Social-Compliance in Trust Management within Virtual Communities

Reda Yaich, Olivier Boissier, Gauthier Picard, and Philippe Jaillon

LSTI - Henri Fayol Institute
ENS Mines Saint-Étienne, France
{firstname.name}@emse.fr

Abstract. Virtual Communities are open socio-technical structures wherein autonomous entities (i.e. agents) with common interests join together to mutually satisfy their goals. The success of these communities relies on collaboration and resource sharing principals, making trust a priority for each member. Such communities need a more flexible trust model wherein both individual (i.e. user-centred) and collective (i.e. community-centred) trust requirements are considered in the decision making-process.
This paper reports our on-going efforts in that perspective and presents a multi-agent-based Adaptive and Socially-Compliant Trust Management System (ASC-TMS). Policies are used, in the system, to specify individual and collective trust requirements, while meta-policies enable agents to dynamically adapt their policies. The ASC-TMS allows agent to make socially-compliant trust decisions through automatic combination of individual and collective policies.

Keywords: Multi-agent Systems, Trust, Policies, Virtual Communities

1 Introduction

Virtual communities (VCs) are known to be dynamic, uncertain and risky environments wherein autonomous entities (i.e. agents) with common objectives join together to mutually satisfy their goals. The success of these communities relies on collaboration and resource sharing, making trust a critical issue for each member [11]. Policies are considered as a viable solution to capture and specify trust requirements [6, 7, 19, 26]. They constitute a flexible way to express both the conditions under which trust decisions have to be made, and the factors on which these decisions rely.

One of the main issues when specifying and enforcing trust policies – within open, dynamic and ever-evolving environments such as VC – is the ability of these policies to handle unforeseen situations. Such adaptation capability is essential to overcome unpredictable events, otherwise, running policies could become rapidly irrelevant or inefficient [9, 14]. Another important issue, when deploying trust policies in virtual communities, is the capability of trust management systems to combine individual and collective policies during the trust assessment. Seamless integration of aggregated trust

requirements along with individual ones has been recognized as an important factor of social integration in the social influence theory [4, 12, 21].

Both issues raised above make trust decisions within VCs complex and motivate the need for more flexible and better adaptive trust models. To that aim, we focus in this paper on endowing Trust Management Systems (TMSs) with such capabilities based on a rich, adaptive and socially-enabled trust model. By rich we mean that trust is built upon a wide – and unbound – range of trust factors. By adaptive we mean that policies are constantly tailored and can evolve over time. By socially-enabled we mean that trust decisions at the individual level are made by taking into account collective requirements of the community to which the agent belongs.

We propose an agent-based Adaptive and Socially-Compliant Trust Management System (ASC-TMS) wherein policies are considered as concrete implementations of trust requirements, meta-policies are used to automatically adapt individual policies, and where collective policies are combined with individual ones to help agents making socially-compliant trust decisions (i.e. in accordance with the requirements of their communities). To implement and illustrate desired aspects of the system, we used the JaCaMo platform [15], a new multi-agent oriented programming framework, which allowed us to address the issues discussed earlier while preserving the generality of the approach.

The rest of the paper proceeds as follows. In the next section we motivate our approach using an *Innovation Community (IC)* scenario wherein requirements for trust management has been identified. In Section 3, we formally introduce the system model, basic foundations and concepts our proposal is built on. In Section, 4 we outline the proposed trust model and its related concepts. In Section, 5 we illustrate how trust policies are specified, adapted, combined and evaluated. Finally, we briefly survey related works and conclude with discussion and some perspectives in Section 6.

2 Requirements for Trust Management in Virtual Communities

In this section, we first analyse trust issues in the case study motivating our work. We then use it to derive and argue a set of important features that should be addressed by the trust management system that we propose.

2.1 Motivating Example

Open Innovation [5] is currently recognized as a new model that companies and organizations are adopting to enhance innovation in their departments by harnessing external ideas and stimulating creativity. Based on this concept, a number of commercial (e.g. Hypios.com[1]) and free (e.g. W3C Community Group[2]) platforms have been proposed to facilitate the innovation process through the creation of Innovation Communities (IC). These platforms constitute a showcase for problems to which the community members try to find a solution. Members can try to solve individually proposed problems, but in most of the cases they join together and create dynamic communities wherein problems

[1] http://www.hypios.com/

[2] http://www.w3.org/community/about/#cg

are solved collaboratively. The success of these communities relies on full collaboration and massive resource sharing, making trust a critical issue for each member.

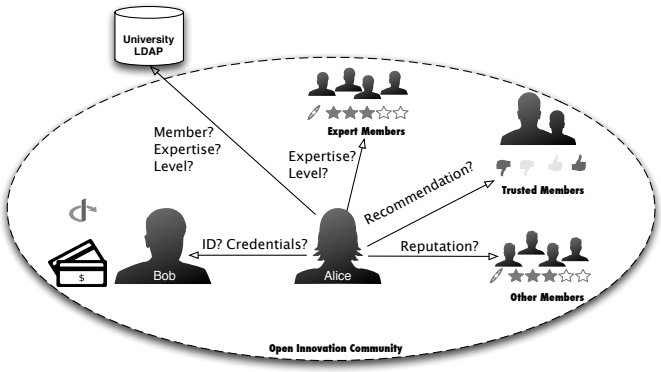

Fig. 1. Trust factors and information sources diversity within Innovation Communities.

Members of these communities like *Alice* and *Bob* have a variety of information sources they can use to estimate partners' trustworthiness. Each source represents a factor for trust establishment and can be used to specify a requirement within a policy. For example, in Figure 1, *Alice* can use four types of trust factors to evaluate *Bob*'s trustworthiness within the community: (i) she can use recommendations of members she trusts, (ii) she can compute *Bob*'s reputation based on the other members' past experiences with *Bob*, (iii) she can also ask confirmed members whether *Bob* is skilful in certain disciplines and what is his expertise level, and finally, (iv) she can ask *Bob* for credentials – he owns – to testify his identity, his skills or other properties. Then she can check the validity of these credentials from the *certification authority* that issued the credential (e.g. University). This example will be used throughout this article to motivate and illustrate different aspects of our approach.

2.2 Desiderata

The study of the above IC example has led us to identify important requirements for trust management, which are as follow.

Semantics. One of the characteristics of ICs is the great heterogeneity among its members. The problem lies in the fact that trust is built upon statements (e.g. credentials and reputations) that are manipulated by multiple parties, making semantic heterogeneity an important concern for trust management in virtual communities. The descriptions of trust factors must have a well-defined semantics [26], so that the policies specified based on this factors and the information used to satisfy this policies can be understood by all. For instance, *Bob* must have confidence that when he considers that the credentials he owns satisfy *Alice*'s policy, *Alice* should make the same conclusion.

Subjectivity. The heterogeneity among the community members raises also the problem of trust subjectivity. For instance, in Figure 1, Alice and Bob may rely on different trust factors when specifying the policy they use to protect a common resource. Historically, the majority of trust management models are built upon a controlled – and limited– set of trust factors. Such limitation represents a lock for subjectivity and constraints the expressiveness of trust requirements. Moreover, due to the dynamic character of ICs, some trust criteria can unpredictably become irrelevant. For instance, reputation pertinence is very sensitive to collusion or selfishness treats. So the requirement that we highlight here is the necessity for users to build their trust based on a rich and extensible set of trust information. Our TMS should allow community members to specify policies using various trust factors, characterizing the subjective nature of trust.

Adaptivity. ICs are open environments that members can spontaneously join and leave and where resources are unpredictably created, updated and destroyed. In such settings, future interactions are impossible to predict, making the specification of trust policies hazardous and risky [14]. The question one should ask, in this context, is not *"how efficient the policy I specified is?"* but *"how adaptive, in response to an ever-evolving environment, it could be?"*. For example, the policy that Alice is using relies on reputation (and other trust factors as well). The more Alice is aware of collusion risks, the less she will give importance to reputation factor in her final decision. Thus, her policies need to be automatically adapted; otherwise, she could make inappropriate decisions. So policies within virtual communities should not be specified *ad vitam æternam*[3]. The traditional *dogmatic* vision of trust policies is neither possible nor desirable and should even be proscribed.

Social-Awareness. Trust is closely tied to values, traditions, and norms of the society. Many sociologists [4, 12, 21] demonstrated the impact of social influence on our practices and attitudes. Trust practices are not an exception but, to the best of our knowledge, no research initiative was conducted in that direction in terms of trust management. *"Why should an individual integrate collective requirements when making trust-decisions?"*. Principally for two reasons: (i) *social exclusion* and (ii) *competitive exclusion*. Both effects arise when the decision of a member is too far apart from the decision of the other members of his community. For instance, members of the community in Figure 1 co-regulate the access to resources they produced collaboratively. If Bob does not comply with the requirements of his community (i.e. collective policies), his behaviour will be considered as an antisocial deviant behaviour, and can expose Bob to *social exclusion*. Also, ICs represent competitive environments where different communities are competing in solving the same problem. In such situation, using a too restrictive or a too permissive policy exposes agents to *competitive exclusion*. So, trust decision made by an agent in its community based on its individual requirements should be tempered by the integration of collective requirements.

[3] Specified once and executed for ever!

3 Multi-Agent System Model for Trust Management

The system (cf. Fig. 2) is represented by a multi-agent system $\mathcal{S}$. $\mathcal{S}$ is defined by $\langle \mathcal{C}, \mathcal{A}, \mathcal{R}, \mathcal{O}, \mathcal{I}, \mathcal{F}, \mathcal{T} \rangle$ where $\mathcal{C}$ is a set of communities c, $\mathcal{A}$ is a set of agents a, $\mathcal{R}$ is a set of resources r, $\mathcal{O}$ is a set of operations o, $\mathcal{I}$ is a set of interactions ι, $\mathcal{F}$ is a set of feedbacks e, and $\mathcal{T}$ is an ontology of trust factors in $\mathcal{S}$.

Communities are dynamic social structures that agents from $\mathcal{S}$ can join to solve specific problems in collaboration with other members of the community. Each community $c \in \mathcal{C}$ is represented by $\langle \varepsilon_c^{\mathcal{C}}, \Pi_c \rangle$ where $\varepsilon_c^{\mathcal{C}}$ is its unique identifier in $\mathcal{S}$ and Π_c is a set of *collective policies* that each member receives when he joins the community (defining how these policies are built is outside the scope of this paper. However, we consider this issue as an interesting perspective for this work). This *collective policies* are used to ensure minimal trust conditions community members should ensure during the interactions undertaken within the community. The *social compliance* of an agent depends on whether his trust decisions are made in accordance with this set of *collective policies* or not.

Agents are autonomous entities, able to perform operations on resources on behalf of the platform users. Autonomous means that no agent has direct control over operations performed by another agent. $\forall a \in \mathcal{A}$, $a = \langle \varepsilon_a^{\mathcal{A}}, \alpha_a, \beta_a, \kappa_a, \omega_a, \psi_a, \mathcal{TP}_a \rangle$, where, $\varepsilon_a^{\mathcal{A}}$ is the agent's unique identifier in $\mathcal{S}$, α_a is a set of credentials attesting its various properties, $\beta_a \in 2^{\mathcal{C}}$ is the set of communities that *a belongs* to, $\kappa_a \in 2^{\mathcal{O}}$ is the set of operations that a can perform (i.e. *capabilities*), $\omega_a \in 2^{\mathcal{R}}$ is the set of resources that a *controls*, $\psi_a \in 2^{\mathcal{A} \times \mathcal{F}}$ is a set of feedback the agent a maintains about other's behaviour after each interaction, and $\mathcal{TP}_a$ is a set of trust patterns (cf. Section 4 and 5 for more details about trust patterns). $\forall tp \in \mathcal{TP}_a$, tp includes individual policies ($p_a^i / p_a^i \in \pi_a$), associated meta-policies ($\mathcal{M}(p_a^i)$) and collective policies ($p_c^i / p_c^i \in \Pi_c$ and $c \in \beta_a$).

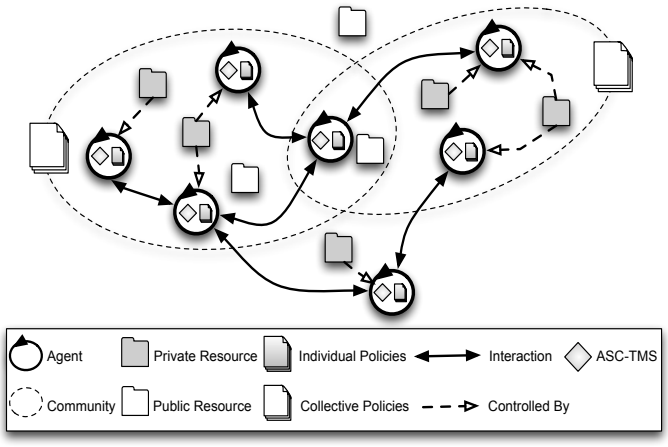

Fig. 2. Multi-Agent System Model for Trust Management

Example 1. *Eurêka* is the IC platform to which *Alice*, *Bob* and *Charlie* participate. They joined together and created an innovation community *Com*2 where they commit to find a solution for a specific problem (e.g *"reducing energy consumption"*). Every member participating in *Eurêka* is provided with an agent that assists him in his activities. Each agent relies on a dedicated Trust Management System when making trust-based decisions. In the following, capital letter names refer to members (e.g. *Alice*) while their assistant-agents' names are in lowercase (i.e. *alice*).

Resources are artifacts in $\mathcal{S}$ on which **operations** can be performed. $\forall r \in \mathcal{R}$, r is represented by $\langle \varepsilon_r^{\mathcal{R}}, \tau_r, \varsigma_r \rangle$ where: $\varepsilon_r^{\mathcal{R}}$ is its unique identifier, (ii) τ_r represents its type ($\tau_r \in \tau_{\mathcal{R}}$ / $\tau_{\mathcal{R}} = \{service, data, credential, object\}$) and ς_r qualifies its sensitivity, $\varsigma_r \in [0, 5]$. Each resource is either *private* or *public*. *Public* resources are passive artifacts that can be manipulated by any agent within the community, while *private* resources represent active artifacts that require a permission to be manipulated.

Example 2. The agent *alice* controls a *service* through which *Alice*, as a mathematician, provides mathematics skills to her community members. *alice* has also a pair of public-private key and a set of certificates, binding her key with *Alice*'s testified properties and attributes like her name, her address and her qualifications. These keys and certificates constitute the *credential*s she can use to establish trust with other members. During the innovation process, *Alice* can take some notes about ideas she had and she can send and receive private messages from other members. These are considered as *data* and are handled by *alice* like other resources. Finally, *alice* controls all objects *Alice* is using in her innovation activities such as scientific articles or applications binaries.

Interactions are message exchange sequences. Each message $m \in \iota$ ($\iota \in \mathcal{I}$) is defined by $\langle a_s, a_r, \tau_m, \theta \rangle$, where a_s, a_r are respectively the sender and the receiver of the message, $\tau_m \in \{request, inform, reply\}$ is the message's performative type and θ its content. As trust decisions are made with respect to requests, our focus here is on such messages where the content θ is a pair $\langle o, r \rangle$ stating that an agent a_r (called the requester) is asking another agent a_c (called the controller) the permission to perform an operation $o \in \mathcal{O}$ on a private resource $r \in \mathcal{R}$ that a_c controls. Each request is associated to a type based on the $(o, \tau_{\mathcal{R}})$ pair. These types are then used to build trust patterns.

Permissions are made through *reply* messages where the content θ is of the form *allow*(a_r, o, r) stating that the controller a_c accepted a_r's request. When the *reply* content is a *deny*(a_r, o, r) that means that the request was refused. Each permission is signed by its issuer and is later considered as a credential that testifies the agent's rights with respect to resources' manipulation. This permission can be revoked or out-dated and becomes invalid, but for simplicity reasons we will not address such issues in this paper.

Example 3. *alice*'s mathematic *service* is a sensitive resource. So when *bob* tried to use it he was asked a credential for that. In order to get that permission, he sent the request $\langle bob, alice, request, \langle access, mathAlice \rangle \rangle$ to *alice*. If *bob* credentials satisfy *alice*'s policy, *alice* will issue the permission $allow(bob, access, mathAlice)$. Otherwise, $bob's$ request will be refused.

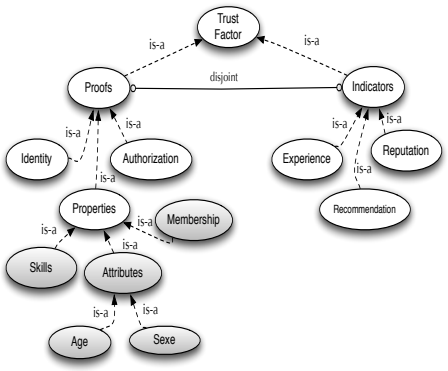

Fig. 3. Fragment of the Trust Factors Ontology: white concepts are context-independent while grey ones are context-dependent

The **Trust Ontology** $\mathcal{T}$ represent a semantic description of trust factors available in $\mathcal{S}$. $\forall f_i \in \mathcal{T}$, f_i is a concept defined by $(\tau_{f_i}, \Delta_{f_i})$ where τ_{f_i} is the type of f_i and Δ_{f_i} its domain (i.e. the set of values f_i can take). Each trust factor describes agent's characterizing traits (e.g. identity or reputation) on which restrictions can be specified using policies.

The first level root concept represents a generic trust factor type, it is further divided into two sub-types *proofs* and *indicators*: proofs represent agent's traits and properties that must be testified using credentials, while indicators include any fact of belief that are based on personal and pairs feedbacks (for more details see [28] and [27]).

Example 4 (proofs and indicators). Both *alice*, *bob* and *charlie* can issue permissions to use a common document they are working on in *Com*2. So when *alice* issues a permission for the agent *dave*, this credential is considered as a *proof* of trust for *bob* and *charlie* as they know that it means that *alice* trusts *dave*. But the fact that *Carol* is a good member within the *Eurêka* platform (e.g. with high reputation) can not be considered as an evidence. Yet, it can be considered as an *indicator* of trustworthiness.

4 An Adaptive and Socially-Compliant Trust Model

Based on concepts defined in the previous section and requirements formulated in Section 2, we built an Adaptive and Socially Compliant Trust Model (cf. Fig. 4). The proposed model is implemented in each agent as the Adaptive and Socially-Compliant TMS (ASC-TMS). We first briefly describe how agents use this model to build and manage trust, then we review each of the basic elements used in the model.

Each agent a is provided with a set of individual policies (π_a) and associated meta-policies ($\mathcal{M}(p_a)$). These policies and meta-policies are sorted and stored into *trust patterns* ($\mathcal{TP}_a$). When the agent a joins a community c, he retrieves the set of collective policies (Π_c) that he dispatches into the appropriate *trust patterns*. Once the agent a

receives a request, he identifies its type t based on the operation to be performed and the type of resource. Then he selects and activates the appropriate *trust pattern* that contains the *individual policy*, the *meta-policies* and *the collective policy* dedicated to that type of request. The *individual policy* is first adapted to the running context using associated meta-policies, then it is combined with the *collective* one (cf. Section 5.3). Finally the generated policy is evaluated with respect to trust information and a trust level is computed.

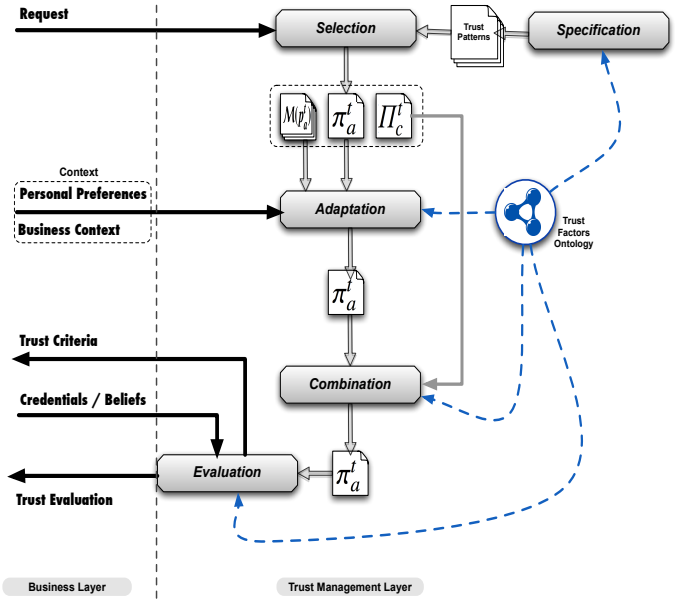

Fig. 4. The Adaptive and Socially-Compliant Trust Management System Model

Now we describe more in detail each of the elements used in the above description. **Trust Patterns** ($\mathcal{TP}_a$) are used to manipulate individual policies, meta-policies and collective policies with respect to request types. Indeed, as presented in Section 4, requests were classified into types based on the resource's type and the operation to be performed. Each type is associated to a trust pattern that integrates a generic policy, adaptation meta-policies used to adapt it and the collective policy used within the community.

Trust patterns have been introduced in our model to overcome limitations we faced when we tried to specify policies in ICs scenarios. The main challenge was that in traditional approaches, each resource is governed by a specific policy. But in virtual communities as the one addressed in this paper, resources are dynamic and unpredictable (e.g. spontaneously created and their sensitivity can change over time). This constraint obliged us to consider a new model where policies are specified independently from the resource they govern. So, the question was *how to protect resources that have not*

yet been created? and our solution was to focus on requests rather then the resources themselves.

Policies are statements that specify, for each request pattern, the set of conditions required for trust establishment. As described previously, each decision made by an agent $a \in \mathcal{A}$ within a community $c \in C$ to which it belongs considers two policies. An **individual policy** (π_a^t) that characterizes his personal trust requirements, and a **collective policy** (Π_c^t) that represents trust requirements shared by the members of c. Both *individual* and *collective policies* constitute a set $\{tc_1, tc_1, \ldots, tc_n, \}$ where every *trust criteria* tc_i ($i \in [1, n]$) represents a condition stated by the policy. A *trust criteria* tc_i is a triplet $\langle \tau_{f_i}, \aleph_i, w_i \rangle$. Where τ_{f_i}, is its type and corresponds to a trust factor $f_i \in \mathcal{T}$ within the ontology, $\aleph_i$ is a threshold value among possible values ($\aleph_i \in \Delta_{\tau_{f_i}}$) and $w_i \in \mathbb{N}^*$ represents the importance (i.e. weight) of tc_i in the final trust level computed when evaluating the policy.

Adaptation Meta-Policies ($\mathcal{M}(p_a^t)$) represent a sequence of domain-dependent event-condition-action rules. Agents use this rules to automatically adapt running policies in response to context events. Each rule is of the form:

$$Upon\langle event \rangle : If\langle condition \rangle \leftarrow Do\langle action \rangle \tag{1}$$

The $\langle event \rangle$ is a generic triggering event, the $\langle action \rangle$ is one or more adaptation action used while adapting policies, while the $\langle condition \rangle$ is a general expression, typically a conjunction or a disjunction of literals, to be checked before the execution of the rule. These conditions represent filters that are defined over the context.

Context information are domain-dependent beliefs agents rely on when adapting their policies. In our approach we used, but are not limited to, *personal preferences* and *business contexts*. **Personal Preferences** are personal input parameters an agent uses to know whether he has to adapt a policy or not.

Example 5. alice is using resource sensitivity and the associated reward to tune her policies. If the reward in granting access to her resources is higher than the potential risk deduced from its sensitivity, then she will be more likely to adapt her policy.

The **Business Context** has also a great influence on trust decisions. Whether the community is highly competitive or fairly cooperative, trust requirements can be formulated in different ways affecting the final decisions. This model enables agents to make context-aware trust decision based on such indicators. In this paper, for simplicity reasons, only few business parameters are considered. They are domain-dependent but most of them can be generalized to handle other VC scenarios. Our *Business Context* is qualified by the number of agents requesting the resource, the number of agents providing it, the population of the community and the past experience with the other members of the community[4].

5 The ASC-TMS Implementation

In this section, we explain and illustrate how the concepts presented above are implemented within each agent. Concretely, our focus will be on describing how policies are

[4] This constitutes a good barometer of the relevance of indicators like *reputation* and *recommendation*.

specified, adapted, combined and *evaluated* as they reflect the adaptive and socially-compliant aspects of our model.

5.1 Policies specification

We adopted logic programming to represent policies within the agent's knowledge base. Prolog is especially interesting as it is declarative and flexible enough which allow adaptation and combination of policies by adding, removing and updating trust criteria. A policy $\{tc_1, tc_1, \ldots, tc_n, \}$ is represented by a predicate structure as follows:

$$p_a^t([\langle \tau_{f_1}, \aleph_1, w_1 \rangle, \langle \tau_{f_2}, \aleph_2, w_2 \rangle, \ldots, \langle_{f_m}, \aleph_m, w_m \rangle]) \tag{2}$$

Each triplet $\langle \tau_{f_i}, \aleph_i, w_i \rangle$ constitute a trust criterion where, τ_{f_i} is its type and corresponds to a trust factor f_i within the ontology $\mathcal{T}$, $\aleph_i$ is a threshold value among possible values ($\aleph_i \in \Delta_{\tau_{f_i}}$) and $w_i \in \mathbb{N}^*$ represents the importance (i.e. weight) of tc_i in the final trust level computed when evaluating the policy. $a \in \mathcal{A}$ is the agent issuing the policy and t the handled request type $t = (o, r)$ such as $o \in O$ and $r \in \tau_{\mathcal{R}}$.

Example 6 (policies). Let $t = (access, object)$ be a request interaction pattern. The individual policy the agent *alice* uses to issue permissions for requests of type t is stated as follows: $p^t_{alice}([\langle identity, complete, 1\rangle, \langle statistics, 0.5, 2\rangle \langle reputation, 0.6, 1\rangle, \langle recommendation, 0.2, 2\rangle])$. While the community *Com2* to which *alice* belongs requires from its members to use the following policy: $p^t_{Com2}([\langle identity, ultimate, 3\rangle, \langle reputation, 0.5, 1\rangle]$

The examples presented above show how simply policies could be specified. But it also allowed us to realize how our greedy definition of trust policies could be complex when dealing with long policies. This makes writing a syntactically correct policy file difficult and motivate the use of XML. Thus, each agent manipulates internally (at runtime) the prolog-based policies while it maintains a cloned XML (cf. Fig 5) version for communication, adaptation, combination and persistence issues.

```
<?xml version="1.0" encoding="UTF-8"?>
<policy resource ="Object" operation = "Access">
  <tc type="identity" value="complete" weight="1" source="individual" />
  <tc type="statistics" value="0.5" weight="2" source="individual" />
  <tc type="recommendation" value="0.2" weight="2" source="individual" />
  <tc type="reputation" value="0.6" weight="1" source="individual" />
</policy>
```

Fig. 5. XML-based version of an individual policy

5.2 Policies adaptation (Meta-policies)

The meta-policies presented in this paper are specified by means of Jason plans. Each plan is represented as follows:

$$triggering\_event : context \leftarrow body \tag{3}$$

The mapping between Jason plans and ECA meta-policies is relatively trivial. The *triggering_event* denote the purpose for that plan (e.g adapting a particular policy) and is followed by a conjunction of belief literals qualifying a context (e.g. when the adaptation should be launched). The body of a plan is a sequence of basic actions the agent has to achieve when the plan is executed, if the context conditions hold. The key feature in using Jason plans lies in the possibility to execute legacy code through *internal actions*. In this work, we use this mechanism to perform adaptation operations presented in our trust model[5].

The result of the execution of each meta-policy affects the original policy by adding, setting, removing, restricting or relaxing trust criteria as follows:

$.add(\langle \tau_{f_i}, \aleph_i, w_i \rangle)$: this action adds the criteria $\langle \tau_{f_i}, \nu_i, w_1 tc_i \rangle$ in the current policy. If another *trust criteria* of the same type exists, the action $.set(\langle \tau_{f_i}, \aleph_i, w_i \rangle)$ is triggered otherwise.

$.del(\tau_{f_i})$: the actions removes all *trust criteria* of the type τ_i. τ_i is either a specific type as the identity or a general one such as proofs or indicators.

$.set(\langle \tau_{f_i}, \aleph_i, w_i \rangle)$: this action affects t_i and ν_i, w_1, respectively, to the strength and the weight of the *trust criteria* of type t_i in the current policy. If such *trust criteria* does not exists, an $.add(\langle \tau_{f_i}, \aleph_i, w_i \rangle)$ is triggered.

$.restrict(\tau_{f_i})$ and $.relax(\tau_{f_i})$: These two actions behave similarly. When no parameter is given, all *trust criteria* of the policy are affected by the action. Otherwise only the specified trust criteria type is affected. This action checks the *ontology* for a lower (resp. higher) value for the specified *trust criteria*. If such value exist, a set with the new value is called otherwise the weight of the *trust criteria* is lowered (resp. increased). If the weight becomes null, the *trust criteria* is removed from the policy.

Example 7 (Adaptation). The agent *alice* is particularly mindful about the number of agents viewing the ideas she is proposing (i.e. notes) and the risk associated to such access. Therefore, *alice* is automatically adapting her policy depending on such information. Let x be an access request on the document *maths* containing ideas on new mathematical model for incentives *alice* is developing, So $\tau_x = (access, object)$ and p^t_{alice} be the policy *alice* uses to handle such request. Meta-policies she is using for that purpose are as follow[6]:

$+adapt(p^t_{alice}, x, maths) : .count(member(\_, "Com2"), X)$
$\wedge hreshold(minUsers, Y) \wedge (X < Y) \leftarrow .relax(reputation).$

[5] For more details on AgentSpeak plans and internal actions concepts see [8, 23].

[6] .count is a standard internal action available in Jason that counts the number of beliefs that unify with *member*(_,"*Com*2") then X unifies with this quantity.

$+adapt(p^t_{alice}, x, maths) : sensitivity(maths, S)$
$\wedge threshold(sensitivity, T) \wedge (S > T) \leftarrow .restrict(reputation)$,
$.add(\langle recommendation, 0.1, 2\rangle.)$

The first meta-policy lowers the required reputation value depending on the users number while the second increases it and adds a recommendation criterion when the sensitivity of a resource is over a certain threshold. Both meta-policies are expressed using Jason plans.

5.3 Policies combination

Combining individual and collective policies illustrates the socially-enabled aspect of our trust model. The combination we opted for is relatively straightforward as the focus in this paper was to present the abstract vision of our trust model. The heuristic implemented in our proof-of-concept framework is an extension of the "Deny Override" combination algorithm used on XACML policy language [20]. The algorithm receives as input two policies and outputs a combined policy that is as restrictive as the most restrictive policy. In this paper we used this mechanism to guarantee that the resultant policy will integrate both individual and collective trust criteria (cf. [28] for more details on the used algorithm).

5.4 Policies evaluation

Let $p^t_a = \{tc_1, tc_2, \ldots, tc_m\}$ be the policy used by the agent $a \in \mathcal{A}$ to handle requests of type t. p^t_a evaluation represent a function $f(p^t_a)$ that evaluates the trust level l relative to an instance request r of type t. The trust level associated to the request r is computed by:

$$f(p^t_a) = \frac{\sum_{i=1}^{m} f(\tau_i, \aleph_i, w_i)}{\sum_{i=1}^{m} w_i} \tag{4}$$

Where $f(\tau_i, \aleph_i, w_i) \in [0, w_i]$ is the weighted evaluation of the constraint $\aleph_i$ satisfaction on the criteria of type τ_i. Each evaluated criteria returns one (1) or zero (0), respectively, whether the criteria is fulfilled or not. This result is then multiplied by the w_i and divided by the sum of all w_i in p^t_a.

Example 8. Let's continue in example 6 where *alice* was using the policy
$p^t_{alice}([\langle identity, complete, 1\rangle, \langle statistics, 0.5, 2\rangle, \langle reputation, 0.6, 1\rangle]$,
$\langle recommendation, 0.2, 2\rangle$ when handling requests of the type t.
And let $p^t_{Com2}([\langle identity, ultimate, 3\rangle, \langle reputation, 0.5, 1\rangle]$ be the collective policy she is using from her community $Com2$ to handle the same request type t.
And $\hat{p}^t_{alice}([\langle identity, ultimate, 3\rangle, \langle statistics, 0.5, 2\rangle, \langle recommendation, 0.2, 2\rangle$,
$\langle reputation, 0.6, 1\rangle]$ is the combined policy (cf. Section 5.3) that respects *alice* and her community's trust requirements.

If we consider the request from *David* who wants to get access to the document specifying one of the ideas that *alice* is working on. And assuming that *David* owns an *identity* credential that reveals to be *complete*, a *statistics* skills credentials with a

value of 0.5, his global *reputation* value is 0.5 and three other agents are willing to recommend him. When evaluating each of the above policy, *alice* ASC-TMS outputs the following results:

$$\textit{LocalStatisfaction(5, 6)} \tag{5}$$

$$\textit{CollectiveStatisfaction(1, 3)} \tag{6}$$

$$\textit{CombinedStatisfaction(4, 8)} \tag{7}$$

This represents to what extent the credentials provided by *David* and the information collected from the community satisfied *alice* policy (5), her community's policy (6) and the combined one (7). If *alice* considers her individual policy or the combined one she will accept the interaction otherwise she should refuse. In the first case she is considered as a deviant member while in the second case she is a compliant one[7].

6 Discussion and Concluding Remarks

We introduced in this paper the notion of social compliance for trust management within virtual communities. To realize it, we developed a rich, dynamic and socially-enabled trust model that mimics human trust. The proposed model preserves real world semantics of trust as individual requirements are adapted and merged with collective ones. The originality of this proposition lies in the joint use of flexible trust policies and adaptation meta-policies to capture and adapt trust requirement. The concept of adaptive policies has already been investigated by many scholars (e.g. [10]). However, in many cases, the proposed solution is built upon a known and finite universe of resources and entities, which is a hard constraint if we consider open environments like virtual communities. Further, the idea of combining different policies has also been studied (e.g. [20] and [16]). However, the focus of existing approaches on policies integration was principally on static security policies. The combination issues addressed in this paper could not be managed by former approaches as the policies to be combined are specified, adapted and managed on the fly.

The last decade has seen an increasing interest on trust that gave rise to numerous researches and many definitions (cf. [3]). Existing trust models can be categorized into two distinct families: *Hard Trust Models* where trust is established using credentials and trust policies (e.g. [6] and [17]) and *Soft Trust Models* where trust is built upon pairs experiences, recommendations and reputation (e.g. [11] [25] [14] and [18]. Full integration of soft and hard trust models has not been done yet. However, [7] and [19] paved the way recently for such perspective by developing new *Hybrid Trust Models*. This work is an additional right step toward that direction as it proposes an ontology-based *hybrid trust model* that enables agents to grasp the meaning of trust factors. We used the ontology $\mathcal{T}$ in the model (cf. Fig. 4) at different levels, which makes it central to our approach. (i) For policies specification where its concepts constitute trust

[7] The concept of compliance was intentionally simplified here to focus on adaptation features. However, in our simulation settings, different profiles has been defined and associated to each agent in order to investigate all effects [4] of the social influence theory (cf. Section 6).

conditions' types and their instances' values are used as acceptance thresholds; (ii) for policies adaptation where the ontology offers values ordering that agents can refer to when strengthening or weakening their policies; (iii) while combining individual and collective policies where it is used to align and merge trust criteria from both individual and collective policies. And finally, (iv) when evaluating policies where credentials and trust information are compared with respect required values.

The Adaptive and Socially-Compliant TMS is an on-going project that implements the rich, adaptive and socially-enabled trust model presented in this paper (cf. Section 4). We used the JaCaMo platform [15] to leverage the requirements discussed earlier while preserving the generality of the approach. JaCaMo is a newborn multi-agent oriented programming framework allowing the development of MAS by taking into account three different programming dimensions, namely the agent, environment, and organisation levels. The use of JaCaMo was motivated by its integration of three existing agent-based technologies (i) Jason [8], (ii) MOISE [13], and (iii) CArtAgO [24]. The Jason language was used to programme the agents, MOISE was used to specify communities as loosely structured flat organizations, and CArtAgO was used to programme distributed artifact-based environments where artifacts represent resources.

The next steps in our work would be to improve the "quite simplistic" heuristics used for policies adaption and combination by drawing some mechanisms from the prolific literature on trust and security policies (e.g. policies integration is a good track to follow). We also would like to extend this model in order to integrate other effects of the social influence theory [4]. Another interesting issue is to investigate how individual trust requirements (policies) could be integrated into collective ones. The idea would be to study how agents may influence / trigger adaptation or evolution of collective policies, and which mechanisms can they use to build collective policies from individual ones. Such feature is particularly interesting for decentralized and self-organized communities.

References

1. A. Abdul-rahman. The PGP Trust Model. *Architecture*, pages 1–6, 1997.
2. P. Anantharam, C.A. Henson, K. Thirunarayan, and A.P. Sheth. Trust Model for Semantic Sensor and Social Networks : A Preliminary Report. *Scenario*.
3. D. Artz and Y. Gil. A survey of trust in computer science and the semantic web. *Web Semant.*, 5(2):58–71, 2007.
4. S. E. Asch Opinions and social pressure. in *Scientific American, 193, 31-35., 1955*
5. M. Antikainen and H. Väätäjä. Collaboration in Open Innovation Communities - Do Users Want It?. In *Proceedings of The XIX ISPIM Conference Open Innovation Creating Products and Services through Collaboration*, 2008.
6. M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. *In Proceedings of IEEE Symposium on Security and Privacy*, 1996.
7. P. Bonatti, C. Duma, D. Olmedilla, and N. Shahmehri. An integration of reputation-based and policy-based trust management. In *In Proceedings of the Semantic Web Policy Workshop*, 2005.
8. R.H. Bordini and J. Hübner. Semantics for the jason variant of agentspeak. In *Proceeding of the 2010 conference on ECAI 2010*, pages 635–640, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.

9. T. Dimitrakos, B. Matthews, J. Bicarregui Towards security and trust management policies on the Web *CLRC Rutherford Appleton Laboratory, Oxfordshire, OX11 0QX, UK.* , 2001.
10. Ryutov, T. and Zhou, Li and Neuman, C. and Foukia, N. and Leithead, T. and Seamons, K. E. Adaptive Trust Negotiation and Access Control for Grids *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing* , 2005.
11. R. Falcone and C. Castelfranchi. *Social trust: a cognitive approach*, pages 55–90. Kluwer Academic Publishers, Norwell, MA, USA, 2001.
12. E. Gracia and J. Herrero Determinants of Social Integration in the Community: An Exploratory Analysis of Personal, Interpersonal and Situational Variables in Journal of Community Applied Social Psychology, 14, no. 1: 1-15, 2004
13. J. F. Hubner, J. S. Sichman, and O. Boissier. Developing organised multiagent systems using the moise+ model: programming issues at the system and agent levels. *Int. J. Agent-Oriented Softw. Eng.*, 1:370–395, 2007.
14. A. Jøsang. Foundations of security analysis and design iv. chapter Trust and reputation systems, pages 209–245. Springer-Verlag, Berlin, Heidelberg, 2007.
15. A. Ricci, J.F. Hubner, R. H Bordini, and O. Boissier. JaCaMo Project (http://http://jacamo.sourceforge.net/), 2010.
16. D.D. He and Y. Jian Security Policy Specification and Integration in Business Collaboration. *IEEE International Conference on Services Computing (SCC 2007), 2007.*
17. A. Herzberg, Y. Mass, J. Michaeli, Y. Ravid, and D. Naor. Access control meets public key infrastructure In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, 2000.
18. K. Krukow, N. Mogens and S. Vladimiro. A framework for concrete reputation-systems with applications to history-based access control. In *In Proc. of the 12th CCS*, pages 7–11, 2005.
19. A. J. Lee, T. Yu, and Y. Le Gall. Effective trust management through a hybrid logical and relational approach. pages 169–179, 2010
20. P. Mazzoleni, E. Bertino, B. Crispo and S. Sivasubramanian. XACML policy integration algorithms: not to be confused with XACML policy combination algorithms!. *Proceedings of the eleventh ACM symposium on Access control models and technologies, 2006.*
21. S. Milgram Obedience to Authority; An Experimental View. in *Harpercollins. ISBN 0-06-131983-X, 1971*
22. S. Moscovici, A. Mucchi-Faina, and A. Maass Minority Influence. Nelson-Hall, Chicago. C. Nemeth and J. Kwan, Dirigent Thinking and the Detection of Correct Solutions. in *Jounal of Appliaed Social Psychology 17 : 788-99.*
23. A. Rao. AgentSpeak (L): BDI Agents speak out in a logical computable language. (L), 1996.
24. A. Ricci, M. Piunti, L. D. Acay, R. H. Bordini, J. F. Hübner, and M. Dastani. Integrating heterogeneous agent programming platforms within artifact-based environments. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 1*, AAMAS '08, pages 225–232, Richland, SC, 2008.
25. J. Sabater and C. Sierra. Regret: reputation in gregarious societies. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, 2001.
26. K. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills, and L. Yu. Requirements for policy languages for trust negotiation. In *Proceedings of POLICY '02, USA, 2002.*
27. R. Yaich, O. Boissier, P. Jaillon, and G. Picard. Social-Compliance in Trust Management within Virtual Communities. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011*
28. R. Yaich, O. Boissier, P. Jaillon, and G. Picard. Trust Management within Virtual Communities: Adaptive and Socially-Compliant TMS. *FAYOL-EMSE Technical Report N 2011-700 -005*, 2011.