An effective implementation of norms in trust-aware open self-organising systems

Jan-Philipp Steghöfer, Gerrit Anders, Jan Kantert, Christian Müller-Schloer, Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Steghöfer, Jan-Philipp, Gerrit Anders, Jan Kantert, Christian Müller-Schloer, and Wolfgang Reif. 2014. "An effective implementation of norms in trust-aware open self-organising systems." In *2014 IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems Workshops, 8-12 September 2014, London, UK*, edited by Alexander Artikis, Jan-Philipp Steghöfer, and Didac Busquets, 76–77. Piscataway, NJ: IEEE. https://doi.org/10.1109/sasow.2014.34.

An Effective Implementation of Norms in Trust-Aware Open Self-Organising Systems

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

Jan Kantert, Christian Müller-Schloer
Institut für Systems Engineering – SRA
Leibniz Universität Hannover, Germany
Email: {kantert, cms}@sra.uni-hannover.de

***Abstract*—We discuss the implementation of a normative system in an open self-organising system, including an OCL-based format for norms we settled on, the design of the feedback loops for their observation and adaptation, as well as a corresponding software architecture. These elements allow designers to quickly integrate a normative sub-system in a MAS and to define norms based on existing design concepts.**

I. Norms in Open Self-Organising Systems

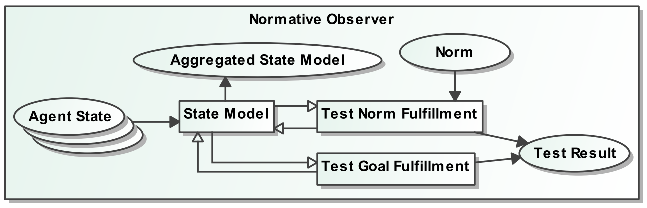
Norms play two distinct roles in open self-organising systems: 1) They serve as the basis for long-term adaptations, e.g., when business rules or legislation changes in a distributed power management system [6]; 2) they guide the agents' decisions when the system goals are violated and thus serve for short-term incident mitigation, e.g., when the trust relation between agents in a desktop grid systems deteriorates [1]. In both cases, norms influence the behaviour of the underlying agents indirectly, thus avoiding regimentation, but instead changing incentives in a way that causes rational agents that strive to maximise their own utility to adapt their behaviour accordingly [2]. Essentially, norms have to contribute to reaching a *goal* of the system, be it a business rule or the maximisation of cooperation.

We distinguish *normative agents* and *norm-aware agents*. The former type of agent is capable of creating and adapting norms, as well as observing the norm-aware agents and sanctioning norm-violations. These duties make it necessary to observe whether the current set of norms causes the behaviour of the system to contribute to the fulfilment of the current system goal. The latter type of agent uses norms in its decision processes. Since norm compliance is usually associated with an incentive (or its violation with a sanction), the agent can use this information to select the action with the highest overall incentive, incorporating its personal utility function. Compared to existing work (e.g., [4]), our implementation uses an established notation and tool-chain and exploits synergies with the system design.

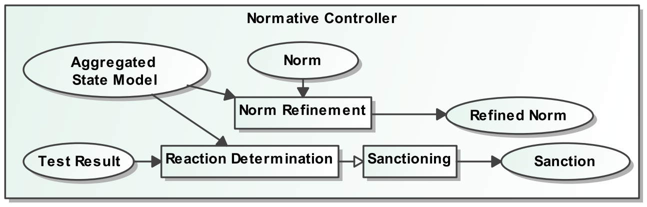
II. Observer/Controller for Normative Agents

The normative feedback loop is embodied by a normative observer (cf. Fig. 1a) and a normative controller (cf. Fig. 1b). They extend the standard Observer/Controller (O/C) architecture [5] with additional components for the evaluation and creation of norms. The normative observer generates a state model from the individual states of the agents and tests the aggregated state for norm violations that are captured as test results. In addition, it tests whether the system goal that is supposed to be achieved by the application of norms, is fulfilled. The normative controller, on the other hand, is able to refine norms based on analysis of the aggregated state model and detect and sanction norm violations based on the test results.

This research is partly sponsored by the research unit *OC-Trust* (FOR 1085) of the German Research Foundation.



(a) Conceptual outline of a normative observer



(b) Conceptual outline of a normative controller

Fig. 1. Normative Observer and Controller

The architecture outlined conceptually above can be realised as shown in the class diagram in Fig. 2. One or more `NormAware-Agents` are observed by a `NormMetricObserver` and influenced by a `NormativeController`. The `NormativeAgent` serves as the container for one or several `NormMetricObservers` that observe different sub-systems or norms. For this purpose, each `NormAwareAgent` provides a `StateModel` that can be evaluated by the `NormMetricObserver` by using a specialised `NormMetric`. This allows to detect norm violations that can only be observed indirectly, e.g., by the occurrence of certain interaction patterns. A `NormMetricObserver` can combine several `StateModels` as provided by `NormAwareAgents` into an `aggregatedStateModel`. This model can also be used in the evaluations of metrics or can be forwarded to a super-ordinate `NormativeAgent` if a hierarchical system structure is present.

By using a publish/subscribe infrastructure, an `Observer` can be informed of changes in the `StateModel` without constant polling. Likewise, the `Controller` is informed only when necessary. This approach corresponds to the "Abstract Observation Model" in [3].

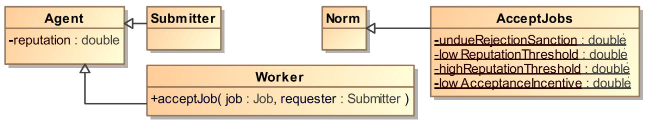

Agent
-reputation : double
Submitter
Norm
AcceptJobs
-undueRejectionSanction : double
-lowReputationThreshold : double
-highReputationThreshold : double
-lowAcceptanceIncentive : double
Worker
+acceptJob(job : Job, requester : Submitter)

Fig. 3. Exemplary class diagram introducing the concepts used in Fig. 4.

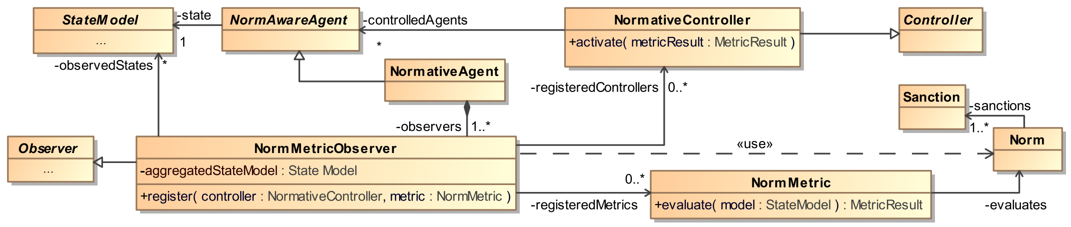


Fig. 2. UML class diagram showing the concepts and relations for normative and norm-aware agents and the corresponding O/C infrastructure.

III. An OCL-based Syntax for Norms

We use the norm structure outlined in [7] that specifies the elements listed below. An important difference to [7] is, however, that we use a postcondition instead of an action. This extends the expressiveness and reduces the interface that has to be specified as the agents are free in the way they bring about the condition. Our norms can always be interpreted as obligations in the deontic sense.

Target/Role: The agent or role the norm applies to. Can be a specific agent, type of agent, or agent that plays a specific role.

Name: An optional identifier of the norm.

Evaluator: Originally used to denote the agent that observes norm compliance. Since the normative agent that created the norm is always responsible for this task in our system, the information is omitted.

Pertinence Condition: The condition under which the norm must be applied.

Postcondition: The condition that has to be brought about by the agent if the pertinence condition holds.

Policy: Consists of a condition and a sanction. The sanction is applied when the condition holds. Several policies can be given to make the sanction dependent on the situation. Sanctions can also be incentives and expressed as functions, which allows for gradual sanctions.

We suggest to make this structure more usable and versatile by instantiating the abstract description in a syntax based on OCL[1]. This allows us to reuse parts of the existing OCL-toolchain, including grammars and parsers for OCL with only small changes. OCL's expressiveness is also useful in reasoning about collections and relations. In addition, there is a clear mapping between the concepts referred to in the norms and the concepts depicted in design diagrams. This allows checking norms for consistency at design time. A class diagram introducing the concepts referred to in the norm in Fig. 4 is depicted in Fig. 3. In addition, an agent can exploit the direct mapping of the attributes used in the norm with its own knowledge. It is therefore not necessary to create an additional domain (or information) model as in some other approaches (e.g., [4]).

The norm in Fig. 4 describes the intended behaviour of an agent in a trusted desktop grid [1]. The `Worker`, an agent role that is responsible for accepting and processing `Jobs` is supposed to accept jobs from all requesters whose reputation value exceeds a certain threshold. If the norm is violated (i.e., jobs are rejected even though the requester's reputation is sufficient), the current agent's reputation is decreased. Likewise, if the requester's reputation is very high, the `Worker` receives an incentive if the `Job` is accepted. This ensures that a high reputation yields advantages in the system, thus leading rational agents to maximise their reputation.

[1]Object Constraint Language (http://www.omg.org/spec/OCL/), a formal language used to describe expressions on UML models.

$$\begin{aligned}
&\textbf{context}\ \overbrace{\text{Worker}}^{\text{Target/Role}}\ \textbf{norm}\ \overbrace{\text{AcceptJobs}}^{\text{Name}}:\\
&\quad\underbrace{\texttt{requester.reputation} \geq \text{lowReputationThreshold}}_{\text{Pertinence Condition}}\\
&\quad\textbf{implies}\ \underbrace{\texttt{self.acceptJob(job,requester)} = \texttt{true}}_{\text{Postcondition}}\\
&\textbf{sanctioned}\\
&\quad\textbf{if}\quad \underbrace{\text{violated}}_{\text{Default Sanction Condition}}\\
&\qquad\textbf{then}\ \underbrace{\text{self.reputation += -undueRejectionSanction}}_{\text{Sanction}}\\
&\textbf{incentivised}\\
&\quad\textbf{if}\ \underbrace{\texttt{requester.reputation} < \text{highReputationThreshold}}_{\text{Sanction Condition}}\\
&\qquad\textbf{then}\ \underbrace{\text{self.reputation += lowAcceptanceIncentive}}_{\text{Incentive}}
\end{aligned}$$

Fig. 4. An exemplary TDG norm in the proposed OCL-like syntax

References

[1] Yvonne Bernard, Lukas Klejnowski, Christian Müller-Schloer, Jeremy Pitt, and Julia Schaumeier. Enduring institutions and self-organising trust-adaptive systems for an open grid computing infrastructure. In *Proc. of the 2012 Sixth IEEE Int. Conf. on Self-Adaptive and Self-Organizing Systems Workshop (SASOW)*, pages 47–52. IEEE, 2012.

[2] Guido Boella, Gabriella Pigozzi, and Leendert van der Torre. Normative Systems in Computer Science – Ten Guidelines for Normative Multi-agent Systems. In *Normative Multi-Agent Systems*, number 09121 in Dagstuhl Seminar Proceedings. Schloss Dagstuhl, Leibniz-Zentrum fuer Informatik, Germany, 2009.

[3] Benedikt Eberhardinger, Jan-Philipp Steghöfer, Florian Nafz, and Wolfgang Reif. Model-driven Synthesis of Monitoring Infrastructure for Reliable Adaptive Multi-Agent Systems. In *Proc. of the 24th IEEE Int. Symposium on Software Reliability Engineering (ISSRE 2013)*, Pasadena, CA, November 2013. IEEE CS, Washington, D.C.

[4] Nicoletta Fornara and Marco Colombetti. Specifying and enforcing norms in artificial institutions. In *Declarative Agent Languages and Technologies VI*, volume 5397 of *LNCS*, pages 1–17. Springer Berlin Heidelberg, 2009.

[5] Urban Richter, Moez Mnif, Jürgen Branke, Christian Müller-Schloer, and Hartmut Schmeck. Towards a Generic Observer/Controller Architecture for Organic Computing. In *INFORMATIK 2006*, volume P-93 of *LNI*, pages 112–119. Bonner Köllen Verlag, 2006.

[6] Jan-Philipp Steghöfer, Gerrit Anders, Florian Siefert, and Wolfgang Reif. A System of Systems Approach to the Evolutionary Transformation of Power Management Systems. In *Proc. of INFORMATIK 2013 – Workshop on "Smart Grids"*, volume P-220 of *LNI*, pages 1–16. Bonner Köllen Verlag, 2013.

[7] Andreea Urzica and Cristian Gratie. Policy-Based Instantiation of Norms in MAS. In Giancarlo Fortino, Costin Badica, Michele Malgeri, and Rainer Unland, editors, *Intelligent Distributed Computing VI*, volume 446 of *Studies in Computational Intelligence*, pages 287–296. Springer Berlin Heidelberg, 2013.