LS^{2}C – A Platform to Design, Implement and Execute Social Computations

Flavio S. Correa da Silva[1], David S. Robertson[2] and Wamberto W. Vasconcelos[3]
[1]*Dept. of Computer Science, University of Sao Paulo, Sao Paulo, Brazil*
[2]*School of Informatics, University of Edinburgh, Edinburgh, U.K.*
[3]*Dept. Computing Science, University of Aberdeen, Aberdeen, U.K.*

Keywords: Interaction Models, Social Computation Models, Social Interaction Protocols.

Abstract: Social computers have been characterised as goal oriented complex systems comprised of humans as well as computational devices. Such systems can be found *in natura* in a variety of scenarios, as well as designed to tackle specific issues of social and economic relevance. In the present article we introduce the *Lightweight Situated Social Calculus (LS^{2}C)* as a language to design executable specifications of interaction protocols for social computations. Additionally, we describe a platform to process these specifications, giving them a computational realisation. We argue that *LS^{2}C* can be used to design, implement and execute social computations.

1 INTRODUCTION

Social computers have been characterised as complex systems that harness the innate problem solving, action and information gathering powers of humans and the environments in which they live to tackle large scale social and economic problems (Giunchiglia and Robertson, 2010):

- The "hardware" of a social computer is supplied by humans (taken as individuals as well as collectively in the form of human-powered institutions) and the environment where these humans live, including all relevant artifacts which can be natural or man-made, as well as computational devices.
- The "software" of a social computer is comprised of human capabilities, organisational and social rules and norms, social conventions, as well as computer software.
- The "algorithms" of social computation are defined by socially accepted goals and corresponding actions which can be taken to achieve local as well as global goals.
- Finally, the "processing" of algorithms in social computers are collective, decentralised, goal-oriented actions whose emergent results can be iteratively evaluated and steered towards active goals.

Evidently, one cannot program a social computer the way conventional computational devices are programmed. Social computers are evolving social systems, whose components (i.e. their "hardware", "software", "algorithms" and "processing") are dynamically and evolutionarily designed together with their goals and available resources. The analysis and design of social computers require novel methodological practices, blending existing techniques and experiences from applied social sciences and computational sciences (Correa da Silva et al., 2013).

In order to design, implement and continuously monitor and steer the behaviour of social computers, specialised languages are required to build specifications, and corresponding computational platforms are required to support, manage and provide a computational realisation of social computations. An essential aspect to be represented in such languages is interaction between components of social computers, so that the internal behaviour of these components can be abstracted and the resulting systems can be analysed as a whole. Additionally, since these languages should be used to communicate specifications as well as processing results to participants in social computers (i.e. humans who behave as components in social computers), they should be concise and simple to understand. Finally, in order to build social computers whose behaviour can be verified with respect to desired requirements and attributes, these languages should have a formal underpinning and the corresponding specifications and processing results should be formally verifiable.

In the present article we present the ongoing development of a language and a platform for social computers. The proposed language and companion computational platform – coined the *Lightweight Situated Social Calculus (LS^{2}C)* is a fusion of two previously existing languages, respectively the *Lightweight Social Calculus (LSC)* and the *JamSession platform*.

In section 2 we present some related work and the preliminary concepts that have guided the development of the *LS^{2}C* language and platform. In section 3 we introduce in detail the *LS^{2}C* language. In section 4 we briefly describe the platform in which this language shall be used. In section 5 we illustrate how this platform can be used in practice. Finally, in section 6 we present some conclusions and proposed future work.

2 RELATED WORK AND PRELIMINARY CONCEPTS

The *LS^{2}C* platform is a fusion of the *Lightweight Social Calculus (LSC)* and the *JamSession platform*. *LSC*, in turn, is an extension of the *Lightweight Coordination Calculus (LCC)*. In the following paragraphs we briefly introduce these languages and platforms.

The *LCC* is an executable specification language grounded on the notions of process algebras and initially proposed for the specification and processing of interaction models for distributed software components (Robertson, 2004). It has been extended in a variety of ways, e.g. for contextual reasoning about distributed software systems (Sindhu et al., 2006), for the specification and execution of choreographies for web services (Bai et al., 2012) and, more recently, for the specification of social computers, under the name of *Lightweight Social Calculus – LSC* (Murray-Rust and Robertson, 2014). It has also been successfully implemented using the logic programming language *Prolog*, the object oriented programming language *Java* and the object-functional programming language *Scala*.

LCC and its variations – particularly *LSC* – fulfill most of the requirements to be a language for the specification, implementation and processing of social computations. *LSC* is a compact formal language that can be used to specify and to mediate ongoing social interaction protocols. The syntax of *LSC* (as well as all other variations of *LCC*), however, can lead to lengthy specifications which can be difficult for human reading and understanding. Moreover, the extension of *LCC* to manage contexts (coined *Ambient LCC* (Sindhu et al., 2006)) departs from the lightweight approach and becomes more complex than the other variations of *LCC*, resulting in a not so concise and effective platform for the specification and execution of interaction protocols by human system designers.

The explicit management of contexts can be a powerful technique to help in the analysis and design of social interactions, given that many of these interactions are context-dependent (e.g. business negotiations must occur in adequately equipped meeting rooms, to ensure privacy and the availability of required communication resources; healthcare must occur in hospitals and clinics, to ensure the availability of required specialised equipment and personnel; bank transactions must occur over the appropriate counters; the automated interactions among communicating portable devices in an Internet of Things scenario must be context sensitive to ensure privacy and reliability of interactions; and so on). Therefore, a useful feature in a language for social computers is the explicit representation and management of contexts, which can be abstracted as *locations* in which certain interactions are allowed to occur.

The *JamSession platform* is a language developed for purposes similar to those of *LSC*. It was initially conceived as an executable specification language to manage the interactions between human controlled and synthetic characters in *Second Life*-style virtual worlds and multiplayer computer games, and later employed to mediate business interactions between organisations in cross-organisational workflows (Correa da Silva, 2011; Correa da Silva et al., 2012). A simplified prototype of *JamSession* has been implemented in *Prolog*, and a cloud-based prototype of *JamSession* has been implemented using the functional language *F$\sharp$*, based on which sample demos of applications have been developed (David, 2012). The fundamental concept in *JamSession* is the notion of *situated interaction protocols*, which determine how and where agents can interact with each other and with the environment. The semantics of situated interaction protocols can be formally characterised in terms of Nested Petri Nets, which are an extension of coloured Petri nets to handle recursion (Fernandez Venero and Correa da Silva, 2013a). Nested Petri Nets, in turn, can be translated into the specification language Promela and verified using the model checker SPIN with respecto to properties of their operational behaviour (such as liveness and termination) (Fernandez Venero and Correa da Silva, 2013b).

There have been initiatives by other authors to analyse social interactions based on formal languages capable of capturing the dynamics of interactions, in many cases grounded on the notions of dynamic modal logics, preference logics and public announcement logics (Christoff and Hansen, 2013; Hansen, 2014; Seligman et al., 2011; Zhen and Seligman, 2011). Our work distinguishes from these initiatives in two relevant senses:

1. We focus on systems *design* as well as analysis, whereas those initiatives focus primarily on analysis of existing social networks grounded on formal theories.
2. Since we are interested in the design of systems for goal-oriented social interactions, we have taken into account scalability and computational performance issues, as well as interaction design issues. Previously existing initiatives have mostly focused on theoretical issues, accounting for computational and system level performance as secondary issues. Scalability in *LS*^{2}*C* shall be ensured by the appropriate use of asynchronous state management based on *Linda*-style tuple spaces (Gelernter, 1985), following the implementation practices used in *JamSession*.

3 THE *LS*^{2}*C* LANGUAGE

The *Lightweight Situated Social Calculus (LS*^{2}*C)* is based on the notion of *situated social interactions*. A situated social interaction is comprised of actions which are permitted to occur if performed by specific agents at specific locations, together with messages exchanged among agents in order to enable and trigger further actions, and with migrations of agents across locations. In *LS*^{2}*C*, locations are an abstraction used to represent a variety of concepts, such as:

- Actual physical locations, e.g. the counter in a bank where financial transactions are permitted to occur.
- Contextual information, e.g. characterising the collective acceptance of interaction protocols by agents in a business transaction (buy, sell, legal procedures, and so on) so that it is common knowledge that the transaction can be carried out as long as all actions in all protocols – the "setting" for the transaction – are fulfilled.

Locations are represented as nodes in a directed graph, in which edges represent accessibility relations, characterising allowed transitions between locations or contexts. We denote the set of nodes in a graph of locations as $\mathcal{S} = \{s_1, ..., s_r\}$[1].

[1]We abuse notation and also refer to the graph of locations itself as $\mathcal{S}$.

Each location can host an unlimited number of agents. An agent is capable of:

- Moving between directly connected locations.
- Performing allowed actions while in specific locations.
- Reading, writing or deleting messages in locations.

We have a set of agents $\mathcal{A} = \{a_1, ..., a_m\}$ whose behaviour is constrained and determined by each role that they adopt, according to each location to which they move. Actions and message types are available only to agents bearing specific roles at specific locations. The positioning of agents in locations is the way to control the processing of algorithms in social computers represented using *LS*^{2}*C*.

Actions can be enabled by and influence or transform objects that can be found in the environment. We have a set of objects $\mathcal{B} = \{b_1, ..., b_n\}$, which are subject to the actions of agents. Objects can be physical objects as well as their digital counterparts.

We build mappings pointing to agents and objects, so that we can refer to them indirectly through built connections between them (such as *FatherOf(X)* to refer to an agent by naming another agent). For this reason, we also include in the language a set $\mathcal{F}$ of n-ary functions, $0 < n < \infty$, such that if $f \in \mathcal{F}$ has arity j, it can be used to build or point to an element of $\mathcal{A} \cup \mathcal{B}$ given j elements of $\mathcal{A} \cup \mathcal{B} \cup \mathcal{S}$. In other words, $f : (\mathcal{A} \cup \mathcal{B} \cup \mathcal{S})^j \mapsto \mathcal{A} \cup \mathcal{B}$.

In order to be able to build terms as in first order logics, we also include a countable set of variables $\mathcal{X} = \{x_1, ...\}$. Every formula built in this calculus is assumed to be existentially closed, i.e. free variables are implicitly bound to existential quantifiers.

We build relations involving agents and objects, representing information that can be known by agents and action statements. We have three sets of n-ary predicates to represent each of these relation types:

- $\mathcal{P}$: set of n-ary knowledge predicates, $0 \leq n < \infty$.
- $\mathcal{Q}$: set of n-ary action predicates, $0 \leq n < \infty$.
- $\mathcal{R}$: set of n-ary protocol names, $0 \leq n < \infty$.

Predicates are prefixed by modal operators as follows, in which $p \in \mathcal{P}, q \in \mathcal{Q}, r \in \mathcal{R}, s \in \mathcal{S}$ and $a \in \mathcal{A}$:

- $[k]_a^s p$ denotes a knowledge modality – agent a knows fact $p \in \mathcal{P}$ at location s.
- $[e]_a^s q$ denotes an engagement modality – agent a performs action $q \in \mathcal{Q}$ at location s.
- $[i]^s q$ denotes a location-specific computation modality – action $q \in \mathcal{Q}$ is processed at location s.

- $[i]^s r$ denotes a location-specific interaction protocol – protocol $r \in \mathcal{R}$ can be started from location s.

In order to avoid unnecessary complications in our proposed language, we allow modal operators to only prefix a single predicate, i.e. no nesting of modalities is allowed, nor it is allowed to have a modality prefixing arbitrary formulae.

Communicative actions are defined as follows, in which $p \in \mathcal{P}, a, a' \in \mathcal{A}$ and $s, s' \in \mathcal{S}$:

- *null*: a void message.
- $[e]^s_a write(p, a')$: agent a writes message in location s, which is then stored as predicate p known by agent a' in s, i.e. $[k]^s_{a'} p$. In other words, agent a tells p to a' in s.
- $[e]^s_a del(p, a')$: agent a deletes message which was previously stored in location s as predicate p known by a' in s, i.e. the piece of knowledge $[k]^s_{a'} p$ is retracted from location s.

In order to continue with the definition of *LS*^{2}*C*, we need to define two connectives:

- Non-commutative conjunction: given two existentially closed formulae φ and ψ, the conjunction $\varphi \sqcap \psi$ is evaluated as $\top$ if:
  1. φ is evaluated as $\top$ *AND*
  2. the variable bindings performed during the evaluation of φ are used to bind the values of variables in ψ, producing the instatiated formula $\hat{\psi}$ *AND*
  3. $\hat{\psi}$ is also evaluated as $\top$.

  Otherwise, the conjunction $\varphi \sqcap \psi$ is $\bot$.
- Non-commutative disjunction: given two existentially closed formulae φ and ψ, the disjunction $\varphi \sqcup \psi$ is evaluated as $\top$ if:
  1. φ is evaluated as $\top$, in which case ψ is never evaluated *OR*
  2. φ is evaluated as $\bot$, and ψ is evaluated as $\top$. In this case, the variable bindings performed during the evaluation of φ are not used to bind the values of variables in ψ.

  Otherwise, the disjunction $\varphi \sqcup \psi$ is $\bot$.

We define an atomic event *AE* as one of the following expressions, in which $p \in \mathcal{P}, q \in \mathcal{Q}, r \in \mathcal{R}, a \in \mathcal{A}$ and $s, s_i, s_j \in \mathcal{S}$:

- $[k]^s_a p$.
- $[e]^s_a q$.
- $[i]^s q$.
- $[i]^s r$.
- $[i]^{s_i \to s_j}_a mv$, in which the special predicate mv is used to state that agent a is being moved from location s_i to location s_j.
- a communicative action M.

We define an event E as a conjunction of atomic events, i.e. $E = \sqcap_i AE_i$.

Finally, we define an interaction protocol as a pair $\langle [i]^s r, \sqcup_i E_i \rangle$, in which $[i]^s r$ is a location-specific interaction protocol and $\sqcup_i E_i$ is a non-commutative disjunction of events.

The interaction protocol $\langle [i]^s r, \sqcup_i E_i \rangle$ is triggered by a formula that unifies with the left hand side expression $[i]^s r$. Variable bindings are applied to the right hand side expression $\sqcup_i E_i$, which is then computed. Each event E_i is an alternative course of actions that can be tested. If one of the events E_i returns $\top$, then the interaction protocol succeeds and the corresponding variable bindings are presented. If all alternatives in $\sqcup E_i$ return $\bot$, then the interaction protocol fails and variable bindings are discarded.

It should be observed that, since location-specific interaction protocol expressions $[i]^{s_i} r_i$ can occur as atomic events in the right hand side of interaction protocols, recursive interaction protocols are allowed in *LS*^{2}*C*.

LS^2*C* is a coordination language. Knowledge is encoded in the platform using communicative actions that update knowledge predicates $p \in \mathcal{P}$, and action predicates are expected to be evaluated by external actors, which can include human as well as computational agents.

4 THE *LS*^{2}*C* PLATFORM

We are working on a robust implementation for the *LS*^{2}*C* language, benefitting from existing implementations of *LCC* and of *JamSession*, that shall be freely deployed as the *LS*^{2}*C platform*. In this software platform, the graph of locations, the list of pairs $\langle s, a \rangle, s \in \mathcal{S}, a \in \mathcal{A}$ for each predicate indicating where and by whom it can be evaluated, and the state of each location are managed in a centralised cloud server.

Interaction protocols are stored in distributed hosts. The processing of these protocols may require human intervention, this way characterising the *LS*^{2}*C Platform* as a tool to support and manage social interactions. The physical location where interaction protocols can be found is stored in the centralised cloud server as an address catalog. This catalog can be rearranged locally according to private ranking criteria, defined by priority policies used in different sites which can be used to rank interaction protocols.

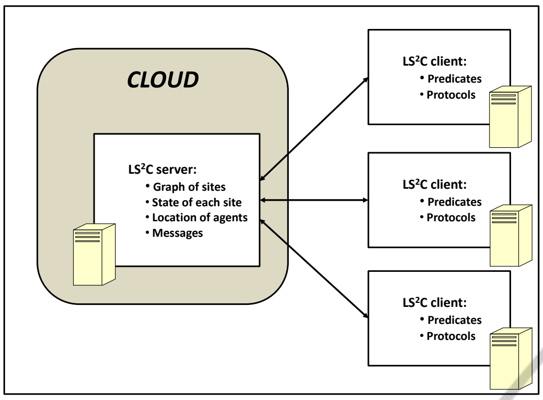

Figure 1: The architecture of the *LS*^{2}*C* Platform.

The locations of agents are also managed in the centralised server, characterising the notions of *virtual worlds* as featured in the *JamSession* literature (Correa da Silva, 2011) and *mirror worlds* as featured in the *LSC* literature (Murray-Rust and Robertson, 2014).

The definitions of predicates – including action predicates, which can capture the input-output expected behaviour of human actions – are stored in the distributed hosts. The locations of interaction protocols and their corresponding predicates are stored in the centralised server.

The architecture of the *LS*^{2}*C* Platform is depicted in Figure 1.

Protocols can be triggered concurrently and asynchronously by several users. As a consequence, the verification of properties related to distribution and concurrency is important to ensure an expected behaviour in a system whose interactions are specified using *LS*^{2}*C*. We are working on the characterisation of *LS*^{2}*C* protocols using Nested Petri Nets, based on our experience using the same formalism to characterise *JamSession* protocols. Nested Petri Nets can be used to formally verify properties such as fairness, liveness and termination. Given that Nested Petri Nets can also be translated as Promela programs to be verified using the model checker SPIN (Fernandez Venero and Correa da Silva, 2013b), we will be able to verify such properties also for *LS*^{2}*C* social interaction protocols.

Similarly to *LCC* and to what can be observed in business process modeling (Correa da Silva et al., 2012; Robertson, 2004), social interaction protocols can be considered at specification time and at run time. Specification time refers to the design of social computers, while run time can refer to the *a posteriori* analysis of the actual execution of social computations, in which e.g. specific protocols are used to enact concrete interactions. Such analysis can reveal social network properties involving interacting peers, such as centrality of a location, and cohesiveness and density of location-related interactions (Jackson, 2008), whose interpretation can be relevant to understand features of specific domains.

5 AN EXAMPLE – *LS*^{2}*C* FOR *S*^{3}

In this section we mention some potential applications for the *LS*^{2}*C* platform, and sketch how some interaction protocols can be encoded for one of these applications.

The *LS*^{2}*C* platform has been conceived to design and implement social computations in which situated interactions are most relevant. Social computers of this sort can be found in *urban computing* and in *intelligent city* environments, which are urban landscapes augmented with digital communication and processing devices and applications (Jiang et al., 2013; Komninos, 2006; Schaffers et al., 2011; Zheng et al., 2011).

Urban computing refers to requesting citizens to carry (most likely within their smartphones) software applications that track their activities, interact with them and provide information to service managers, so that the quality of service provisioning can be improved in issues such as traffic monitoring, public transportation and emergency relief.

Intelligent cities are urban settings which have been augmented with ubiquitous computing devices, in such way that existing services can become more effective and novel services can be offered to citizens, businesses and governments. A representative example of what can be achieved under the concept of intelligent cities is the structuring of effective business clusters supported by digital services. In the following paragraphs we detail this possibility.

An important well known factor for regional economic development is innovation. Innovative entrepreneurship is frequently associated with start-up companies, which in most cases are small companies which hold deep knowledge and skills over a narrow and specialised domain. One factor that has proven to be influential for the survival of these companies is their ability to cooperate with other companies, possibly forming or entering a network of cooperating organisations. Local and regional governments have taken notice of that and have created programmes to support and incentive the blooming of such networks, as well as studied how these networks should be structured in order to minimise the risk of failure of participating companies and maximise the economic efficiency of the networks (Feldman and Audretsch, 1999; Lazzarini et al., 2001; Mesquita and Lazzarini, 2007; Pedrozo and Pereira, 2006).

Business clusters are emergent agglomerations of companies which benefit from the proximity of each other to grow. Smart Specialisation Strategies (S^3) have been proposed recently as means for policy makers to build "smart clusters", in which the use of knowledge and resources is optimised and cooperation among companies is brought to be most effective (EU, 2012; EU, 2013). S^3 can be seen as an effort to design business clusters, instead of simply providing appropriate means for them to emerge.

Business relations are partially constrained by rationality rules (such as profit maximisation and risk minimisation). On the other hand, companies are human-powered and human controlled institutions, and therefore – especially small companies – are influenced by human decision making, which takes into account rules beyond those that can be captured by simplified models of pure rationality (e.g. brand fidelity, intuition driven trust relations, aesthetic considerations and cultural affinity). Company relations are diversified and include customer-supplier relations as well as cooperative relations involving similar companies (Lazzarini et al., 2001; Pedrozo and Pereira, 2006; Mesquita and Lazzarini, 2007). Hence, we suggest that business networks can be treated as social computers, and that the *LS*^{2}*C Platform* can be a useful tool to design, implement, run, monitor and iteratively refine Smart Specialisation Strategies.

In order to illustrate how this can be done, we show a simplified version of interaction rules that could be relevant in a customer-supplier relationship involving two companies. In this example, company **A** asks company **B** to provide a service that is required to carry on production activities within company **A**[2].

Company **A** may wish to minimise risk in its operations by limiting the number of open requests sent to company **B** to a fixed value N: once **A** has sent N requests to **B**, it will only send a new request after **B** has fulfilled at least one of the queued requests.

In order to model this small example, the graph consists of two locations s_A and s_B, and edges connecting these two locations in both directions (Figure 2). Agents, in this example, represent orders: when company **A** places an order, an agent is sent from s_A to s_B, and when this order is delivered by company **B** the agent is sent back from s_B to s_A. In Figure 2 we depict agents as black dots. In that figure, company **A** accepts to have seven simultaneous open orders at

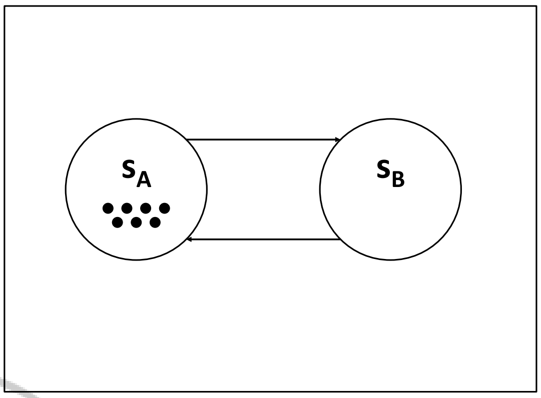

Figure 2: The two locations and corresponding agents for the customer-supplier example.

most (i.e. $N = 7$), as shown by the seven agents that are inside s_A.

The following three small interaction protocols implement this interaction[3]:

- *Protocol 1*:

  1. $\langle [i]^{s_A} req1(X, Order),$
  2. $[i]_X^{s_A \to s_B} mv \sqcap [e]_X^{s_B} write(message(Order), X) \sqcap$
  3. $[i]^{s_B} supply(X, Order)$
  4. $\rangle.$

- *Protocol 2*:

  1. $\langle [i]^{s_B} supply(Y, W),$
  2. $[e]_Y^{s_B} message(W) \sqcap [e]_Y^{s_B} prOrder(W) \sqcap$
  3. $[i]^{s_A} req2(Y, W)$
  4. $\rangle.$

- *Protocol 3*:

  1. $\langle [i]^{s_A} req2(Z, U),$
  2. $[e]_Z^{s_B} del(message(U), Z) \sqcap [i]_Z^{s_B \to s_A} mv \sqcap$
  3. $[k]_Z^{s_A} orderEnd(U) \sqcap [e]_Z^{s_A} contProd(U)$
  4. $\rangle.$

Interaction protocols 1 and 3 reside in a host managed by company **A**, and interaction protocol 2 resides in a host managed by company **B**. $[i]^{s_A} req1(X, Order)$ triggers the interactions, by asking an agent X in location s_A to start interaction $req1$, in which order $Order$ will be requested to company **B**. This is performed by moving the agent to location s_B, where it registers the order and triggers protocol 2.

By pattern matching on the right hand side of protocol 2, the message stored in location s_B containing the specification of the order is verified as being part of the knowledge of Z while in s_B, based on which the

[2]This example is borrowed and adapted from (Correa da Silva et al., 2012). Evidently, we are exhibiting only a very small fraction of a model for S^3 using this example. Our goal is simply to illustrate how rules that could be used to model a S^3 would look like.

[3]We adopt the Prolog convention that variables begin with capital letters, and all other terms begin with small letters.

order is processed (by triggering the action predicate *prOrder*) and finally protocol 3 is triggered.

By pattern matching on the right hand side of protocol 3, the message is deleted and the agent is moved back to location s_A, then it is checked whether the order has been properly delivered (using the knowledge predicate *orderEnd*) and the protocol returns the control back to company **A** internal actions (using the action predicate *contProd*).

This is a simplified example, in which success/failure verifications of performed operations and security issues are not taken into account. Additional features can be implemented by extending these protocols and/or by adding special purpose protocols, towards the design of interaction rules that can specify and characterise successful relations between companies in a supply chain.

Other protocols can be designed to compete with these protocols, and protocols can also be designed to characterise cooperative behaviour of suppliers to provide combined services to customers, towards the design of interaction rules that can specify and characterise relations between companies in a network. Hence, the relations involving companies in a netchain, i.e. a network of relations mixing supply chains and cooperative/competitive relations (Lazzarini et al., 2001) can be designed.

Based on theoretical analysis of properties of the interaction protocols, as well as empirical analysis of actual relations that can result from the use of these protocols, iterative refinements and adjustments can be made.

6 CONCLUSION AND FUTURE WORK

In this article we have introduced the LS^2C platform to design, implement and execute social computations, and sketched how it can be used to model and support a complex system of economic relevance, namely the organisation of companies in a business cluster according to S^3.

A platform for social computations should present features such as:

- The possibility to empower domain experts and end users to build specifications and execute them,
- Technology-agnosticism, meaning that implementations can be built based on various and diverse software platforms, operating systems and programming languages,
- Explicit account of participants in social interactions and their possible behaviours,
- Resources for the design of interaction protocols as well as for the analysis of existing protocols, including formal analysis based on algebraic and logical concepts.

The design and implementation of the LS^2C platform is work in progress. Since it inherits features and properties of *LCC/LSC* as well as of *JamSession*, we claim that this platform addresses all these features.

Our immediate future work concerns the implementation of the LS^2C platform, and its field test in the development of realistic social computers.

The specification of social interactions as characterised in the LS^2C platform can be used at least in three different ways:

1. As a design tool to specify desired features of interaction protocols in a decentralised way,
2. As a platform for the execution of social computations, and
3. As a tool to reason about specifications, including strategic reasoning (e.g., given alternative protocols that can be built, what is best for me/my company?), whereby participants may try out certain behaviours "in vitro" before these can be actually enacted.

LSC has been combined with an existing social network platform (Murray-Rust and Robertson, 2014), and *JamSession* has been combined with an existing workflow management platform (David, 2012). We envisage that a full *LS^2C Platform* can be implemented as the combination of a novel implementation of the LS^2C language, a workflow management system (e.g. Bonita[4]) and a social network platform (e.g. elgg[5] or eXo[6]). The implementation of the LS^2C language shall benefit from previous experience implementing *LSC* and *JamSession*.

We are particularly interested in the characterisation of Smart Specialisation Strategies (S^3) as a discipline to steer the emergency of networks of social interactions involving human-powered agencies aiming at regional economic efficacy. We believe that this approach can be appropriate to implement S^3 effectively, and that the LS^2C platform can be useful to support the design and operation of business clusters following S^3. In future work, we shall explore these views, hopefully through the analysis of empirical data resulting from the actual structuring of clusters of innovation as goal-oriented social interaction networks.

[4]http://www.bonitasoft.com/
[5]http://elgg.org/
[6]http://www.exoplatform.com/

ACKNOWLEDGEMENTS

This work has been partially supported by FAPESP and CNPq.

REFERENCES

Bai, X., Klein, E., and Robertson, D. (2012). *Choreographing web services with semantically enhanced scripting*, pages 583–587. Web Intelligence and Intelligent Agent Technology.

Christoff, Z. and Hansen, J. U. (2013). A two-tiered formalization of social influence. In *Logic, Rationality, and Interaction*, volume 8196 of *Lecture Notes in Computer Science – Springer LNCS 8196*, pages 68–81.

Correa da Silva, F. S. (2011). Knowledge-based interaction protocols for intelligent interactive environments. *Knowledge and Information Systems*, 30:1–24.

Correa da Silva, F. S., Fernandez Venero, M. L., David, D. M., Saleem, M., and Chung, P. W. H. (2012). Interaction protocols for cross-organisational workflows. *Knowledge Based Systems*, 37:1–16.

Correa da Silva, F. S., Robertson, D., and Vasconcelos, W. (2013). Experimental interaction science. *Artificial Intelligence and Simulation of Behaviour - Annual Convention 2013: Workshop on Social Coordination – Principles, Artifacts and Theories*.

David, D. M. (2012). *Protocolos de interacao baseados em conhecimento : implementacao da plataforma JamSession (in Portuguese)*. MSc dissertation, University of Sao Paulo, Brazil, Brazil.

EU, E. C. (2012). *Guide to Research and Innovation Strategies for Smart Specialisation*. European Union, Brussels.

EU, E. C. (2013). *The role of clusters in smart specialisation strategies*. European Union, Brussels.

Feldman, M. P. and Audretsch, D. B. (1999). Innovation in cities: science-based diversity, specialization and localized competition. *European Economic Review*, 43:409–429.

Fernandez Venero, M. L. and Correa da Silva, F. S. (2013a). *Modelling and simulating interaction protocols using Nested Petri Nets*. Workshop on Formal Methods in the Development of Software.

Fernandez Venero, M. L. and Correa da Silva, F. S. (2013b). *On the use of SPIN for studying the behavior of nested Petri nets*. 16th Brazilian Symposium on Formal Methods.

Gelernter, D. (1985). Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7:80–112.

Giunchiglia, F. and Robertson, D. S. (2010). The social computer - combining machine and human computation. *University of Trento Technical Report*, DISI-10-036.

Hansen, J. U. (2014). Reasoning about opinion dynamics in social networks. In *Proceedings of the eleventh conference on logic and the foundations of game and decision theory (LOFT 11)*.

Jackson, M. O. (2008). *Social and Economic Networks*. Princeton University Press, USA.

Jiang, S., Fiore, G. A., Yang, Y., Ferreira Jr, J., E., F., and Gonzalez, M. C. (2013). A review of urban computing for mobile phone traces: current methods, challenges and opportunities. In *Urban Computing 2013*.

Komninos, N. (2006). The architecture of intelligent cities: integrating human, collective, and artificial intelligence to enhance knowledge and innovation. In *Intelligent Environments 2006*.

Lazzarini, S. G., Haddad, F. R., and Cook, M. (2001). Integrating supply chain and network analyses: the study of netchains. *Journal of Chain and Network Science*, 1(1):7–22.

Mesquita, L. M. and Lazzarini, S. G. (2007). Horizontal and vertical relationships in developing economies: implications for smes access to global markets. *Academy of Management Journal*, 51(2):359–380.

Murray-Rust, D. and Robertson, D. (2014). *LSCitter: building social machines by augmenting existing social networks with interaction models*. International World Wide Web Conference Committee.

Pedrozo, E. A. and Pereira, B. A. D. (2006). Empreendedorismo coletivo e possivel? uma analise do processo de constituicao de relacionamentos cooperativos em rede. *Revista de Administracao (in Portuguese)*, 12(4).

Robertson, D. (2004). *Multi-agent coordination as distributed logic programming*, pages 416–430. Proceedings 20th International Conference on Logic Programming – Springer LNCS 3132.

Schaffers, H., Komninos, N., Pallot, M., Trousse, B., Nilsson, M., and Oliveira, A. (2011). *Smart cities and the future internet: towards cooperation frameworks for open innovation*, pages 431–446. Future Internet Assembly – Springer LNCS 6656.

Seligman, J., Liu, F., and Girard, P. (2011). *Logic in the Community*, pages 178–188. Logic and Its Applications – Springer LNCS 6521.

Sindhu, J., Perreau De Pinninck, A., Robertson, D., Sierra, C., and Walton, C. (2006). *Interaction model language definition*. Open Knowledge Project – technical reports, UK.

Zhen, L. and Seligman, J. (2011). A logical model of the dynamics of peer pressure. *Electronic Notes in Theoretical Computer Science*, 278:275–288. Proceedings of the 7th Workshop on Methods for Modalities (M4M2011) and the 4th Workshop on Logical Aspects of Multi-Agent Systems (LAMAS2011).

Zheng, Y., Liu, Y., Yuan, J., and Xie, X. (2011). Urban computing with taxicabs. In *UbiComp 2011*.