Towards Executable Representations of Social Machines

Dave Murray-Rust[1], Alan Davoust[1],
Petros Papapanagiotou[1], Areti Manataki[1], Max Van Kleek[2], Nigel Shadbolt[2], and Dave Robertson[1]

[1] University of Edinburgh, 10 Crichton Street, Edinburgh EH8 9AB, United Kingdom
{d.murray-rust, adavoust, pe.p, a.manataki, dr}@inf.ed.ac.uk
[2] University of Oxford, Parks Rd, Oxford OX1 3QD
{max.van.kleek, nigel.shadbolt}@cs.ox.ac.uk

Abstract. Human interaction is increasingly mediated through technological systems, resulting in the emergence of a new class of socio-technical systems, often called *Social Machines*. However, many systems are designed and managed in a centralised way, limiting the participants' autonomy and ability to shape the systems they are part of.
In this paper we are concerned with creating a graphical formalism that allows novice users to simply draw the patterns of interaction that they desire, and have computational infrastructure assemble around the diagram. Our work includes a series of participatory design workshops, that help to understand the levels and types of abstraction that the general public are comfortable with when designing socio-technical systems. These design studies lead to a novel formalism that allows us to compose rich interaction protocols into functioning, executable architecture. We demonstrate this by translating one of the designs produced by workshop participants into an a running agent institution using the Lightweight Social Calculus (LSC).

Keywords: Social Machines, diagrammatic interface, rapid assembly, prototyping

1 Introduction

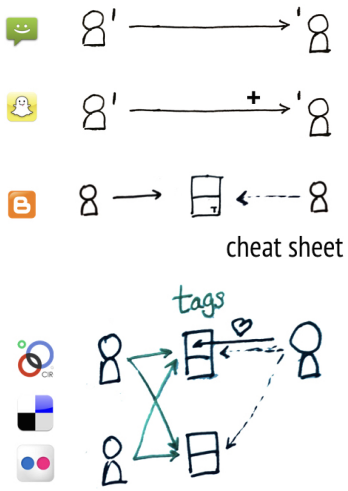
Ubiquitous computation and digital communication systems have produced new forms of socio-technical systems, vast networks where people achieve coordinated action at scale. Examples include Wikipedia, Twitter, Ushahidi and so on. Characterising such systems requires thinking beyond their software infrastructure: one unified lens for viewing them is Berners-Lee's concept of *social machines* [2, 7], that describes systems in which humans and computation play complementary roles.

In this paper, we explore the design and construction of social machines through a diagrammatic language. Our main design goals for the language are (i) to be accessible to non-specialists, enabling them to craft their own social machines themselves, and (ii) to be sufficiently formal to be able to turn into executable code, so that designers can create running systems directly from diagrams. The language we propose has been shaped by a series of design workshops where non-expert participants designed Social Machines with limited guidance, using some suggested constructs based on the notions of roles and protocols – and their own free-form diagramming skills.

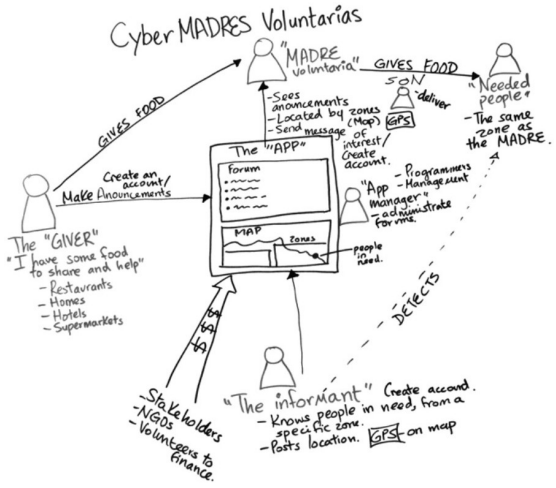
From the outcomes of these workshops we designed a diagrammatic language and methodology to design Social Machines as Electronic Institutions, formalized with the Lightweight Social Calculus (LSC [5]) to provide executable infrastructures. We present an intuitive design process for users to construct Social Machine models using this formalism, and demonstrate the execution of a model using a generic LSC engine.

2 Design method

Our language has been refined based on four participatory design workshops, mostly involving non-experts (who were new to the concept of social machines), where participants collaboratively designed their own



(a) Some of the graphical elements from the first Sociograms workshop



(b) The *Cybermadres* social machine, sketched by workshop participants. This Social Machine would support the activities of volunteers in Mexico, who collect excess food from restaurants and distribute it to people in need.

social machines from scratch, using paper and markers. At the beginning of each session, participants were provided handouts with example diagrammatic primitives that we designed (Fig. 1a), that they could (optionally) use to help them sketch their social machines.

Figure 1b shows one ad-hoc diagram created by non-specialists, showing roles, coordination, interactions, implementation hints and social aspects of the system concisely and comprehensibly. From such diagrams we extracted the following key elements: *People*, playing a range of roles: restaurants that provide waste food, informants who spot people in need, *madres* that mediate the interactions and so on; *Infrastructure*, whether physical or computational, that coordinates the activity of humans around their purpose; and finally *connecting arrows*, representing the transmission of messages or physical objects ('gives food'); more general geo-spatial interactions ('detects'); and bundles of possible operations on computational systems ('make announcements') that implicitly include access control, posting, retracting and updating information etc.

At the conclusion of each session, each group was asked to present their social machine; photos were captured of diagrams made, and verbal descriptions were recorded, transcribed, and archived. We then analysed the graphical vocabulary used in the diagrams, along with descriptions, to identify where graphical primitives were appropriated, reused, and extended. This enabled us to identify robust components, eliminate and refine components that were not used or misunderstood. For the final workshop, we presented the participants with a paper based version of the diagram tool discussed in this paper, prompting them to investigate and define the interactions between actors.

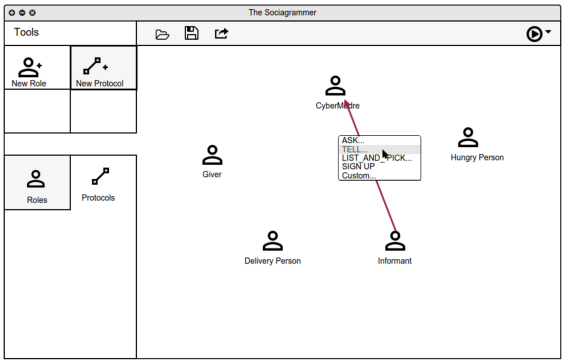
3 Diagram Language

Our diagrammatic language contains the following elements, with a full demonstration given in Figure 2:

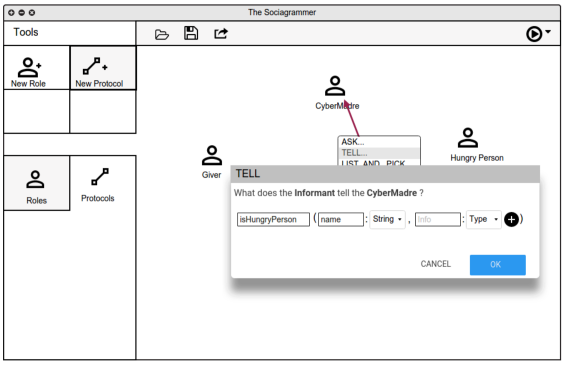
Nodes represent actors in the system, whether computational or human.

Edges define the interactions between these actors, by specifying interaction protocols (Fig 2a).

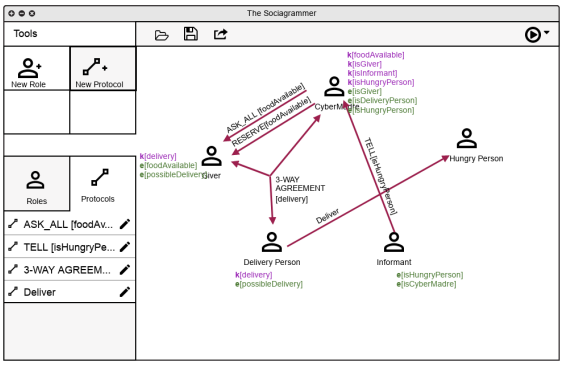
Protocols are specified as generic activities (e.g. ASK, ESCROW), and specialised to define the kinds of data that flows through them. For example, a simple communication protocol might carry a specification



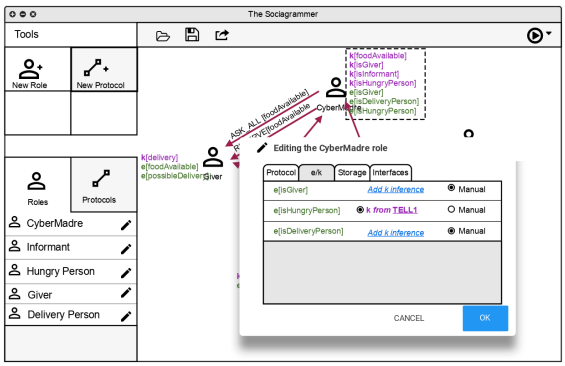
(a) Definition of actors in the system, with protocols describing interactions between them



(b) Specialisations of the protocols to carry appropriate data for the given interaction



(c) A complete system, with multi-way interactions and composed capabilities



(d) Eliciting connections between the inputs and outputs of different actors

Fig. 2: Example operation of the Sociogrammer tool, from initial specification of actors through to linking predicates

of the kinds of messages to be transferred, which then implies that the roles involved in the interaction are capable of providing or processing that type of information.

The diagram shown in Figure 2c details the key interactions of a particular social machine infrastructure—in this case, a system that supports the *CyberMadres* example in Figure 1b. Our interface mock-ups sketch the design process and the extra information needed to create a working system.

3.1 From Diagrams to Executable Systems

Diagrams created in this manner can be automatically transformed into the Lightweight Social Calculus (LSC) [4, 6], a high level protocol language. An LSC protocol consists of the roles that each participating agent may play, by executing the part of the protocol that corresponds to their role locally. Each role may involve message passing and computation, tied together with conditions (`If` ... `then`) and temporal sequencing (`Then` and `Or`). Computation involves input predicates `e`(), and output predicates `k`(). Inputs *elicit* information, typically through a local knowledge base, or by "asking a human" via an interface, at

which point human decision making can enter the computational system. Output predicates indicate that the agent now *knows* something, which implies that the agent be capable of processing the information, e.g. storing it in a local knowledge base or otherwise changing the state of the world in a representative way.

The design process (Fig. 2d) involves connecting inputs from one protocol to outputs of another, or else flagging particular inputs to be fulfilled by processes outside the scope of the system.

4 Discussion and Conclusions

There are two key questions behind this work: i) Can we develop a diagrammatic language for designing social machines, accessible to non-experts? and ii) Can we design this language in such a way that it produces executable infrastructures?

A key design challenge is to find the correct, composable units to build these diagrams from. Here we have used interaction protocols, as a way to cover complex patterns of activity that unfold in time using simple, human identifiers, e.g. "arrange a meeting". Our workshops have shown that participants see real possibilities in designing such systems, and their ad-hoc diagrams show some of the key concepts needed to describe interaction protocols. Participants to produce meaningful and plausible social machine designs in a 2-3 hour workshop, from a standing start. Designs included social interventions that help the homeless, shared diaries for nomads, crowdsourced traffic reports and interpersonal archives.

Relative to the second question, we have shown a prototype interface, where a version of the simple, at-hand iconography can be used to specify enough detail to create executable infrastructures. This demonstrates the potential for a translation from simple readable diagrams into working systems.

The question remains as to what a meaningful set of interaction primitives might be, simple yet expressive enough to describe most social machines. Our initial analysis has brought up a range of useful constructs, which allowed us to fulfil the designs created by workshop participants.

This leaves us with a form of extremely concise Model Driven Development [1, 3] that supports a democratic, participatory approach that allows a wide range of people to design a profusion of small social machines, adapted to their particular communities of practice.

This proof of concept indicates that there is a level of representational complexity that allows people to make their intentions clear about complex systems that would otherwise be beyond their ability to design.

References

1. Agüero, J., Rebollo, M., Carrascosa, C., Julián, V.: MDD for virtual organization design. Trends in Practical Applications of Agents and Multiagent Systems pp. 9–17 (2010)
2. Berners-Lee, T., Fischetti, M., Foreword By-Dertouzos, M.L.: Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor. HarperInformation (2000)
3. France, R., Rumpe, B.: Model-driven development of complex software: A research roadmap. In: 2007 Future of Software Engineering. pp. 37–54. IEEE Computer Society (2007)
4. Murray-Rust, D., Papapanagiotou, P., Robertson, D.: Softening electronic institutions to support natural interaction. Human Computation 2(2) (2015)
5. Murray-Rust, D., Robertson, D.: LSCitter: building social machines by augmenting existing social networks with interaction models. In: Chung, C., Broder, A.Z., Shim, K., Suel, T. (eds.) 23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume. pp. 875–880. ACM (2014), http://doi.acm.org/10.1145/2567948.2578832
6. Robertson, D.: A lightweight coordination calculus for agent systems. In: Proceedings of the Second International Conference on Declarative Agent Languages and Technologies. pp. 183–197. DALT'04, Springer-Verlag, Berlin, Heidelberg (2005), http://dx.doi.org/10.1007/11493402_11
7. Shadbolt, N., Van Kleek, M., Binns, R.: The rise of social machines: The development of a human/digital ecosystem. IEEE Consumer Electronics Magazine 5(2), 106–111 (2016)