Charters for Self-Evolving Communities

Nardine Osman, Carles Sierra, and Marco Schorlemmer

Artificial Intelligence Research Institute, IIIA-CSIC, Barcelona, Spain

Abstract. Self-organisation and self-evolution is evident in physics, chemistry, biology, and human societies. Despite the existing literature on the topic, we believe self-organisation and self-evolution is still missing in the IT tools we are building and using. Instead of creating numerous rigid systems, we should aim at providing tools for creating self-evolving systems that adapt to the ever evolving community's needs. This paper proposes a roadmap for self-evolution by presenting a set of building blocks, which we refer to as community charters. The paper also presents an approach for each of these blocks, helping build the first prototype for self-evolving communities.

Keywords: Community goals, norms, interaction protocols, self-evolution

1 Introduction

Whilst many approaches have been proposed for defining and implementing communities (such as using the notion of organisations [8] or institutions [5]), we believe the current literature lacks the in-depth study of computational accounts of communities' evolution. The very foundation of autonomous agent research is based on the idea that agents evolve, for instance by assuming their beliefs or goals change over time. But how communities as a whole evolve is an overlooked question that this paper aims at raising and addressing.

We argue that just like human communities, e-communities (defined by their software) also need to self-evolve. Instead of creating numerous rigid systems, what we should aim at instead is providing tools for creating self-evolving systems that adapt to the community's needs. We believe different communities should be governed by different rules. These rules should be an ever evolving set resulting from the aspirations of its members. Furthermore, for the community's rules to be effective, they need to be tailored to the specific character traits of the community members as well as considering some other external influences. We note that in this paper we talk about self-evolution, as opposed to evolution, since we are interested evolution that is designed and directed by the community itself.

This paper proposes a roadmap for self-evolving communities. It proposes a set of building blocks needed for self-evolution, and we refer to these building blocks as the community charter. The paper also presents an approach for each of these blocks, helping us build the first prototype for self-evolving communities.

We say communities are formed around certain goals that community members are interested in fulfilling. Their evolution is driven by the fulfilment (or unfulfillment) and the evolution of these goals. Their fall is usually triggered either by the fulfilment or the abandoning of such goals by community members.

In addition to goals, we say interactions are another fundamental constituent of communities. A community, by definition, is a group of interacting peers (where peers may be a combination of humans, agents, and services). Interactions are the backbone that glues a community together. Without interactions, a community is simply a number of individuals. However, we say a community's interactions should aim at fulfilling its goals. Otherwise, either its interactions are ineffective, or its goals have not been properly thought-out.

Between goals and interaction protocols lie the community's norms. Norms may be thought of as describing the declarative rules, i.e. statements that describe *what* a rule is without going into the details of *how* to implement it. Norms provide generic guidelines that help fulfil the community's goals, and the interaction protocols (or the procedural rules) should be designed to abide by these norms.

As such, we propose to define a community by its charter, which we say is composed of the community's goals, norms, and interaction protocols. The concept of such a charter maps with the notion of traditional human communities, which are usually defined by their mission statement (the goals), their bylaws (the norms), and their standard operating procedures (the interaction protocols) [7].

Finally, we say unfulfilled goals can be one automated way for triggering evolution, suggesting a flaw with the charter's components. Different elements of the charter may be modified during the evolution stage in the hope that a more coherent charter is achieved.

The remainder of this paper elaborates further on each of the charter's three components in Sections 2, 3, and 4, respectively. Section 5 then provides a brief insight on how the proposed model helps promote self-evolution, before concluding with Section 6.

2 Goals

A community usually has a general goal which may then be divided into more concrete sub-goals. In human communities, these are defined by the community's *mission statement*.

For instance, we consider the u-Help community, whose software platform is used "for building a community of helpful people and supports them in finding volunteers for day-to-day tasks" [9]. Although the illustrative application for u-Help is to allow parents find volunteers for picking their children up from school or babysitting them, u-Help may be applied to any community with any needs for services. The u-Help community's mission statement may be paraphrased accordingly: *To help community members exchange services in a community with high needs for such services.*

This general mission statement may then be divided into more concrete goals, whose fulfilment may be verified:

G1. To ensure the community's needs for services are being addressed
G2. To ensure the satisfaction of requesters (i.e. to ensure good quality of service)
G3. To ensure the satisfaction of volunteers

These goals are defined by the community members that establish this community. Although goals may evolve over time like any other charter component, their evolution is usually much less frequent in comparison with the evolution of norms, and interaction protocols. This is because communities are essentially viewed as defined by their goals.

2.1 Goal Specification

In multiagent system, goals have been studied extensively as being part of an agent's belief-desire-intention (BDI) model [12]. In such models, goals are viewed as the desires that the agent adopts for active pursuit. When an agent commits to a specific plan for achieving a given goal, then the goal becomes an intention.

In this paper, we do not discuss agent goals, but community goals. Different approaches and/or languages may be adopted for the specification of these goals. We say one way to specify a goal is as a tuple:

$$\langle GId, GSpecification, GDescription \rangle$$

where GId is the goal's unique identifier, *GSpecification* is the goal's specification in first-order logic, and *GDescription* is the description of the goal in text, which may be used to aid human users understand which goals have been fulfilled or not during the evolution stage of a community (we assume community members may be a mix between agents and human users).

Goal Specification Example. The uHelp community's goals may be specified as:

$$\langle G1, \exists R' \subset R \cdot (\forall r \in R' \cdot \exists m \in M \cdot volunteer(r) = m \wedge majority(R', R)), \text{“}\textit{The majority of requests had volunteers to carry them out}\text{''} \rangle$$

$$\langle G2, \exists R' \subset R \cdot (\forall r \in R' \cdot \exists m \in M \cdot volunteer(r) = m \wedge pstvRate(m, r) \wedge majority(R', R)), \text{“}\textit{The majority of requesters are happy with the volunteers' performance}\text{''} \rangle$$

$$\langle G3, \exists R' \subset R \cdot (\forall r \in R' \cdot \exists m \in M \cdot requester(r) = m \wedge pstvRate(m, r) \wedge majority(R', R)), \text{“}\textit{The majority of volunteers believe the requests are reasonable/doable}\text{''} \rangle$$

where, M is the set of all community members, R is the set of all requests for help that community members have issued, $majority(R', R)$ implies that the elements of set R' constitute a majority with respect to the elements of the set R, $requester(r) = m$ specifies that the community member m requested help with task r, $volunteer(r) = m$ specifies that the community member m has volunteered (and been assigned) to fulfil the request r, and $pstvRate(m, r)$

describes that the community member m has been positively rated for task r. If m was a volunteer, then positively rating a volunteer will describe the requester's satisfaction with the volunteer's performance. If m was a requester, then positively rating a requester will describe the volunteer's belief that the request was for a reasonable (or doable) task with a reasonable deadline.

We note that in this specific example, goal $G2$ subsumes goal $G1$, and hence, the community's set of goals may be reduced to goals $\{G2, G3\}$.

2.2 Checking Goal Satisfaction

We say the unfulfillment of community's goals is one of the triggers for self-evolution. When such a situation arises, community members should be alerted by the system, which would then suggest that the charter may need to be revised as it is failing to fulfil its goals. Of course, there may be other triggers specified by the norms and interaction protocol, which we do not discuss here, such as stating which members are allowed to initiate evolution.

When to check for the satisfaction of community goals is also something that needs to be specified by the interaction protocol. For instance, should this check happen on a daily basis? Should it happen every time a new set of 100 requests is issued? This is an issue to be decided by the community itself (or those members who are given the right to do so) and specified accordingly by the interaction protocol.

Nevertheless, we say the minimum requirement for implementation is for interaction protocols to be capable of calling the goal satisfaction checker. Section 5 elaborates further on this.

3 Norms

Norms describe the rights and duties of community members. We say there are two different types of norms, those that are regimented by the interaction protocol, and those that are enforced by other means (such as punishments and rewards) as they cannot be regimented by the interaction protocol. An example of the former is the norm that states that a buyer cannot rate the seller more than once, and the system prevents the buyer to do so. An example of the latter is the norm that states that only people with sufficient credit can bid, where the credit is private information that cannot be accessed and assessed by the system.

3.1 Regimented Norms

Regimented Norms Specification Although numerous logics have been proposed in the literature [6], mostly using some kind of modal logic, the most common approach for specifying norms is through deontic logic, the logic of duties that deals with concepts like permissions and prohibitions. We believe the literature is rich enough with logics for one to choose from.

In this paper, we propose to specify regimented norms in a simplified deontic-based approach:

$$\langle NormId, NormType, Agents, Action, Condition \rangle$$

where $NormId$ is the norm's unique identifier, $NormType = \{permissible, omissible, obligatory, impermissible, optional\}$ specifies the type of the norm (we define the main five deontic operators of the Traditional Scheme [11] that describe what is: permissible, omissible, obligatory, impermissible, and optional), $Agents$ describes the set of agent roles that this norm applies to, $Action$ specifies the action the norm addresses, and $Condition$ specifies the circumstances under which the norm holds. Of course, as with goals, one may add a descriptive field to norms to aid humans in comprehending the norms they are discussing or voting on. Although this would raise a security concern of how to make sure the text properly describes the logic.

Regimented Norms Example. As an example, we specify a couple of norms from an existing service exchange community, the Time Bank community of the Castlehaven Community Association (`www.castlehaven.org.uk`). The norm description (and number) is taken from the community's time bank community rules [17].

3. Volunteers can live outside Camden and join the Time Bank to join in activities and help those who live in the Time Bank area.

$$\langle 3, permissible, member(V), volunteer(V, Task), \\ live\_outside\_TB\_area(V) \rangle$$

5. Everyone who requests help from the Time Bank will be put on a waiting list.

$$\langle 5, obligatory, system, put\_on\_waiting\_list(Task), \\ request(R, Task) \wedge \neg live\_outside\_TB\_area(R) \rangle$$

Regimented norm (3.) states that if a member ($member(V)$) lives outside the Time Bank area ($live\_outside\_TB\_area(V)$), then he is permitted to volunteer for a task ($volunteer(V, Task)$). Regimented norm (5.) states that if a requester R requests help with some task $Task$ ($request(R, Task)$) and the requester lives within the Time Bank area ($\neg live\_outside\_TB\_area(R)$), then the system is obliged to accept the task by putting it on the waiting list ($put\_on\_waiting\_list(Task)$).

Verifying Norm Regimentation Naturally, when norms need to be regimented by the interaction protocol, there should be means for automatically verifying this regimentation each time the norms or the interaction protocols change. As such, we say there is a need for automatic verification, which should happen once at the formation of the community and then again after each evolution.

Automated theorem proving or model checking are popular approaches with rich existing literature [16]. We propose to use the model checker of [14] that can help verify on the fly whether the norms specified in the proposed syntax above are satisfied in an interaction model specified in LCC [15], which is a lightweight process calculus for specifying multiagent interactions. Naturally, other languages may be used for specifying both the norms and the interaction model and other model checkers may be used for verification. However, in such cases, an appropriate translator is needed to translate the chosen language into the input language of the chosen model checker.

3.2 Enforced Norms

Enforced Norms Specification Enforced norms are norms that cannot be regimented by the system. They are then enforced by alternative methods, such as applying sanctions (punishments and rewards). Sanctions usually apply to prohibitions and obligations. As such, while other modal logics may be used for specifying regimented norms, enforced norms usually rely on deontic logic [1], as it is the logic of prohibitions and obligations.

Although any of the existing logics may be chosen, in this paper, we propose to specify enforced norms by extending the specification of regimented norms with sanctioning information. The proposed approach follows the same style as regimented norms and is coherent with many existing approaches [10, 2].

$$\langle NormId, NormType, Agents, Action, Condition, \\ Reward, Punishment, Deadline \rangle$$

As in the case of regimented norms: $NormId$ is the norm's unique identifier, $Agents$ describes the set of agent roles that this norm applies to, $Action$ specifies the action the norm addresses, and $Condition$ specifies the circumstances under which the norm holds and it is specified in first order logic. However, $NormType \in \{impermissible, obligatory\}$ is now restricted to obligations and forbiddances only, as we shortly explain. $Reward$ and $Punishment$ specify the rewards and punishments that an agent receives if they abide to or break the norm, respectively. Last, $Deadline$ specifies the deadline for an action to be prohibited or obligatory. The use of deadlines is clarified further in Section 3.2.

Concerning the restriction of the norm type to obligations and prohibitions, we say enforced norms rely on the concept of sanctions, and sanctions are usually assigned not to permissible actions that one is free to perform or not, but to negative permissions that describe what one is *not* permitted to perform. The concept of punishment and reward only makes sense when addressing negative permissions. We note that when defining deontic operators, one can pick any of the operators to be the basic operator, and the remaining operators may be defined in terms of the chosen basic operator. To illustrate our argument, we choose the permissible operator to be the basic operator, as such:

$$
\begin{aligned}
& \textit{permissible}\ \phi \\
\textit{omissible}\ \phi = \quad & \textit{permissible}\ \neg\,\phi \\
\textit{impermissible}\ \phi = \neg\ & \textit{permissible}\ \phi \\
\textit{obligatory}\ \phi = \neg\ & \textit{permissible}\ \neg\,\phi \\
\textit{optional}\ \phi = \quad & \textit{permissible}\ \phi\ \vee\ \textit{permissible}\ \neg\,\phi
\end{aligned}
$$

As enforced norms focus on negative permissions only, then the two operators that describe negative permissions are prohibitions (describing impermissible actions) and obligations.

A Note on Sanctions. It may be argued that punishment and reward is not always the right approach for motivating the abidance to norms. Furthermore, what may be considered a punishment for one may be viewed as a reward for another. In this paper, we label post-conditions as rewards or punishments, although they may simply be interpreted as the post-conditions of abiding with or breaking the norm.

Enforced Norms Example. As an example, we specify the following two norms:

1. Volunteers are penalised by losing credit if they do not fulfil their duties on time.

$$
\begin{aligned}
\langle 1, \textit{obligatory}, \textit{volunteer}(V), \textit{fulfil\_duty}(V, Task), \textit{assigned\_duty}(Task, V), \\
\textit{gain\_points}(Task), \textit{lose\_points}(Task), \textit{deadline}(Task)\rangle
\end{aligned}
$$

2. Requesters are not allowed to ask for tasks that are paid.

$$
\begin{aligned}
\langle 2, \textit{impermissible}, \textit{requester}(R), \textit{request\_help}(R, Task), \textit{paid\_service}(Task), \\
\textit{nil}, \textit{prohibited\_to\_request}(Next\_10\_days), \textit{nil}\rangle
\end{aligned}
$$

The first enforced norm states that if a task has been assigned to a volunteer V ($\textit{assigned\_duty}(Tasl, V)$) then the volunteer is obliged to perform this task ($\textit{fulfil\_duty}(V, Task)$) within the task's deadline ($\textit{deadline}(Task)$). If he succeeds, then he is rewarded by gaining a certain number of points ($\textit{gain\_points}(Tasks)$); and if he fails, he is punished by losing a certain number of points ($\textit{lose\_points}(Taks)$).

The second enforced norm states that a requester R is forbidden to request help ($\textit{request\_help}(R, Task)$) if the requested task is a paid service ($\textit{paid\_service}(Task)$). This rule has no deadline ($\textit{nil}$); i.e. it holds forever. Requesters are not rewarded for not requesting help with paid services ($\textit{nil}$), but punished if they do by being prohibited by the system from requesting any help over the next 10 days ($\textit{prohibited\_to\_request}(Next\_10\_days)$).

Ensuring Norm Enforcement Several approaches exist that propose norm enforcement mechanisms [13, 3]. In this paper, we propose a basic norm enforcement algorithm, whose pseudocode is presented by Figure 1. The algorithm

essentially states that norms become active (or are instantiated) for a given community member if the condition of the norm holds for that given community member. Community members are then either rewarded or punished either when they perform the action in question or when the deadline passes and they have not yet performed the action in question, depending on the type of the norm. Note that the algorithm is event triggered: The event of having a norm's condition satisfied triggers the activation (and instantiation) of that norm, and the events of performing an action or having a norm's deadline pass trigger sanctions and the removal of the instantiated norm.

```
OnEvent: Constraint C of norm n is satisfied for α
     Do: Instantiate norm n for agent α

OnEvent: α performs action A and there exists an
         instantiated norm n on α's action A
     Do: If norm n is of type impermissible Then
            Agent α is punished and
            the instantiated norm is deleted
         Else
            Agent α is rewarded and
            the instantiated norm is deleted

OnEvent: Deadline of instantiated norm n passes
     Do: If norm n is of type impermissible Then
            The corresponding agent is rewarded and
            the instantiated norm is deleted
         Else
            The corresponding agent is punished and
            the instantiated norm is deleted
```

Fig. 1. Pseudocode for norm enforcement

We note that the system responsible for the execution of interactions should also be responsible for the norm enforcement algorithm, as it needs to keep track of which conditions are being satisfied, which actions are performed, and which deadlines have passed.

4 Interaction Protocols

Interaction protocols describe the operational rules governing the interaction between community members. In abstract terms, they may be specified via labelled transition systems or finite state machines. In multiagent systems, several approaches have been used that one is free to choose from, such as using process calculi [15], electronic institutions [5], or contracts and commitments [4].

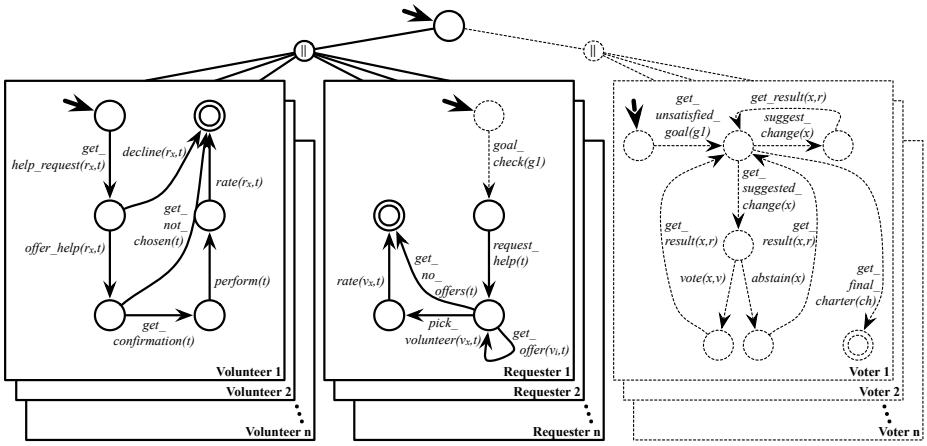

Fig. 2. Interaction protocol example

We say any approach may be adopted, as long as a model checker can verify the specified interaction protocols against regimented norms. In this paper, however, we choose the lightweight coordination calculus (LCC) [15] as it is a process calculus that is used both in the specification of multiagent systems as well as in the execution of their interactions, and more importantly, it is the language of [14]'s model checker. We believe that having the executable interaction model fed directly to the model checker avoids the complexity of modelling the system in another language and the possibility of introducing errors in doing so.

Interaction Protocol Example. Figure 2 provides an example of an interaction protocol, where the nodes present the state of a process (or agent). The protocol states that there can either be a number of volunteers and requesters playing at the same time (note that one agent may play more than one instance for more than one role at the same time), or a number of voters discussing the evolution of the community. Note that || describes the parallel operator in process calculus. As an example, we explain the voter's role, and we leave it to the reader to interpret the rest of the specification, as we believe the names of the agent roles and actions are self-expressive.

The voter's protocol states that when an agent plays the role of a voter, first, it will receive a message stating the unsatisfied goal $g1$ that has initiated the evolution stage ($get\_unsatisfied\_goal(g1)$). Then, either the voter suggests a change x in the charter ($suggest\_change(x)$), or it receives a suggested change x by some other voter ($get\_suggested\_change(x)$). In the first case, it will wait for others to vote on its suggested change, before receiving the final result r of the vote ($get\_result(x, r)$). In the latter case, it can either vote v ($vote(x, v)$), or it can abstain from voting ($abstain(x)$). In both cases it will then be informed of the final voting result r ($get\_result(x, r)$). As the arrows illustrate, the protocol may loop several times with different voters suggesting new changes and voting

on the suggestions, before the final charter ch is agreed upon and the voter is informed ($get\_final\_charter(ch)$).

The reader familiar with process calculi may note that mapping this specification into a process calculus such as LCC becomes straightforward.

5 Self-Evolution

The interaction protocols are expected to specify the details of evolution. They should specify when does evolution take place and how does the system trigger evolution (such as when goals are not satisfied), which community members are allowed to suggest evolution (such as permitting the community's president to trigger evolution whenever he sees fit), the minimum number of people required to discuss evolution (such as stating that at least 30% of the community should be present for discussing evolution), or who can suggest changes and how evolution is discussed and agreed upon (such as following some predefined voting mechanism).

Ideally, there would be specific processes dedicated for evolution. That is if we are thinking of interaction protocols specified in a process calculus. For instance, if one uses electronic institutions to model interaction protocols, then one would think of a dedicated evolution *scene*. In the example presented by Figure 2, the dotted processes, states, and actions are those related to evolution. The processes defining the voters' roles illustrate how voters may vote on recommended changes. And it is the $goal\_check(g1)$ action in the requester's role that is responsible for initiating evolution. Although, we note that for simplification, the conditions that control the flow are omitted.

Naturally, there is the crucial issue of defining how the interaction needs to be paused at a safe state from which it can be resumed after evolution is completed. We leave this open issue for future research. However, we say after evolution takes place and a new charter is agreed upon, all community members need to be informed of the new charter.

6 Conclusion

This paper argues the need for self-evolving communities. We believe self-evolution is still missing in the IT tools we are building and using. And we say that instead of creating numerous rigid systems, we should aim at providing tools for creating self-evolving systems that adapt to the ever evolving community members' needs.

The paper proposes a roadmap for self-evolving communities. It proposes a set of building blocks, which we refer to as the community charter, and presents a concrete approach for each of the proposed building blocks, which helps build the initial prototype for self-evolving communities. We say a community charter should define: (1) the community's goals, (2) the community's norms, and (3) the interaction protocols, which include the evolution protocols.

One proposed approach for triggering evolution is for the system to signal unfulfilled goals, suggesting the modification of the charter's various components in a way that helps address these unfulfilled goals. Future work could also make use of the research on emergence and self-organisation in multiagent systems.

Finally, we note that our ongoing research considers semantics to be a fundamental component of any charter. For instance, unfulfilled community goals might be the result of misunderstandings at the semantic level. In this line of work, where community members are expected to propose and discuss new goals, norms, and/or interaction protocols, ensuring that all members comprehend these elements in a consistent manner becomes a major and crucial challenge.

7 Acknowledgments

This work is supported by the PRAISE project (funded by the European Commission under the FP7 STREP grant number 318770), the CBIT project (funded by the Spanish Ministry of Science & Innovation under the grant number TIN2010-16306), and the Agreement Technologies project (funded by CONSOLIDER CSD 2007-0022, INGENIO 2010).

References

1. Ågotnes, T., Broersen, J., Elgesem, D. (eds.): Deontic Logic in Computer Science - 11th International Conference, DEON 2012, Bergen, Norway, July 16-18, 2012. Proceedings, Lecture Notes in Computer Science, vol. 7393. Springer (2012)
2. Aldewereld, H., Dignum, F., Garca-Camino, A., Noriega, P., Rodrguez-Aguilar, J., Sierra, C.: Operationalisation of norms for electronic institutions. In: Noriega, P., Vzquez-Salceda, J., Boella, G., Boissier, O., Dignum, V., Fornara, N., Matson, E. (eds.) Coordination, Organizations, Institutions, and Norms in Agent Systems II, Lecture Notes in Computer Science, vol. 4386, pp. 163–176. Springer Berlin Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-74459-7_11
3. Criado, N., Argente, E., Noriega, P., Botti, V.: A distributed architecture for enforcing norms in open mas. In: Dechesne, F., Hattori, H., Mors, A., Such, J., Weyns, D., Dignum, F. (eds.) Advanced Agent Technology, Lecture Notes in Computer Science, vol. 7068, pp. 457–471. Springer Berlin Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-27216-5_35
4. Dignum, V., Meyer, J.J., Weigand, H.: Towards an organizational model for agent societies using contracts. In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2. pp. 694–695. AAMAS '02, ACM, New York, NY, USA (2002), http://doi.acm.org/10.1145/544862.544909
5. d'Inverno, M., Luck, M., Noriega, P., Rodriguez-Aguilar, J.A., Sierra, C.: Communicating open systems. Artificial Intelligence 186, 38–94 (Jul 2012), http://dx.doi.org/10.1016/j.artint.2012.03.004
6. Gabbay, D.M., Guenthner, F. (eds.): Handbook of Philosophical Logic. Springer (2001 – to date)

7. Heimlich, J.E., Dresbach, S.H.: Written documents for community groups: Bylaws and standard operating procedures. Fact Sheet on Community Development, Ohio State University Extension, online: http://ohioline.osu.edu/cd-fact/co-bl.html; Last accessed on 03 October 2013
8. Horling, B., Lesser, V.: A survey of multi-agent organizational paradigms. Knowl. Eng. Rev. 19(4), 281–316 (Dec 2004), http://dx.doi.org/10.1017/S0269888905000317
9. Koster, A., Madrenas-Ciurana, J., Osman, N., Schorlemmer, W.M., Sabater-Mir, J., Sierra, C., Jonge, D.D., Fabregues, A., Puyol-Gruart, J., Garcia-Calves, P.: u-help: Supporting helpful communities with information technology. In: Ossowski, S., Toni, F., Vouros, G.A. (eds.) Proceedings of the 1^{st} Int. Conf. on Agreement Technologies. CEUR Workshop Proceedings, vol. 918, pp. 378–392. CEUR-WS.org (2012)
10. Lpez, F.y., Luck, M., dInverno, M.: A normative framework for agent-based systems. Computational & Mathematical Organization Theory 12(2-3), 227–250 (2006), http://dx.doi.org/10.1007/s10588-006-9545-7
11. McNamara, P.: Making room for going beyond the call. Mind 105(419), 415–450 (1996), http://mind.oxfordjournals.org/content/105/419/415.abstract
12. Meyer, J.J., Broersen, J., Herzig, A.: BDI logics. In: Ditmarsch, H.v., Halpern, J., van der Hoek, W., Kooi, B. (eds.) Handbook of Logics for Knowledge and Belief. College Publications, http://www.collegepublications.co.uk/ (2013)
13. Modgil, S., Faci, N., Meneguzzi, F., Oren, N., Miles, S., Luck, M.: A framework for monitoring agent-based normative systems. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1. pp. 153–160. AAMAS '09, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2009), http://dl.acm.org/citation.cfm?id=1558013.1558034
14. Osman, N.: Runtime Verification of Deontic and Trust Models in Multiagent Interactions. PhD thesis, School of Informatics, the University of Edinburgh, Edinburgh, UK (2008)
15. Robertson, D.: A lightweight coordination calculus for agent systems. In: Leite, J.a., Omicini, A., Torroni, P., Yolum, p. (eds.) Declarative Agent Languages and Technologies II, Lecture Notes in Computer Science, vol. 3476, pp. 183–197. Springer Berlin Heidelberg (2005), http://dx.doi.org/10.1007/11493402_11
16. Robinson, A., Voronkov, A. (eds.): Handbook of automated reasoning. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands (2001)
17. Time bank joining form. Castlehaven Community Association (May 2013), online: http://www.castlehaven.org.uk/static/uploads/documents/timebank_Application_form_May_2013.docx; Last accessed on 03 October 2013