to scientific papers.

3.1.1 Meal Coordination

The first example we chose is a pure coordination task, with a slight twist: at meetings and conferences, it is common to try to organise group meals, but the membership of the group is not known; for example, extra people have joined the meeting without being officially added to the mailing list. In this case, Twitter's use of hashtags to form ad-hoc communities is very useful: the meeting can post a hashtag which anyone interested can follow for information. However, actually processing a twitter stream to find out who is interested and what the consensus on where and when to meet is hard work, and it is useful to have some computational support. Figure 1 gives a step by step walkthrough of organising a group meal. A person decides to initiate a meal organising protocol, and messages the central bot, giving it a hashtag to denote the ad-hoc community—in this case `#sociam-dinner`, to organise meals for the SociaM project. The bot responds with a message which can be retweeted, tagged both with the community hashtag—so that it is visible to the community—and a tag identifying this particular interaction, so that the bots can identify the relevant messages. The system then collates responses, keeping track of who would like to join in, and votes for where and when to meet. Ideally, the system will be flexible enough, and the structure of the messages clear enough, that participants do not need to alter their natural behaviour. The initiator of

[4]Interaction models can be found at: `http://bit.ly/1ccXgSv`, and the bindings files can be found at: `http://bit.ly/1awRD5o`

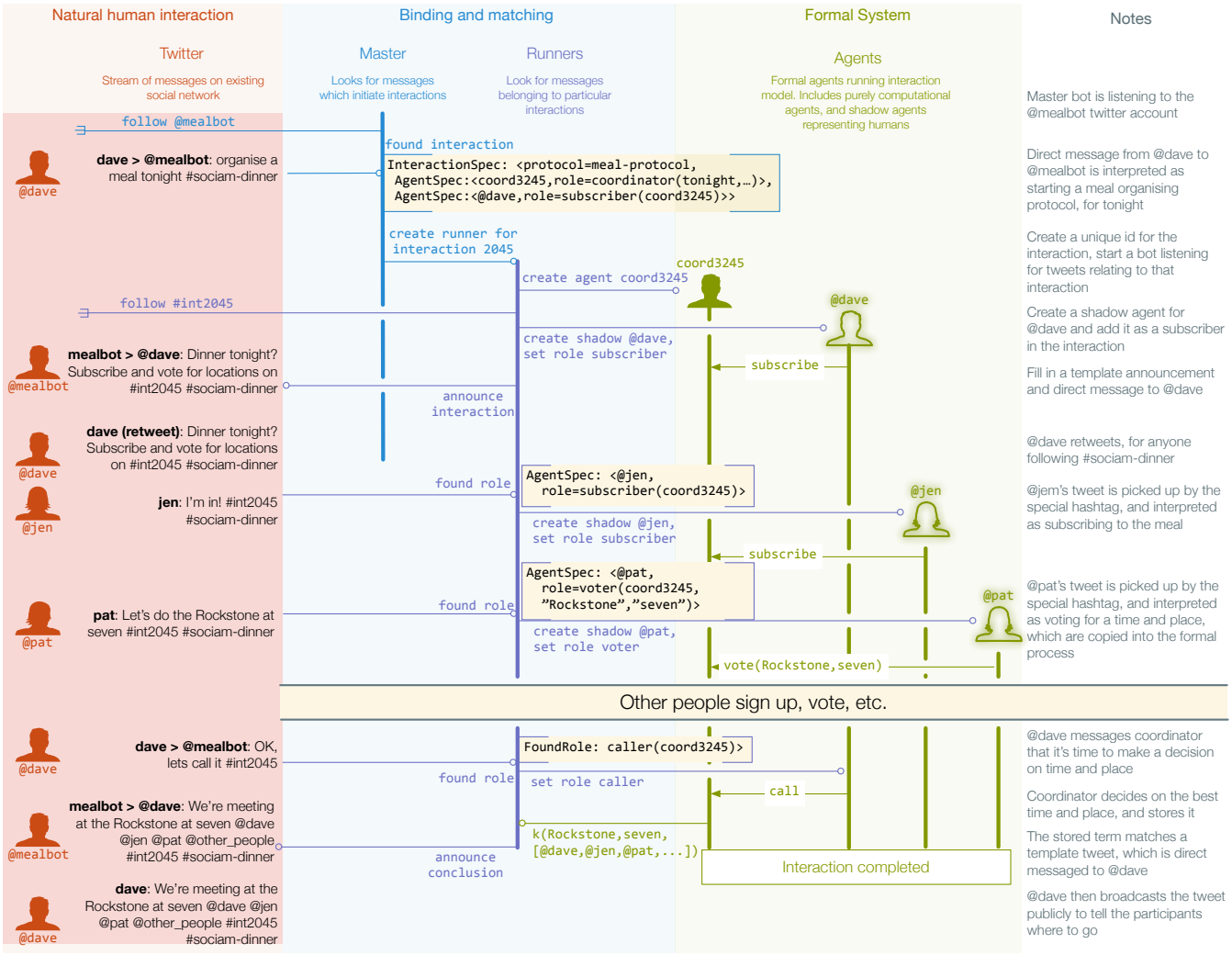

Figure 1: Diagram illustrating all components of an LSCitter interaction: interpretation of a natural interaction stream, binding to formal roles, enaction of interaction protocols and interpretation back into natural language.

the interaction can later message the bot to terminate the interaction, at which point it will notify anyone who has subscribed of the most popular time and place.

3.1.2 Taxi Sharing

In Britain, social norms prevent people in queues from talking to each other; at Edinburgh airport, there is often a large queue for taxis, and in a small city it is very likely that many people are going to nearby locations. Traditional approaches to this kind of problem often involve developing some kind of ride-share system, with trust and reputation metrics etc.; in this case any kind of heavyweight solution would be inappropriate for a situation where simply walking down the line shouting "Is anyone going to Portobello" would suffice. In our model, a taxi-sharing bot can be started for any location, and will run on a given hashtag, e.g. `#edinburghairporttaxis`. If anyone tweets to this bot that they are going to a rough geographic location in Edinburgh, it will search its memory for others going somewhere nearby in the last few minutes (using some geographic entity resolution mechanism to decide what is "nearby"). If a match is found, it will tweet to both of them with a meeting point, so they can then meet and share a taxi.

This exemplifies the benefits of shadow institutions:

- failure is an option, as if the system doesn't work, no-one is worse of than they were before. However, if it does work, people can save money on taxi fares, and queues are reduced.
- Existing networks are used, both Twitter and the taxi infrastructure, and their normal operation is augmented with computational intelligence.
- There is no cost to joining, other than learning that a magic hashtag exists, which could be easily done through e.g. stickers or posters near the taxi rank.

3.2 LSC - the Lightweight Social Calculus

In this paper we use LSC to specify interaction models, although the exact formalism is not important. LSC is an ex-

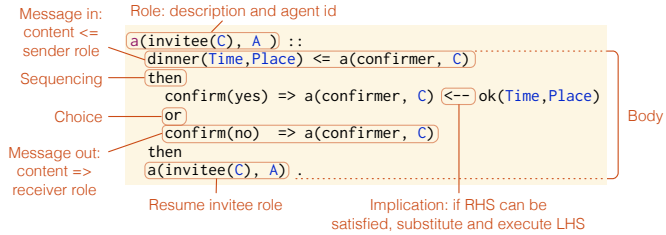

Figure 2: Example LSC clause from the meal organisation interaction model (slightly modified for clarity). An agent playing the role of `invitee` will wait for a message from a `confirmer` specifying the time and place for dinner; the values in the message for `Time` and `Place` are substituted in, and the agent then decides if it `will_attend`, and sends back the appropriate message. It then resumes the role of `invitee` in case of alternate suggestions.

tension of LCC [21], which has been used to represent interaction in many systems [22]. The LSC extension is designed to aid in interactions with humans, primarily by introducing two predicates: `e()` for information which should be elicited from a user, and `k()` to indicate information which becomes known through the course of the interaction. We provide an extremely brief description of LSC here to aid understanding this paper in isolation[5]. LSC is most easily understood as a combination of i) a Prolog-like language; ii) an interpreter which carries out Prolog style satisfaction and unification; and iii) a set of *rewrite rules* which define how interactions progress. The basic features of the language are i) *atoms*, which start with a lower-case letter or are quoted; ii) *variables*, which start with an upper case letter; iii) *predicates* with a name and arguments, e.g. `friends(dave,X)` and iv) *implication*: `likes(dave,X) <-- friends(dave,X)`. When an implication is encountered, the system attempts to find a substitution of variables for values which it can support, and then applies that substitution to the whole tree. So if `friends(dave,jim)` is known, the substitution `X=jim` is generated, and the implication would become `likes(dave,jim)`. This substitution mechanism is used by the engine to thread information through interactions.

An agent's state in an interaction can be described by: i) an LSC term defining the interaction so far; ii) a queue of messages coming in and going out; iii) any persistent state associated with the agent; and iv) the LSC protocol the agent is following. Whenever an agent receives a message or is asked to take on a role, the rewrite rules are run exhaustively over the current state to produce a new state, along with side effects such as sending messages and storing generated knowledge.

LSC protocols define agents and their roles, along with messages which should be sent or received, computation to be carried out, and information which should be used in the interaction. Actions can be carried out either in sequence, or as committed choice alternatives[6] (see Figure 2). Each role definition is strictly separate from any others; this means i) agents only need to know about their half of the interaction ii) any communication of information between agents (or between roles in a single agent) must be done explicitly.

3.3 Shadow institution definition

In our model, each shadow institution is based around a set of LSC protocols, and machinery to bind it to an interaction stream. Definitions for terms used the system can be found in Figure 3.4, and Figure 1 gives an illustration of a complete interaction.

The institution is instantiated by creating a master bot B_m, containing a collection of protocols, along with interaction matchers ($IM : U \nrightarrow IS$) which match messages U in the social network to produce interaction specifications IS. The master bot listens to a certain channel on the social network (often defined by a `@username` or `#hashtag` of interest) and attempts to match incoming messages with its interaction matchers. On a successful match, a runner bot B_r is created to run the interaction, using the generated interaction specification.

The runner bot has a mechanism for creating agents which can interpret LSC messages and protocols from an AS, threading in any necessary configuration and special behaviour. When the runner bot is started, it creates all the agents given in the IS, which are typically i) a coordinator agent to manage the interaction; and ii) a shadow agent to represent the person who initiated the interaction. The runner then subscribes to the social network, but using a channel that only picks up messages relevant to this particular interaction. It attempts to match incoming messages using its collection of RMs. A successful match will generate a FR, which can be used with the message to create a shadow agent for the sender (unless one exists already) and assign the found role to that agent.

3.4 Binding interaction to formal protocols

In order to interact with humans, the system must carry out translations i) from natural language into LSC ii) from LSC into natural language. Figure 3 is an example binding file for a meal organisation protocol. This is the only specification necessary in addition to the LSC protocols to create a working Twitter bot.

Natural language is brought into the system by matching messages received from the social network. Matching, for both master and runner bots can be carried out in any manner desirable, by providing a partial function from messages to interaction or role specifications respectively. Here, we use regular expressions due to their relative transparency, and easy specification. Figure 3 illustrates this mechanism: messages which match the pattern on the left produce the specification on the right, with any named groups in the regex substituted into variables with the same name in the data structure given. This provides a clean and transparent transfer of information from natural language into the formal system. Some transformations are possible (such as converting comma separated terms into lists, or parsing LSC terms if given), and more can be easily added.

As previously mentioned, LSC introduces a knowledge predicate `k()`. The intent of this is to a) describe what is known at the end of the interaction, and b) to allow

[5] A more user-oriented description is given at `http://bit.ly/1jEu3Ji`

[6] By committed choice we mean that once an agent starts down a particular branch of the tree, alternative branches are discarded, and no backtracking is allowed

```
val interactions=Seq(
  ".*a meal.* (?<When>tonight|tomorrow|sunday).*" ~~~>
  ("twitter-meal",
     ("Coord", ("Comm","twitter-meal","a(coordinator(When,[],[],Sender),Coord)")),
     ("Sender",("Comm","twitter-meal","a(subscriber(Coord),Sender)")))

val roles=Seq(
  "(?i).*(let'?s do|how about|i vote for)\\s+(?<Place>\\w+) at (?<Time>\\w+).*"
                                    ~> "a(voter(Time,Place,Coord),Sender)",
  "(?i).*(include|count me in|yes|i'm in|me too).*"
                                    ~> "a(subscriber(Coord),Sender)",
  "(?i).*(call|enough).*"           ~> "a(caller(Coord),Sender)" )

val announcers=Seq(
  "Want dinner tonight? Subscribe and vote for locations
    on #$Comm $InitialTags" > "$Initiator")

def coordinator_sat = store_sat(… Seq(
  "confirmed(T,P,People)"           ~> "We're meeting at $P at $T with $People
                                        #$Comm $InitialTags" > "$Initiator") )
val protocols = "twitter-meal"
```

Figure 3: Complete specification for the meal institution, taken (almost) verbatim from the Scala configuration file. `interactions` and `roles` map regexes onto interaction and agent specifications for the master and runner bots. Named groups are copied into LSC variables, e.g. in the first regex, the contents of (`?<Place...>`) are bound to the variable `Place` in the `a(voter ..)` role. `announcers` gives template messages to be filled in when an interaction begins, here as a direct message to the initiator of the interaction. `coordinator_sat` defines how the coordinator handles storing knowledge, and here LSC terms are mapped onto template messages. Finally, `protocols` defines which protocols should be loaded and made available to the institution

$$
\begin{aligned}
Message(U) &:= \langle sender, body, tags : Str*, dest : Str* \rangle \\
Clause &:= \langle Role, Body \rangle \\
Protocol(P) &:= Clause* \\
RoleSpec(RS) &:= \langle comm\_id, P, Role \rangle \\
AgentSpec(AS) &:= \langle id, P*, RoleSpec*, knol : Term* \rangle \\
Intr.Spec(IS) &:= \langle P, coords : AS*, S_1 : AS, knol : Term* \rangle \\
FoundRole &:= \langle Role, knol : Term* \rangle \\
Intr.Matcher(IM) &:= U \nrightarrow IS \\
RoleMatcher(RM) &:= U \nrightarrow RS \\
ShadowCreator &:= \langle U, FR \rangle \rightarrow AS
\end{aligned}
$$

Figure 4: Specifications used in matching messages to actions and roles in interaction models. Agent Specifications (AS) give enough information to create an agent, with an id, protocols to follow and necessary background knowledge. Role Specifications (RS) define an agent's role in a particular interaction, with a communication identifier and a protocol to use. Interaction Specifications (IS) can be used to initiate an interaction by defining the protocol to be used, a collection of agents and any necessary background knowledge. Found Roles (FR) give the minimal information used to ask a shadow agent to take on a role—a role clause, and any extra background knowledge necessary.

for this knowledge to be stored persistently. When interacting with humans, it can additionally be taken to indicate that the user should be made aware of the information, and within LSCitter, this implies that a message should be sent to inform them. When an LSC interpreter encounters a `k(Knowledge)` term, it will attempt to "store" it: the `coordinator_sat` definition in Figure 3 specifies that if the coordinator agent attempts to store a term matching `confirmed(T,P,A)`, it should also send the given template tweet with the the variables substituted into it—in this case, as a direct message to the person who initiated the interaction. An identical mechanism can be used for shadow agents to send messages to their respective humans. LSC also introduces the elicitation predicate `e()`, which is used when the user should be asked for input; similar techniques can be used for this, although for brevity no details are given here.

3.5 Related work

In *social computation*, humans are used to carry out tasks as an alternative to computers; frameworks exist to script this kind of interaction such as Jabberwocky/ManReduce [1], AutoMan [4] and CrowdDB [11]. BPEL4People similarly integrates humans into traditional formal workflows, but with a more general concept of *tasks* rather than pure computation. In the physical world, TaskRabbit[7] allows the crowdsourcing of physical labour, and efforts have been made to use this to coordinate teams of people on shared, real-world tasks [3]. Our main part of departure from this strand of research is a focus on the participants of interactions, rather than the commissioners—that is, how can we support people doing what they *desire* to do as opposed to being useful to some given purpose. More in-line with our vision are systems like WeDo[8] which helps people coordinate action over Twitter; we would hope to allow communities to build *similar* systems, but with a range of behaviour, by sharing and adapting the interaction models behind them.

The fitting of models to human interaction by extracting communicative actions has been carried out on a variety of interactions, including Twitter [26] (but also e.g. email [9] and music [18])

Additionally, the present work is inspired by work on programming the global brain [6] and the social computer [23]; the liberation of programming [13]; and work on collective intelligence [15].

4. DISCUSSION

This system is a work in progress, so no formal or empirical evaluation has yet been carried out. One aim is to be able to offer *coordination as a service*, where end users run interactions by uploading an interaction model and a bindings file to standard cloud infrastructure. This objective is largely met by the current system, which needs only the two files mentioned to run interactions. Early indications are that the presence of an interaction model makes the natural language processing task much easier; hence, we have been able to work entirely with regular expressions rather than invoking more complex and powerful NLP systems.

An architecture supporting generalised bindings between human online interaction and formal workflows provides many exciting opportunities.

One of the potential powers of the system is the ability to

[7] `http://www.taskrabbit.com`

[8] `http://wedo.csail.mit.edu/`

work across different social networks. In this paper, we have talked exclusively about Twitter, for purely practical reasons: it offers a large userbase, ad-hoc publics, single action messages and a machine friendly API. However, network membership is a personal issue, and varies widely. Some groups, especially younger users, will tend to use multiple networks in parallel to achieve tasks. By sitting outside of any given network, but reacting to information in any of them, a shadow institution can track these cross-network interactions, and bring all of the actions into a single interaction model, keeping track of responses and state and interfacing with participants through their preferred channel.

Secondly, if interactions can be mapped onto formal models, possibilities arise anywhere where computational intelligence can help. For example, interaction histories can be analysed—one of the effects of using LSC is that it generates complete interaction histories, since all operations involve rewriting the current state tree into an expanded version. These finished state trees can be computationally analysed, and information such as provenance can be computed and stored. At the same time external data can be integrated, be it from personal data stores or public repositories—the existing infrastructure supports working with Sparql based RDF stores, and an emerging personal data store [25] to achieve persistence and leverage existing knowledge.

Finally, the use of first-class protocols can democratise the creation of online communities. By creating institutions which parasitise the (largely privately owned) infrastructure of the social web, we offer communities the chance to design their own methods of interaction, without having to make the decision to walk away from existing tools and technologies, or create yet another social network. Interaction protocols can be seen as the DNA of social networks, and making them explicit allows the community a larger say in designing and understanding their behaviour. Similarly, we hope that—possibly coupled with the approaches to identifying patterns in online dialogue mentioned in [9, 26]—an approach in line with "Desire Lines" [2, 19] can be taken: users can interact naturally, according to their desires, and as the patterns of action become apparent, infrastructure can be installed or modified to support them.

We believe that our approach can be used to create a wide variety of social machines, which engage computational intelligence, but leverage existing relationships and communities, allowing the organic augmentation of natural interaction with formal architecture.

Acknowledgements

This work is supported by the EPSRC SociaM project under grant EP/J017728/1.

5. REFERENCES

[1] S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar. The Jabberwocky programming environment for structured social computing. *Proc. UIST 11*, 2011.

[2] C. Alexander. *The Oregon Experiment*, volume 3. Oxford University Press, 1975.

[3] Andrés Monroy-Hernández. Can crowds fill the void left by defunct newspapers? `http://socialmediacollective.org/2013/11/12/can`, 2013.

[4] D. W. Barowy, C. Curtsinger, E. D. Berger, and A. McGregor. AUTOMAN : A Platform for Integrating Human-Based and Digital Computation. In *OOPSLA'12*, Tucson, 2012.

[5] T. Berners-Lee, M. Fischetti, and M. F. By-Dertouzos. *Weaving the Web*. 2000.

[6] A. Bernstein, M. Klein, and T. W. Malone. Programming the global brain. *Comm. ACM*, 55(5):41, 2012.

[7] A. Bruns and J. E. Burgess. The use of Twitter hashtags in the formation of ad hoc publics. In *6th ECPR General Conference*, Reykjavik, Iceland, 2011.

[8] C. Castelfranchi and M. Piunti. AmI Systems as Agent-Based Mirror Worlds: Bridging Humans and Agents through Stigmergy. In T. Bosse, editor, *Agents and Ambient Intelligence*. Ios Press, 2012.

[9] W. Cohen, V. Carvalho, and T. Mitchell. Learning to Classify Email into"Speech Acts". In *Proc. EMNLP*, 2004.

[10] M. D'Inverno, M. Luck, P. Noriega, J. a. Rodriguez-Aguilar, and C. Sierra. Communicating open systems. *Artificial Intelligence*, 186:38–94, 2012.

[11] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: answering queries with crowdsourcing. In *Proc. ACM SIGMOD 2011*, pages 61–72. ACM, 2011.

[12] A. Halfaker, A. Kittur, and J. Riedl. Don't bite the newbies. In *Proc. WikiSym '11*. ACM Press, 2011.

[13] D. Harel. Can Programming Be Liberated, Period? *Computer*, 41(1):28–37, 2008.

[14] J. Liu. Interactions: The Numbers Behind #ICanHazPDF. http://www.altmetric.com/blog/interactions-the-numbers-behind-icanhazpdf/.

[15] T. W. Malone, R. Laubacher, and C. Dellarocas. The Collective Intelligence Genome. *MIT Sloan Management Review*, 51(3):21–31, 2010.

[16] C. Messina. Groups for Twitter. `http://factoryjoe.com/blog/2007/08/25/groups-for-twitter-or-a-proposal-for-twitter-tag-channels/`.

[17] T. Miller and J. McGinnis. Amongst first-class protocols. In *Engineering Societies in the Agents World VIII*, pages 208—-223. 2008.

[18] D. Murray-Rust and A. Smaill. Towards a model of musical interaction and communication. *Artificial Intelligence*, 175(9-10):1697–1721, 2011.

[19] C. Myhill. Commercial success by looking for desire lines. In *Computer Human Interaction*, pages 293–304. Springer, 2004.

[20] A. Omicini, A. Ricci, and M. Viroli. Agens Faber: Toward a Theory of Artefacts for MAS. *Electronic Notes in Theoretical Computer Science*, 150(3):21–36, May 2006.

[21] D. Robertson. A Lightweight Coordination Calculus for Agent Systems. In *DALT 2004, LNAI 3476*, pages 183–197. Springer, 2005.

[22] D. Robertson. Lightweight coordination calculus for agent systems: retrospective and prospective. In *DALT 2011, LNAI 7169*, pages 84–89. Springer, 2012.

[23] D. Robertson and F. Giunchiglia. Programming the social computer. *Phil. Trans. Roy. Soc. A*, 371(1987), 2013.

[24] N. Shadbolt, D. Smith, E. Simperl, M. Van Kleek, Y. Yang, and W. Hall. Towards a classification framework for social machines. In *SOCM2013*, Rio de Janeiro, Brazil, 2013.

[25] M. van Kleek, D. Smith, N. Shadbolt, and M. Schraefel. A decentralized architecture for consolidating personal information ecosystems: The WebBox. In *CSCW PIM 2012*, 2012.

[26] R. Zhang, W. Li, D. Gao, and Y. Ouyang. Automatic Twitter Topic Summarization With Speech Acts. *IEEE Audio, Speech, Language Process.*, 21(3):649–658, 2013.