Patterns to Measure and Utilize Trust in Multi-Agent Systems

Gerrit Anders, Jan-Philipp Steghöfer, Florian Siefert, and Wolfgang Reif
Institute for Software & Systems Engineering
Augsburg University, Germany
E-Mail: {anders, steghoefer, siefert, reif}@informatik.uni-augsburg.de

Abstract—**This paper introduces three patterns to measure and utilize trust values in multi-agent systems (MAS). They allow a software engineer to incorporate mechanisms to gauge the benefit of interactions with other agents into the modeling and design process. In particular, patterns for trust from direct observation, reputation, and Trusted Communities are described. Each pattern contains the elements necessary to incorporate trust into a MAS and the collaboration required between the elements to make use of the measured values. The application of the patterns is demonstrated with an example from the domain of energy management systems.**

I. Introduction

Trust is the basis for cooperation within an organization or society. It influences an individual's thinking, behavior, and interactions. Thus, as in real life, trust is a key enabler for cooperation in software systems consisting of multiple interdependent entities, such as multi-agent systems (MAS) [1].

In MAS, autonomous proactive entities interact with each other in diverse ways to achieve a mutually favorable outcome. Just like in real life, interactions can be exploited for the benefit of only one party. One way to counter such detrimental behavior is to incorporate trust in the system and record the way an interaction partner behaved. If agents continuously interact with the same interaction partners, they can use their own interaction history (*direct trust*) to assess another agent. If interaction partners change often, a form of *reputation* [2] is beneficial which allows all agents to profit from the experiences of a few. In cases where it is desirable to form groups of trusted agents, *Trusted Communities* [3] can be used.

Trust and models to measure and use it are often described in an abstract mathematical fashion (e.g., [2], [4]). Existing trust management architectures and frameworks (for a survey, see [5]) in many cases give no guidance on their concrete implementation and are often focused on the algorithmic aspects of trust management (e.g., [6]) or subsume all trust-related functionality in an opaque trust manager [7]. Research on the software engineering aspects of trust is either concerned with the general possibility to use patterns to incorporate trust during the design process [8] or suggests specific mechanisms in specific domains (e.g., service-oriented computing [9]).

This paper synthesizes these existing works into *patterns* that cover various requirements in the very general setting of MAS. The patterns therefore support a software engineer in implementing these well-established concepts into a MAS. Different models of trust and reputation from literature can easily be adapted and implemented for use with them. This approach offers many advantages. The abstract trust management systems and trust models described in the literature are complex and difficult to implement, both due to their intricacy and the way they are presented. On the other hand, they offer functionality that is often not required in an application or are missing a crucial feature. With the patterns proposed in the following, trust management facilities can be tailored to an application and become part of both the design and implementation process. In many cases, this tight integration allows for quicker and more targeted development.

Trust is a complex concept that consists of multiple facets [10]. While the patterns themselves are generic and can be used to measure and use values of any facet, this paper focuses on *credibility*, i.e., an agent's willingness to participate in an interaction in a desirable manner. This view corresponds to the use of the term "trust" in literature on MAS. As agents are adaptive and can change their behavior, trust can change and deteriorate. Also, agents can behave differently towards different agents, making trust subjective and thus intransitive. Often, agents fulfill different functions in a system and their behavior depends on external factors. Therefore, an agent's role and the context of the interaction need to be regarded [11].

The paper is structured as follows: Sect. II introduces concepts that are important to all three patterns and the infrastructure necessary to store and retrieve trust data. Patterns for *direct trust*, *reputation*, and *Trusted Communities* are introduced in Sect. III. Their application to a MAS for energy management is demonstrated in Sect. IV. The paper concludes with a summary and an outlook.

II. Common Concepts and Infrastructure

From the body of literature and our own experience, a set of key concepts necessary to measure and use trust in MAS, depicted in Fig. 1, emerges. As mentioned before, trust in an `Agent` develops through repeated `Interactions` with it. If no interaction has taken place yet, initial trust [12] or reputation can be used. For each interaction, an agent can store its outcome as an `Experience`. Environmental factors as well as the roles of the interacting agents are captured in the `TrustContext`. This concept can comprise any factors that can change the behavior of an agent, such as the current time of day, the duration of the interaction, or the type of the interaction partner.

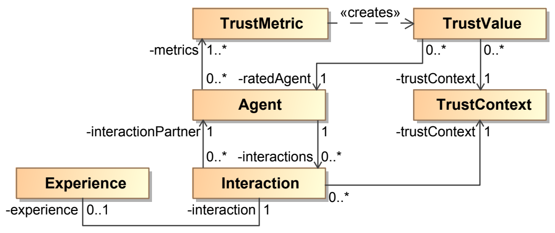


Fig. 1. Concepts for the measurement and use of trust in a MAS

When one or more interactions have been completed and an agent (the *initiator*) wants to determine the trustworthiness of another agent (the *trusted agent*)[1], a `TrustValue` for the trusted agent in a specific trust context is calculated using one of the `TrustMetrics`. Such metrics use the experiences with the trusted agent to determine a trust value and can be used to implement different trust models. In some cases, the required calculations are complex and it might be necessary to *transform* the data collected in the experiences, e.g., to statistical data. These intermediate data can then be *interpreted* to yield a trust value. Such a transformation of the measured data into intermediate data and interpretation to a trust value can be aided by a corresponding infrastructure in a MAS that allows to store experiences, provides interfaces for metrics and trust contexts, and allows to derive trust values [13].

Depending on the concrete requirements of the modeled system, the metrics can also deal with some of the aspects of trust mentioned in the introduction. It might, e.g., be beneficial to discount older experiences and base the calculation of the trust value only on recent ones. Descriptions of different metrics that incorporate this aspect can be found in [14].

III. Patterns for Direct Trust, Reputation, and Implicit Trusted Communities

In the following, we propose three patterns that can be used to incorporate three different mechanisms to measure and utilize trust into MAS. The first pattern (see Sect. III-A) deals with trust through direct observation (from now on abbreviated as *direct trust*). It is applicable in situations where the initiator had previous direct interactions with a trusted agent. The second pattern (see Sect. III-B) introduces *reputation* into a system, i.e., the possibility to learn how other agents rate a trusted agent's trustworthiness on the basis of their experiences with it. The third pattern (see Sect. III-C) shows a way to establish *Implicit Trusted Communities* (ITCs) within a MAS. These are groups of interacting agents that build upon experiences from direct interactions and reputation. To describe these patterns, we use a notation that is based on the notation of Gamma et al. [15]. For each pattern, we give a short definition of relevant terms in the *Motivation*.

A. Direct Trust Pattern

1) Intent: Measure and obtain an agent's personal trust in an interaction partner.

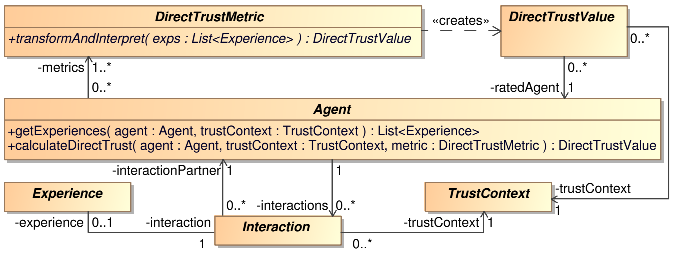
[1]Both *initiator* and *trusted agent* will be used in the paper, even though an initiator can decide not to trust a trusted agent after evaluating its trust value.

2) Motivation: Direct trust is defined as the trust of an initiator a_1 in a trusted agent a_2, based on a_1's personal experiences with a_2 by analyzing a_2's behavior in direct interactions. Since the behavior of a_2 may vary with the context in which interactions take place, direct trust depends on a trust context. If the agents participate in an electronic market, a_2 may, e.g., behave differently as a seller and as a buyer. Depending on the system, even the product that is traded may influence a_2's behavior. Because experiences that were made in the past may lose their relevance over time, direct trust is an aggregated representation of a subset of all experiences with an interaction partner in a specific trust context.

In many situations, the knowledge of an agent's direct trust in its interaction partners is relevant to assess the potential value or risk of an interaction with one of these agents. Agents knowing how their interaction partners usually behave in interactions can prefer more trustworthy alternatives or take measures that ensure the quality of an interaction's outcome.

3) Applicability: The Direct Trust pattern is applicable when the knowledge of an agent's direct trust in its interaction partners is essential, the same agents interact with each other frequently, and it is valid to assume that the prior behavior of an interaction partner is indicative of its future behavior.

4) Structure and Participants: Fig. 2 shows the structure of the pattern. Initiator and trusted agents are instances of `Agent`. The initiator wants to evaluate its `DirectTrustValue` for a trusted agent (`ratedAgent`) in a specific `trustContext`. The resulting trust value depends on `Experiences` the initiator made with the trusted agent in the course of `Interactions` initiated by the initiator in which the trusted agent acted as `interactionPartner`. To evaluate experiences, the initiator uses one of several `DirectTrustMetrics` which transform and interpret the experiences with `ratedAgent` in `trustContext` into a `DirectTrustValue`.



DirectTrustMetric
+transformAndInterpret(exps : List<Experience>) : DirectTrustValue
«creates»
DirectTrustValue
0..*
-metrics
1..*
0..*
0..*
-ratedAgent
1
Agent
+getExperiences(agent : Agent, trustContext : TrustContext) : List<Experience>
+calculateDirectTrust(agent : Agent, trustContext : TrustContext, metric : DirectTrustMetric) : DirectTrustValue
-interactionPartner
1
1
Experience
0..*
-interactions
0..*
TrustContext
-trustContext
1
-experience
0..1
-interaction
Interaction
-trustContext
1
1
0..*

Fig. 2. Direct Trust pattern

5) Collaborations: Fig. 3 illustrates the procedure for determining an agent's trustworthiness. Whenever an initiator `a1` wants to obtain its direct trust value `dt` in a trusted agent `a2` for a trust context `c`, it calls the method `calculateDirectTrust(a2,c,m)`, where `m` is the metric that should be used to generate the direct trust value `dt`. Thereupon, `a1` gathers all experiences made with `a2` in `c` in a list `expList` by calling `getExperiences(a2,c)`. Afterwards, `a1` calls `transformAndInterpret(expList)` on `m`. This initiates the transformation and interpretation of experiences into `dt`, which is returned to `a1` afterwards.

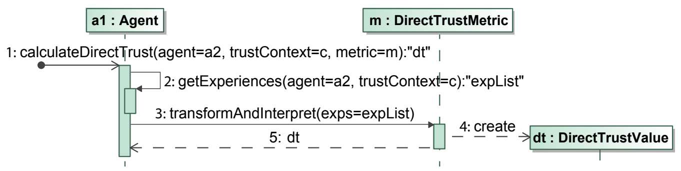

a1 : Agent
m : DirectTrustMetric
1: calculateDirectTrust(agent=a2, trustContext=c, metric=m):"dt"
2: getExperiences(agent=a2, trustContext=c):"expList"
3: transformAndInterpret(exps=expList)
4: create
dt : DirectTrustValue
5: dt

Fig. 3. Direct Trust pattern: interactions

6) Consequences: It is important to identify a suitable granularity of the trust context. If the definition of the trust context is too fine- or coarse-grained, it is difficult to derive representative direct trust values.

7) Implementation: Different trust metrics can be used to implement trust models proposed in literature. Further, different metrics can be used to obtain trust values for facets like, e.g., reliability, by regarding appropriate experiences, or to evaluate experiences from different perspectives, e.g., by using different appraisal criteria and analysis methods.

In systems in which the trust context is irrelevant, `TrustContext`, all associations to and from it, as well as corresponding parameters in method signatures can be omitted.

B. Reputation Pattern

1) Intent: Find out how other agents appraise the trustworthiness of an agent.

2) Motivation: Reputation is a measure of how other agents rate the trustworthiness of an agent in a given trust context. Thus, reputation is an aggregation of various experiences of different agents with an agent acting in a specific trust context.

In some situations, an initiator has not made any experience with a potential interaction partner yet or existing experiences are not sufficient to derive a meaningful direct trust value. To assess the trustworthiness, the initiator can then either use an initial trust value (i.e., a default trust value) or retrieve a reputation value based on the experiences of *other* agents with the trusted agent. Although trust relations are in general not transitive, reputation thereby allows to gauge other agents much better than initial trust. In addition, there are situations in which certain decisions should not be made on the basis of one-sided experiences or can be improved by utilizing a combination of direct trust and reputation.

3) Applicability: The Reputation pattern is applicable when interaction partners often change, untrustworthy agents have a detrimental effect on the system, and an agent's prior behavior is indicative of its behavior in interactions with other agents.

4) Structure and Participants: Fig. 4 shows the pattern that can be used to integrate a reputation mechanism into a system of interacting agents. It is similar to the Direct Trust pattern (see Fig. 2). Each agent has one or more `RepuationMetrics` defining how experiences should be aggregated into a `ReputationValue`. Furthermore, each reputation belongs to a specific trusted agent (`ratedAgent`) and `trustContext`. In order to retrieve the reputation of an agent in a specific trust context, agents have access to a component, the so-called `ReputationProvider`, that provides an interface for querying the reputation of arbitrary agents in the system.

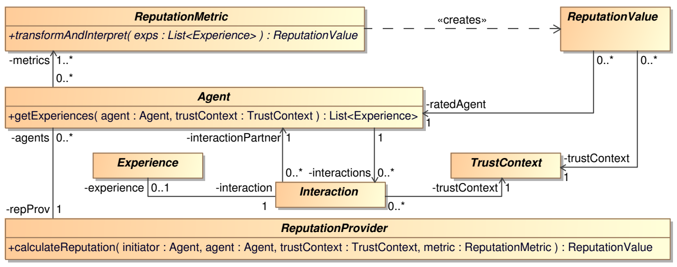

ReputationMetric
+transformAndInterpret(exps : List<Experience>) : ReputationValue
«creates»
ReputationValue
-metrics 1..*
0..*
0..*
0..*
Agent
+getExperiences(agent : Agent, trustContext : TrustContext) : List<Experience>
-ratedAgent
1
-agents 0..*
-interactionPartner 1
1
Experience
TrustContext
-trustContext
1
0..* -interactions 0..*
-experience 0..1
-interaction
1
Interaction
-trustContext 1
0..*
-repProv 1
ReputationProvider
+calculateReputation(initiator : Agent, agent : Agent, trustContext : TrustContext, metric : ReputationMetric) : ReputationValue

Fig. 4. Reputation pattern (poll)

5) Collaborations: As shown in Fig. 5, an initiator `i` can query the reputation `rep` of a trusted agent `a` in a specific trust context `c` by calling `calculateReputation(i,a,c,m)` for a reputation metric `m` on the reputation provider `repProv`. First, the reputation provider creates an empty list `expList` as container to store the experiences of agents with `a` in `c`. Then, it asks each agent `ag` in the list of known `agents` (except initiator `i` because its experiences with `a` should not influence `a`'s reputation) for experiences with `a` in `c` by calling `getExperiences(a,c)`, whereupon the result `tempExps` is added to `expList`. The accumulated experiences are processed and transformed into `rep` by calling `transformAndInterpret(expList)` on `m`. As the reputation provider collects the experiences of the agents on demand, this form of collaboration is called *poll variant*. A version of the pattern where experiences are *pushed* to the reputation provider is described in Sect. III-B7.

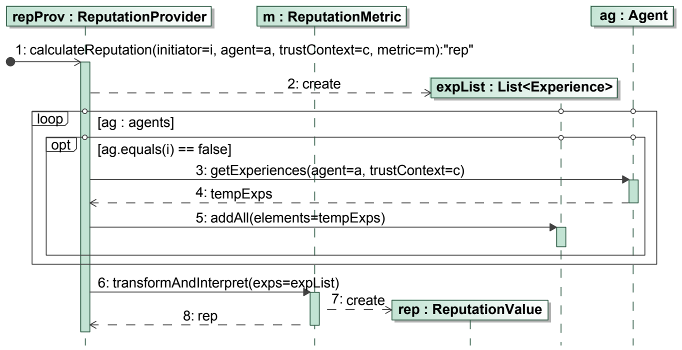

repProv : ReputationProvider
m : ReputationMetric
ag : Agent
1: calculateReputation(initiator=i, agent=a, trustContext=c, metric=m):"rep"
2: create
expList : List<Experience>
loop
[ag : agents]
opt
[ag.equals(i) == false]
3: getExperiences(agent=a, trustContext=c)
4: tempExps
5: addAll(elements=tempExps)
6: transformAndInterpret(exps=expList)
7: create
rep : ReputationValue
8: rep

Fig. 5. Reputation pattern (poll): interactions

6) Consequences: Because of the subjective and thus non-transitive nature of trust, reputation cannot replace direct trust in systems in which personal experiences are important to identify trustworthy agents. However, since the Reputation and the Direct Trust pattern share common concepts, it is is easy to complement an implementation of the Direct Trust pattern with an implementation of the Reputation pattern or vice versa.

7) Implementation: In some cases it is not desirable that the reputation provider has full knowledge about all agents and queries them when required. Instead, the agents in the system can provide their experiences in advance, making the process of gathering experiences unnecessary. This form of collaboration is called *push variant* as the agents push their experiences to the reputation provider.

The *push variant* differs from the *poll variant* in the following points: every time an agent `o` registers an experience `e` with an agent `a` in a trust context `c`, it calls `updateDataBase(o,a,c,e)` on the reputation provider. Hence, the reputation provider manages a list of experiences `exps`. Whenever an initiator `i` requests the reputation `rep` for `a` and trust context `c`, the list of relevant experiences `expList` is compiled by filtering the existing list of experiences `exps` by calling `filterExperiences(i,a,c)`. The resulting list `expList` only contains experiences associated with `a` and trust context `c`; experiences of `i` are not included. Apart from that, the procedure does not change.

The pattern description assumes that each agent has access to a central (perhaps distributed) reputation provider acquainted with and available to all agents in the system. It is, however, possible to have multiple reputation providers that only know a subset of agents.

A different variant of this pattern can use the agents' `DirectTrustValues` instead of their experiences to calculate the `ReputationValue`.

As with the Direct Trust pattern, the `TrustContext` concept, all associations to and from it, as well as corresponding parameters in the method signatures can be omitted.

C. Implicit Trusted Communities Pattern

1) Intent: Form groups of trustworthy interaction partners.

2) Motivation: An *Implicit Trusted Community* (ITC) [3] is a dynamic organization of agents that builds upon trust relations. Each agent forms its own ITCs by applying a *ranking* to all agents known to it and selecting only the best performers ranked higher than a given threshold for interactions. This implies that each agent can be a member of multiple ITCs and the communities form only from the agents' local decisions. As a consequence, an agent is not aware of being a member of an ITC and it is not mandatory that an ITC includes agents that mutually trust each other.

ITCs evolve in systems in which many interactions take place and that feature a large set of possible interaction partners. The main idea is that by ranking these agents according to direct trust and reputation, each agent eventually chooses interaction partners that are expected to be particularly trustworthy and suitable. As a result, agents exclude untrustworthy agents from their ITCs. A system using ITCs can thus increase its robustness and improve its efficiency as agents communicating within an ITC can abstain from certain preventive security measures or result checking mechanisms.

The agents' ranking may depend on additional criteria such as an agent's performance or current workload. Once an interaction is complete, the initiator gains experience with its interaction partner. This influences the partner's trustworthiness and in turn the composition of the initiator's ITCs.

3) Applicability: Systems in which it is beneficial to interact and cooperate with trustworthy agents profit from the application of the ITCs pattern. In particular, such systems are characterized by various participants and a high number of interactions. Furthermore, the Direct Trust and Reputation pattern have to be applicable (see Sect. III-A and Sect. III-B).

4) Structure and Participants: Since the selection of interaction partners depends on direct trust and reputation values, the ITCs pattern (see Fig. 6) makes use of several concepts that are already known from the Direct Trust (see Sect. III-A) and Reputation pattern (see Sect. III-B). This includes the concepts `DirectTrustValue` and `DirectTrustMetric` as well as the concepts `ReputationValue`, `ReputationMetric`, and `ReputationProvider`. The set of possible interaction partners is restricted to the set of agents the interaction's initiator is acquainted with (`knownAgents`). In order to decide which agent should become an interaction partner, the initiator creates a ranking of all agents contained in `knownAgents`. This is done by utilizing one of several `RankingMetrics`, each drawing upon one or more metrics for evaluating the direct trust (`dtMetrics`) in and reputation (`repMetrics`) of an agent.

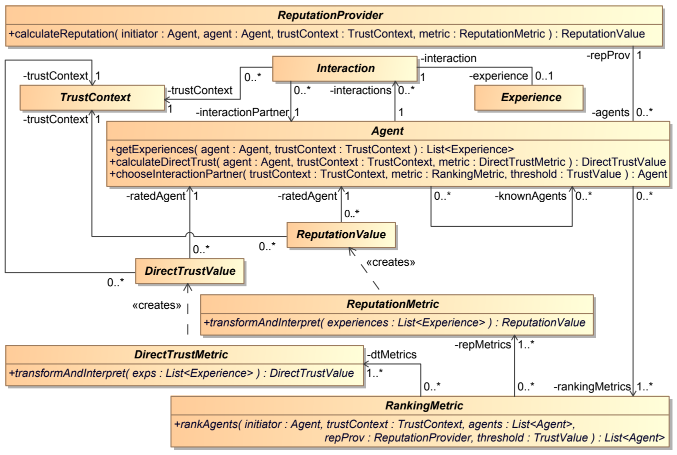
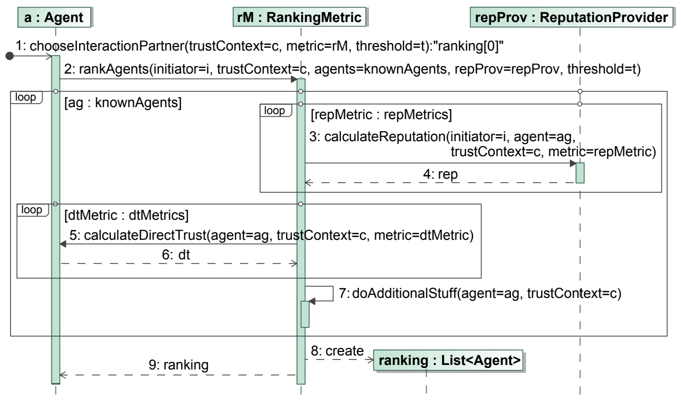

ReputationProvider
+calculateReputation(initiator : Agent, agent : Agent, trustContext : TrustContext, metric : ReputationMetric) : ReputationValue
-repProv 1
-interaction
Interaction
-experience 0..1
Experience
-trustContext 1
TrustContext
-trustContext
0..*
1
0..*
-interactions 0..*
1
-interactionPartner 1
-trustContext 1
-agents 0..*
Agent
+getExperiences(agent : Agent, trustContext : TrustContext) : List<Experience>
+calculateDirectTrust(agent : Agent, trustContext : TrustContext, metric : DirectTrustMetric) : DirectTrustValue
+chooseInteractionPartner(trustContext : TrustContext, metric : RankingMetric, threshold : TrustValue) : Agent
-ratedAgent 1
-ratedAgent 1
0..*
0..*
-knownAgents 0..*
0..*
ReputationValue
0..*
0..*
DirectTrustValue
0..*
«creates»
«creates»
ReputationMetric
+transformAndInterpret(experiences : List<Experience>) : ReputationValue
-repMetrics 1..*
DirectTrustMetric
+transformAndInterpret(exps : List<Experience>) : DirectTrustValue
-dtMetrics
1..*
0..*
0..*
-rankingMetrics 1..*
RankingMetric
+rankAgents(initiator : Agent, trustContext : TrustContext, agents : List<Agent>,
repProv : ReputationProvider, threshold : TrustValue) : List<Agent>

Fig. 6. Implicit Trusted Communities pattern

5) Collaborations: As depicted in Fig. 7, whenever initiator `i` is looking for a trustworthy partner for an interaction which is to take place in a specific trust context `c`, it calls the method `chooseInteractionPartner(c,rM,t)`, where `rM` is the ranking metric used to determine the interaction partner and `t` is a threshold specifying a lower bound for the `TrustValue` of ITC members. Subsequently, the method `rankAgents(i,c,knownAgents,repProv,t)` is called on `rM` with reputation provider `repProv`. The method returns a list of agents in `knownAgents`, sorted according to their ranking. To create this ranking, for each agent `ag` in `knownAgents`, `rM` queries `ag`'s reputation by utilizing `repProv` for each reputation metric `repMetric` in `repMetrics` and requests `i`'s direct trust in `ag` for each direct trust metric `dtMetric` in `dtMetrics`. The call `doAdditionalStuff(ag,c)` is a placeholder that represents all other calls and computations of a specific application to further influence `ag`'s ranking.

Agents ranked higher than the given threshold `t` define an ITC. `chooseInteractionPartner(c,rM)` finally returns the first element of the `ranking` if it is ranked higher than `t`. This procedure is repeated for every new interaction.

The method `doAdditionalStuff(ag,c)` may also introduce some randomization into the ranking. Such a measure ensures that, from time to time, less trustworthy agents are selected as interaction partners. This is useful as these agents get a chance to change their behavior, thus becoming trustworthy and a member of an ITC again.



a : Agent
rM : RankingMetric
repProv : ReputationProvider
1: chooseInteractionPartner(trustContext=c, metric=rM, threshold=t):"ranking[0]"
2: rankAgents(initiator=i, trustContext=c, agents=knownAgents, repProv=repProv, threshold=t)
loop
[ag : knownAgents]
loop
[repMetric : repMetrics]
3: calculateReputation(initiator=i, agent=ag, trustContext=c, metric=repMetric)
4: rep
loop
[dtMetric : dtMetrics]
5: calculateDirectTrust(agent=ag, trustContext=c, metric=dtMetric)
6: dt
7: doAdditionalStuff(agent=ag, trustContext=c)
8: create
ranking : List<Agent>
9: ranking

Fig. 7. Implicit Trusted Communities pattern: interactions

6) Consequences: In contrast to the Direct Trust and Reputation patterns, the ITC pattern not only evaluates an agent's trustworthiness but includes selecting interaction partners.

As `chooseInteractionPartner(...)` is called for a specific trust context, ITCs evolve with respect to a trust context. In general, ITCs do not necessarily contain agents that mutually trust each other. In case it is crucial or beneficial that interaction partners trust the interaction's initiator, implementations of the method `rankAgents(...)` of the initiator's `RankingMetrics` have to consider this requirement in order to form appropriate ITCs.

7) Implementation: If the trust context is irrelevant, `TrustContext`, associations to and from it, and corresponding parameters in the method signatures can be omitted.

IV. Applying the Patterns

This section shows how to apply the patterns for Direct Trust and for ITCs to an energy management system based on Autonomous Virtual Power Plants (AVPPs) [16]. One of the purposes of AVPPs is to subsume small, individual power plants (e.g., solar panels or domestic combined heat and power units) to allow them to participate in the energy market. If such an AVPP also contains consumers, it can either offer energy in case it produces more than is consumed locally or buy energy from other AVPPs otherwise. In this context, an interaction is a contract between two AVPPs. Compliance to a contract can be evaluated by comparing the actual consumption or production of energy with the stipulated amount of energy. If an AVPP fails to produce or consume the amount of energy agreed upon, its trust value will decrease and it will be more difficult or expensive to buy or sell energy at the market at a later time.

[17] introduces an agent-based architecture for energy management applications the patterns were applied to. Our implementation includes energy producers and consumers, an

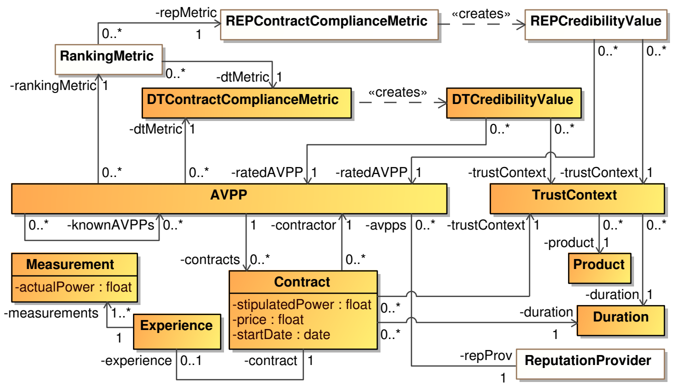


Fig. 8. Instantiation of the Direct Trust (relevant concepts are highlighted in dark color) and Implicit Trusted Communities (all concepts relevant) pattern for an energy management application

energy market, and other relevant concepts that measure and use trust values to make appropriate decisions.

In Sect. IV-A, the Direct Trust pattern is used to gauge trust between AVPPs if they negotiate with each other directly while Sect. IV-B demonstrates the use of ITCs in case it is beneficial for a group of AVPPs to negotiate with each other repeatedly. Sect. IV-C hints at some important implementation issues.

A. Applying the Direct Trust Pattern

If the number of AVPPs that take part in the market is small, it is likely that an AVPP makes contracts with AVPPs it interacted with in the past. In such a case, an AVPP that wants to negotiate a contract benefits from the consideration of its direct trust in former interaction partners (e.g., it can avoid contracts with AVPPs which failed to fulfill contracts). As a result, the robustness of the entire system can be increased which is of particular interest in safety-critical systems.

The instantiation of the Direct Trust pattern (see Sect. III-A) is depicted by the concepts highlighted in dark color in Fig. 8. As already mentioned, the concept `AVPP` maps onto `Agent` and the concept `Contract` onto `Interaction`. A `Contract` consists of the `stipulatedPower` that should be continuously generated or consumed by the `contractor` at the agreed upon `price` starting at `startDate` for a specified amount of time (`duration`). During the validity of the contract, the AVPP makes `Experience` with the contractor gained through periodic `Measurements` that hold the average value of the `actualPower` that has been generated or consumed by the `contractor` within a given time period to enable statistical evaluations. There are multiple measurements for each contract because the interaction, i.e., the fulfillment of the contract, takes a long time. Since the quality of the fulfillment of a contract may depend on the `Product` (energy generation or consumption) and the `Duration` of the contract, these two concepts form the `TrustContext`. For example, an AVPP could be trustworthy in situations in which it has to generate power for a short period of time, but untrustworthy whenever it has to consume energy. The direct trust value for a specific AVPP is determined by a `DTContractComplianceMetric` which statistically evaluates the experience with this AVPP by, e.g., calculating the deviation of `actualPower` values from `stipulatedPower` values. In this example, the direct trust value is thus a credibility value (`DTCredibilityValue`).

B. Applying the Implicit Trusted Communities Pattern

The possibility to be dependent on AVPPs with which no experiences have been made increases with the number of AVPPs. As the number of energy producers and consumers is steadily increasing, this is a realistic scenario for energy management systems. Hence, ITCs and thus the combination of direct trust with reputation becomes crucial to select trustworthy interaction partners.

As Fig. 8 shows, the concepts introduced in the previous section are completed by a `ReputationProvider` and a reputation metric (`REPContractComplianceMetric`) that determines the trustworthiness of an AVPP by transforming and interpreting the experiences of other AVPPs with a potential `contractor` into a reputation value (`REPCredibilityValue`). Again, the reputation value represents the AVPP's credibility. The `RankingMetric` creates a ranking based on direct trust and reputation, and could be further influenced by the price of energy production or consumption. Consequently, whenever a suitable and credible `contractor` needs to be found, the corresponding AVPP (i.e., the initiator) puts a request to its `RankingMetric`.

C. Lessons Learned

The implementation of the patterns in the energy management system showed the importance of the right granularity of the trust context to keep experiences comparable. In addition, the possibility of agents to cheat by lying about other's trust value or to collude has become evident. These issues will be addressed in future work.

V. Conclusion and Future Work

In this paper, we presented three patterns that assist a software engineer in enhancing a multi-agent system (MAS) with mechanisms to measure and utilize trust and thus to enable efficient cooperation between agents. All patterns share that trust is derived from experiences made with an agent in the course of previous interactions and that it depends on the context in which an interaction takes place. First, we introduced the Direct Trust pattern to determine an agent's trust into its interaction partners. For situations in which it is worthwhile to make use of other agents' trust in another agent, we proposed the Reputation pattern. The Implicit Trusted Community pattern is a guidance for creating implicit agent organizations that build upon trust relations. We also showed how to apply these patterns to an existing architecture.

Future work includes patterns for establishing trust in MAS with respect to reliability and security. The latter encompasses patterns for authentication and authorization in MAS as well as security measures to avoid false reporting and collusion. Furthermore, the integration of the patterns with the MAS infrastructure mentioned in Sect. II will be pursued. The implementation of different trust models as metrics is also currently considered.

Acknowledgment

This research is partly sponsored by the German Research Foundation (DFG) in the project "OC-Trust" (FOR 1085).

References

[1] S. Ramchurn, D. Huynh, and N. Jennings, "Trust in Multi-Agent Systems," *The Knowledge Engineering Review*, vol. 19, no. 01, pp. 1–25, 2005.

[2] L. Mui, M. Mohtashemi, and A. Halberstadt, "A Computational Model of Trust and Reputation," in *Proc. of the 35th Hawaii International Conference on System Sciences*, 2002, pp. 188–196.

[3] Y. Bernard, L. Klejnowski, J. Hähner, and C. Müller-Schloer, "Towards Trust in Desktop Grid Systems," in *Cluster Computing and the Grid, IEEE International Symposium on*. Los Alamitos, CA: IEEE Computer Society, 2010, pp. 637–642.

[4] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The Eigentrust algorithm for reputation management in P2P networks," in *Proceedings of the 12th international conference on World Wide Web*, ser. WWW '03. New York, NY, USA: ACM, 2003, pp. 640–651.

[5] G. Suryanarayana and R. N. Taylor, "A survey of trust management and resource discovery technologies in peer-to-peer applications," Institute for Software Research, University of California, Irvine, Tech. Rep. UCI-ISR-04-6, 2004.

[6] M. Kinateder and K. Rothermel, "Architecture and algorithms for a distributed reputation system," in *iTrust 2003, Heraklion, Greece*, ser. Lecture Notes in Computer Science, P. Nixon and S. Terzis, Eds., vol. 2692. Springer, 2003, pp. 1–16.

[7] C. Lin, V. Varadharajan, Y. Wang, and V. Pruthi, "Enhancing grid security with trust management," in *IEEE International Conference on Services Computing (SCC 2004), Shanghai, China*. IEEE Computer Society, 2004, pp. 303–310.

[8] B. Biel, T. Grill, and V. Gruhn, "Patterns of Trust in Ubiquitous Environments," in *Proc. of the 6th International Conference on Advances in Mobile Computing and Multimedia*, ser. MoMM '08. New York, NY, USA: ACM, 2008, pp. 391–396.

[9] P. Li, M. Xiangxu, S. Zhiqi, and Y. Han, "A Reputation Pattern for Service Oriented Computing," in *Proc. of the 7th International Conference on Information, Communications and Signal Processing*, 2009, pp. 1–5.

[10] J.-P. Steghöfer, R. Kiefhaber, K. Leichtenstern, Y. Bernard, L. Klejnowski, W. Reif, T. Ungerer, E. André, J. Hähner, and C. Müller-Schloer, "Trustworthy Organic Computing Systems: Challenges and Perspectives," in *Proc. of the 7th International Conference on Autonomic and Trusted Computing (ATC 2010)*. Springer, Oct. 2010.

[11] Y. Wang and J. Vassileva, "Trust and Reputation Model in Peer-to-Peer Networks," in *Proc. of the 3rd International Conference on Peer-to-Peer Computing*, 2003.

[12] D. McKnight, L. Cummings, and N. Chervany, "Initial Trust Formation in New Organizational Relationships," *The Academy of Management Review*, vol. 23, no. 3, pp. 473–490, 1998.

[13] R. Kiefhaber, F. Siefert, G. Anders, T. Ungerer, and W. Reif, "The Trust-Enabling Middleware: Introduction and Application," Universitätsbibliothek der Universität Augsburg, Tech. Rep. 2011-10, 2011.

[14] R. Kiefhaber, B. Satzger, J. Schmitt, M. Roth, and T. Ungerer, "Trust Measurement Methods in Organic Computing Systems by Direct Observation," in *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 105–111.

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Addison Wesley, 1995.

[16] G. Anders, F. Siefert, J.-P. Steghöfer, H. Seebach, F. Nafz, and W. Reif, "Structuring and Controlling Distributed Power Sources by Autonomous Virtual Power Plants," in *Proc. of the Power & Energy Student Summit 2010 (PESS 2010))*, October 2010, pp. 40–42.

[17] G. Anders, L. Klejnowski, J.-P. Steghöfer, F. Siefert, and W. Reif, "Reference Architectures for Trustworthy Energy Management and Desktop Grid Computing Applications," Universitätsbibliothek der Universität Augsburg, Tech. Rep. 2011-11, 2011.