Automated Requirements-Driven Definition of Norms for the Regulation of Behavior in Multi-Agent Systems

Martin Kollingbaum [1] and **Ivan J. Jureta** [2] and **Wamberto Vasconcelos** [3] and **Katia Sycara** [4]

Abstract. The engineering of heterogenous distributed systems is a complex task. Traditional software engineering methods fail to account for new demands of flexibility and adaptability in the construction of software systems. On the other hand, concepts of Virtual Organizations and Electronic Institutions cater for the need of open, heterogenous software environments, where agents may dynamically organize themselves into organizational structures, determined by roles, norms and contracts. Our work aims to facilitate the engineering of heterogenous and distributed systems by providing only a specification of the desired overall system behavior, expressed as a set of norms, and rely on capabilities and properties of individual agents that allow them to dynamically form the desired complete software system. In particular, we present a framework, called *Requirement-driven Contracting* (RdC), for automatically deriving executable norms from requirements and associated relevant information. RdC facilitates the governance of MAS by ensuring that all requirements, along with runtime changes of requirements are appropriately and automatically reflected in the norms regulating the behavior of MAS.

1 INTRODUCTION

Specifying requirements is in general a difficult and critical task – even more so for heterogenous systems, in which software developed, maintained, and operated by, and distributed across various organizations should cooperate in order to achieve joint goals. The perceived quality of the future system is determined by the fit between its behavior and the requirements. Moreover, such systems are expected to continually operate and adapt to changes in highly volatile environments. Changes lead to situations in which requirements (i) may not all be known before/during development, (ii) become obsolete over the course of development, and (iii) vary at runtime. Currently, established requirements and software engineering processes are built with homogenous, closed systems in mind. Classical requirements and software engineering processes start from the goals of the system, then identify and specify operations whose execution satisfies the goals, and finally design and implement components (agents) capable of executing the said operations. If requirements or environment conditions change, redesign and redeployment occur – this usually takes long and is costly, often leading system owners to miss opportunities for which they introduced the systems in the first place.

[1] The Robotics Institute, Carnegie Mellon University, email: mkolling@cs.cmu.edu
[2] PReCISE, University of Namur, Belgium, email: iju@info.fundp.ac.be
[3] Dept. of Computing Science, University of Aberdeen, UK, email: wvasconcelos@acm.org
[4] The Robotics Institute, Carnegie Mellon University, email: katia@cs.cmu.edu

Addressing these problems requires a methodological shift from the assembly of passive components towards systems that are based on active autonomous entities. Research into multi-agent systems (MASs) supports such a methodological shift, as it investigates the creation of systems involving dynamic, heterogenous, distributed and autonomous agents.

Virtual Organizations (VOs) and Electronic Institutions facilitate the dynamic formation of agent-based systems by introducing an organizational structure into an agent community and prescribing certain "rules of engagement" or norms, to regulate the actions and interactions of these agents. Agents may take on roles in such an organizational structure by signing contracts and thereby committing to observe established behavioural standards, i.e., *norms*.

Norms, that is, obligations, permissions, and prohibitions are useful abstractions for expressing rights and duties, and thereby regulating the behavior of heterogenous agents that act on behalf and in interest of their owners. Norms are critical for VOs, for they specify the organizational structure: agents adopt roles within the VO, and thereby the norms ascribed to the roles. By enabling agents to process norm specifications, a *separation of concerns* occurs: agents are not only described at an individual level, in terms of their capabilities, but a separate specification is also introduced to describe, at a social level, the compulsory, allowed, or forbidden behaviors. Norm-governed VOs, therefore, are an attractive approach to the engineering of heterogenous systems in changing environments, as executable normative specifications dynamically direct and tune the behavior of heterogenous agents.

It is *only if norms are appropriately defined* that agents can fulfil the purpose of the VO. As the VO's purpose is defined by the requirements of VO stakeholders (i.e., owners, users, etc.), *norms are appropriate only if they regulate the VO so that these requirements are satisfied to the most desirable (and feasible) extent*. Given, e.g., a requirement for a Banking application engineered on VO principles, that a *Letter of Credit* cannot be issued if there is no deposit, norms must ensure that this is the case at runtime. Approaches to the engineering of norm-governed MAS [11, 6, 10] focus on writing norms, either using limited requirements conceptualizations or leaving out requirements-level notions. Stakeholders, however, use richer notions than norms to communicate requirements. Their statements instead speak of VO goals in terms of functionality to provide and quality to achieve, how to provide/achieve these, and what to comply to (e.g., Sarbanes-Oxley Act). Stakeholders have preferences over alternative functionalities and quality levels, and priorities over preferences that cannot be jointly satisfied to the highest extent.

We therefore encounter two challenges if we are to further facilitate VO engineering: (a) provide a rich set of concepts for representing requirements, and (b) automatically derive appropri-

ate norms from the requirements. We present a framework, called *Requirements-driven Contracting* (RdC) that responds to both of these challenges, by integrating rich requirements-level concepts, a corresponding specification language, and algorithms for automatically deriving norms from requirements-level information. RdC thereby ensures that all requirements, along with runtime changes thereof are appropriately and automatically reflected in the norms regulating a VO. The RdC proceeds in three steps: (1) The VO engineer elicits the stakeholders' natural language statements about the purpose of the system. According to the speech act in which each statement is given, the VO engineer identifies requirements, domain assumptions, preferences, and priorities (Sect.2). (2) Using a set of templates and formal language constructs, the VO engineer writes the *environment specification* (ES) to transform requirements into system goals (Figure 2), domain assumptions into domain constraints, and define preferences over goals and priorities over conflicting preferences (Sect.3). (3) The VO engineer inputs the ES into RdC algorithms (Figures 3 and 5) to obtain an executable specificatons of norms that subsequently regulate the VO (Sect.4).

We consider these steps of the RdC process in detail in Sect.2–4. Sect.5 overviews related work; Sect.6 outlines conclusions, limitations, and directions for future work.

2 UNDERSTANDING REQUIREMENTS

Consider the transaction in which a Letter of Credit (LoC) is issued by a banker agent for use by a customer agent to finance the acquisition of goods from a supplier agent. The customer makes a deposit with the bank, then receives an LoC. The banker informs the supplier that an LoC has been issued to the customer, so that the customer can provide the LoC to the supplier, who can subsequently transfer the goods to the customer. To obtain the funds, the supplier sends the LoC to the bank. To engineer the VO for this setting, the engineer first elicits stakeholders' statements in natural language about the process, determines whether each of the statements is a requirement or otherwise, subsequently produces a specification that is translated into contracts comprising norms. In doing so, the engineer moves across three levels of abstraction covered by RdC: the *requirements*, the *ES*, and the *contracts* level. Figure 1 shows key useful RdC notions (only those discussed herein) at each of these levels.

At Level 1, statements about the high-level requirements, which the VO must satisfy, are elicited. A *functional requirement*, such as "Issue an LoC", will communicate what is desired or intended, whereas a *domain assumption* will indicate what is already the case [12], and is therefore an assertive or a declaration. A *nonfunctional requirement*, such as "Quickly issue an LoC" places additional constraints on a functional requirement by communicating an attitude thereon (through an expressive). Nonfunctional requirements typically involve gradable adjectives, such as quick, convenient, secure, or useful, efficient, accurate, and so on. Expressives therefore communicate preferences in an indirect manner: asking for "quick" implies that faster is preferred to slower. The same applies, e.g., for security, efficiency, maintainability, usability, and so on. We make the preference order explicit and thus order both nonfunctional and functional requirements. Introducing preferences entails the need for *priorities*, as often all preferences cannot be satisfied by the MAS. When aiming to satisfy one preferred requirement affects negatively the ability to satisfy some other requirement, we say that the involved preferences are conflicting. In such a case, a priority is defined to indicate which of the two preferences it is more important to satisfy.

We classify any requirement also as either compulsory or optional, and either conditional or free. The MAS must satisfy *compulsory requirements*, while it will (hopefully, though not necessarily) satisfy *optional requirements*. A requirement is *conditional* if it needs to be satisfied only when some particular conditions hold; otherwise, it is *free*. A domain assumption is either *free* or *conditional*, in the same sense as a requirement can be conditional or free. All mentioned taxonomic dimensions are relevant as they affect how the requirements and assumptions transform into norms (e.g., compulsory requirements give obligations and sactions).

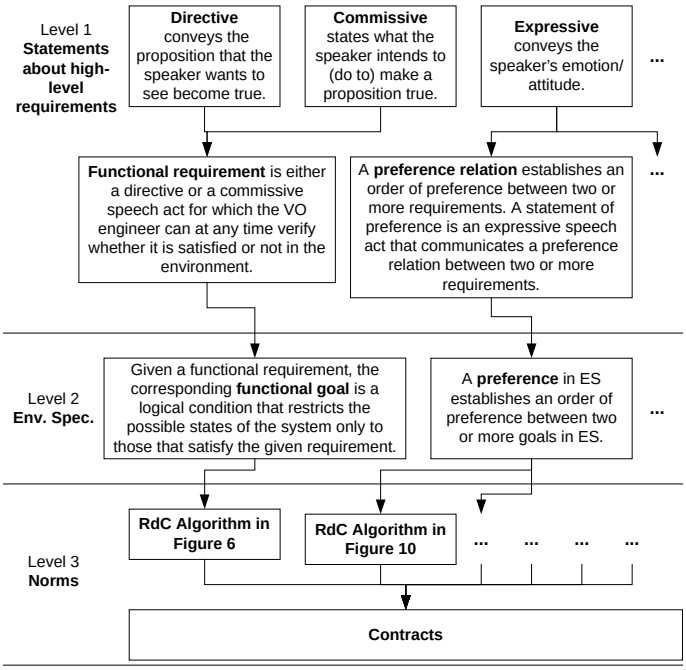

Figure 1. Part of RdC notions.

3 ENVIRONMENT SPECIFICATION

Given requirements, domain assumptions, preferences and priorities, we proceed to manually write the corresponding ES. The ES is divided into four parts. *Functional ES* specifies *functional goals* that need to be achieved (including *preferences* thereon), *plans* that should be followed, and *domain constraints* that must not be violated if the MAS is to satisfy functional requirements and domain assumptions. Achieving some functional goals requires that roles cooperate: we define *dependencies* between roles when the agent occupying a role can achieve a functional goal only if the agent occupying another role provides assistance. *Nonfunctional ES* define measures on the VO for monitoring purposes; *nonfunctional goals* define preferences over the values of these measures – if these values are observed at runtime, nonfunctional requirements and preferences thereon are satisfied. *Priorities ES* indicates *priorities* between conflicting preferences. Finally, *Terminological ES* specifies the domain ontology relevant for the VO, so as to better delimit the meaning intended for terms used throughout the ES, and subsequently carried over to norms. The ES is written as a collection of templates, one for each functional and nonfunctional goal (Figure 2), plan, domain constraint, priority, and dependency. Preferences are defined through the Preferences slot in templates (see, Figure 2; i.e., a preference order over functional goals appears in the Preferences slot of each of the goals in that preference order). Below, we discuss only functional goals and preferences thereon. The entire RdC framework is presented in the longer report [3].

3.1 Functional Goals

The compulsory and conditional functional requirement "Record a deposit in electronic and paper form if the deposit is received and approved" leads to the template in Figure 2. A unique identifier is given to the goal in UID for cross-referencing in ES and any graphical representation thereof. Type states whether the goal is functional or nonfunctional, compulsory (if the corresponding requirement is compulsory) or optional (if the requirement is optional), and conditional or free. Preferences gives the preference order in which the given goal appears. Element on the left hand side of $>>$ is preferred to the one on the right. As usual, the $>>$ relation is modeled as a strict partial order. A goal can be refined, that is, we can identify a set of goals (possibly easier to achieve) whose joint achievement is equivalent to achieving the refined goal. Supergoals lists those goals in whose refinement the given goal participates.

UID:	Record deposit in el. & paper form
Type:	Goal (Functional/Compulsory/Conditional)
Preferences:	(Record deposit in electronic form only $>>$ Record deposit in el. & paper form)
Supergoals:	*none*
BelongsTo:	Role (Banker)
Source:	Record a deposit in electronic and paper form if the deposit is received and approved.
Source type:	Requirement (Functional/Compulsory/Conditional)
Parameters:	d: Deposit
Satisfy:	$eL : elLog,\ pL : paLog\ logs(eL, pL, d) \leftarrow isPaperLog(pL, d), isElectronicLog(eL, d)$
Conditions:	$received(d), approved(d)$
Concepts:	Deposit = (and Amount (> 0) (exists has-purpose Credit) (all in-currency aset(USD,EUR))); paLog = (and Log (all has-purpose TransactionRecord) (all recorded-on Database)); elLog = (and Log (all has-purpose TransactionRecord) (all recorded-on Paper))

Figure 2. Instantiated Template for a Funct. Goal.

BelongsTo identifies the role or dependency to which the goal is associated. We then relate the goal to the requirement which it specifies: Source identifies the requirement at RdC Level 1 and SourceType indicates the classification of that requirement. Satisfy indicates the logical conditions that must be brought about in order to satisfy the goal's Source requirement. Logical conditions are written as Horn clauses. A general form of Horn clause is $a_0 \vee (\neg a_1) \vee \ldots \vee (\neg a_n)$, where each $a_i, i = 1, \ldots, n$ is an atom. We adopt here the standard notation: $a_0 \leftarrow a_1, \ldots, a_n$; whereby a_0 is true if $a_1, \ldots, a_n$ are true. Since the problem of testing a set of Horn clauses for satisfiability is known to have linear time solution algorithms, the ES supports rather efficient inferences compared to RE specification formalisms, which rely on variants of linear temporal first-order logic. For conditional goals, we write down in Conditions the logical conditions that must hold for the agents to know that the goal is to be achieved. Concepts lists the definitions of concepts whose instances are referred to in the template. We use the ITL (Information Terminological Lang. [7]) to define the domain ontology; therein, conceptual knowledge about a given domain is defined by a set of concepts and roles these concepts play in relationships, in which they take part. Each term intended to define a concept C is a conjunction of logical constraints, which are necessary for any object to be an instance of C.

4 NORMATIVE SPECIFICATION

We show in this section that the ES can be directly mapped to and expressed with normative concepts investigated in VO and Electronic Institutions research [5, 8]. Norms support the development of flexible as well as open VOs. Agents may join and leave VOs by adopting the normative standards of a VO via automated negotiation and signing of contracts. The normative specification defines contracts, whereby each contract is a set of norms. Norms explicitly specify behavioral directives for agents. Obligations, permissions and prohibitions of agents make requirements explicit at the normative level. Normative specifications, therefore, ensure that agents act only in ways that satisfy requirements and preferences, and are in accord with domain assumptions and priorities.

We use an available normative model [4], which is based on the notation outlined in [6]. We show how the ES relates to various governance measures, including norms, contracts, and roles and how they are derived from the ES. The building blocks of this notation are first-order terms, that is, constants, variables and functions (applied to terms). According to the chosen model, agents form social or organizational structures by taking on specific roles. These roles are determined by a set of norms, that is: the obligations the agent has to fulfill in the course of its actions and interactions with other agents, its prohibitions and permissions. The *role* concept allows us to abstract from individual agents and formulate patterns of behaviour that agents may adopt and conform to, with contracts defining these roles and the organizational structure of a VO. The set of norms Ω, determining the normative state of a complete VO, is described in the following way:

Definition 1. *A global normative state Ω is a finite and possibly empty set of norms.*

Ω describes the current overall normative state. Norms that are contained in this set are relevant to the agent – for example, an obligation contained in Ω must be fulfilled. When it is fulfilled, it has to be removed. Such a maintenance of the normative state has to be accommodated in order to capture the fact that requirements from the ES may be relevant to the overall system under specific circumstances only. A practical approach to the maintenance of a normative state is outlined in [2]. As a simplification, we add so-called *activation* and *expiration* conditions to norm specifications in order to capture circumstances when norms will be added to or removed from Ω.

In addition, the normative model we use introduces constraints. With that, the actual influence of norms on the agent achieving specific states of affairs can be restricted. These constraints are defined as follows:

Definition 2. *A constraint γ is any construct of the form $\tau \lhd \tau'$, where $\lhd \in \{=, \neq, >, \geq, <, \leq\}$.*

With that, we put forward following definition of norms:

Definition 3. *A norm ω is a tuple $\langle \nu, t_d, A, E \rangle$, where*

- *ν is any construct of the form $\mathsf{O}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i$ (an obligation), $\mathsf{P}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i$ (a permission) or $\mathsf{F}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i$ (a prohibition), where*
- *τ_1, τ_2 are terms, with τ_1 specifying a set of agents and τ_2 specifying a set of roles;*
- *φ is an atomic first-order formula, expressing the achievement of a state of affairs;*
- *$\gamma_i, 0 \leq i \leq n$, are constraints restricting the domains of variables occurring in φ;*
- *t_d is a time stamp recording the time of declaration of the norm;*
- *$A = \bigwedge_{j=0}^{n} \psi'_j, 0 \leq j \leq n$, is the activation condition comprising a conjunction of first-order predicates*
- *$E = \bigwedge_{k=0}^{n} \psi''_k, 0 \leq k \leq n$, is the expiration condition comprising a conjunction of first-order predicates*

In this formulation of norms, term τ_1 identifies the agent(s) to whom the norm is applicable. Term τ_2 is the role (or set of roles) of these agents. For example, $\mathsf{O}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n}\gamma_i$ thus represents an obligation on agent τ_1 taking up role τ_2 to bring about φ, subject to constraints γ_i, $0 \leq i \leq n$. The obligation *activates* (is added to Ω) when $\bigwedge_{j=0}^{n}\psi'_j$ holds, whereas the obligation deactivates (is removed from Ω) when $\bigwedge_{k=0}^{n}\psi''_k$ holds (the same holds for permissions and prohibitions). The γ_i's express constraints on those variables occurring in φ.

As pointed out earlier, the ES contains compulsory *and* optional goals. In both cases, an agent must be motivated to act. In addition to represent mandatory goals as obligations, means have to be put in place to *enforce* the fulfilment of obligations. A traditional means of enforcing law-abiding behavour in a social context is the specification of sanctions, i.e., actions typically performed by an authorised third party in case of norm violation. The purpose of sanctions is to either keep individuals from violating their duties or to compensate for an agent's behaviour. It is important to understand that sanctions are obligations for such an authorised party to act. As a consequence, the introduction of norms and contracts also requires the introduction of specific organizational structures where the role of such an authorised third party is established. Agents adopting such a role are permitted and obliged to pursue the activities defined by sanctions. In order to specify sanctions in a contract (or, in context of this paper, to generate them from the ES), obligations have to be specified for this specific authority role. Sanctions are then activated once a state of affairs indicating a violation of obligations holds. We define sanctions as follows:

Definition 4. *The concept of a sanction amounts to an obligation* $\mathsf{O}_{\tau_1:auth}\varphi \wedge \bigwedge_{i=0}^{n}\gamma_i$*, assigned to a specific Authority role* ***auth****, expressing an obligation for an agent in this role to achieve the state of affairs expressed by term* φ*.*

We ensure within RdC algorithms that sanctions are defined to react to violations of obligations, themselves derived from compulsory goals. In order to capture optional goals (corresponding to optional functional or nonfunctional requirements) with norms, we need the concept of *incentive*. Intuitively, an incentive motivates the agent to act in a certain way by indicating a reward for taking the desired actions. In our work, an incentive means that an agent will be motivated and not sanctioned if it is not successful. In other words, instead of rewarding, we do not sanction. Optional functional goals will therefore amount to obligations, for which sanctions are not defined, whereas compulsory functional goals will give rise to obligations with corresponding sanctions.

4.1 RdC Algorithms

As the purpose of contracts is to ensure that agents in a VO behave according to stakeholders' requirements and obey domain assumptions, we derive the normative specification, that is, contracts as sets of norms, using algorithms. The full framework contains algorithms for converting all ES-level information into norms. Below, we present two of them for transforming functional goals while accounting for preferences and dependencies. First, we have a simple definition for the *contract* concept:

Definition 5. *A contract* ***C*** *is of the form*

$$\langle\langle r_1, \{\omega_1^{r_1}, \ldots, \omega_m^{r_1}\}\rangle, \ldots, \langle r_q, \{\omega_1^{r_q}, \ldots, \omega_p^{r_q}\}\rangle\rangle$$

where each r *is a role identifier, and each* ω *a norm.*

For simplicity, we observe some notational conventions. Let FG be a functional goal, and FG.UID be the value of the UID slot in the template for FG (e.g., Figure 2). We refer to values in other template slots in the same obvious way (e.g., FG.Satisfy for the value of the Satisfy slot in the template for FG). Following earlier discussions, we assume that the value of the Satisfy and Uphold attributes is of the form $a_0 \leftarrow a_1, \ldots, a_n$ whereby $n \geq 1$ and a_0 can be empty. Also let the value of Conditions slot wherever it appears in ES be of the form $b_0 \leftarrow b_1, \ldots, b_m$ whereby $m \geq 1$ and b_0 can be empty. Recall that any $a_i, 0 \leq i \leq n$ and any $b_j, 0 \leq j \leq m$ is an atomic first-order formula. Also, we write $(\cdot)$ for content of no interest in the particular discussion or algorithm – e.g., if we write the norm $\langle(\cdot),(\cdot),\bigwedge_{j=0}^{n}\psi'_j,(\cdot)\rangle$, we are interested only in $\bigwedge_{j=0}^{n}\psi'_j$ while the content of the other elements can be any allowed by the norm definition.

4.1.1 Contracts from Functional Goals

In this section, we present the RdC algorithm, shown in Figure 3, for deriving a contract from a functional goal. We illustrate the output of the algorithm by converting the functional goal in Figure 2 into the contract shown in Figure 4. The algorithm proceeds as follows. Given the ES, we first select a functional goal that has not yet been subjected to the algorithm in Figure 3. We consider only primitive goals in the goal hierarchy. We generate as many obligations as needed to cover the Horn clause in the goal template's Satisfy slot, whereby each obligation deactivates as soon as it is realized (i.e., $\bigwedge_{k''=0}^{r''}\psi''_{k''} = a_i$). If the functional goal is conditional, the activation condition for each obligation corresponds to the condition for that functional goal (lines `3–8` in Figure 3). If the goal is also compulsory, we create sanctions, that is obligations for the authority role. Sanctions activate if the corresponding obligations are not realized (`9–11`). If the goal is optional, we create no sanctions (`12–14`). We then create a new contract (`15–17`), in which the roles vary if the functional goal belongs to a role or a dependency. The algorithm returns for each goal a set of norms according to the type of the functional goal. Obligations and sanctions are generated if the goal is compulsory, while no sanctions are created if the goal is optional. Contracts are created and norms are distributed between the role being supervised and the authority (i.e., supervisor) role. The supervisor role (Authority) receives responsibility for sanctions, while the supervised role receives obligations. The algorithm terminates, as each of the **for each** loops goes through finite sets and considers one element at a time. We have the contract $\langle\langle Banker, \{\omega_1, \omega_2\}\rangle, \langle BankerAuth, \{\omega_1^{sanct}, \omega_2^{sanct}\}\rangle\rangle$ on the functional goal "Record deposit in electronic & paper form" with norms and sanctions shown in Figure 4.

The $BankAuth$ is the role that acts as the authority for the $Banker$ role. The functional goal gives us two obligations for the $Banker$ role and the corresponding sanctions enforced by the $BankAuth$ role. $\mathsf{O}_{a:BankAuth}\phi'$ and $\mathsf{O}_{a:BankAuth}\phi''$ are defined by the VO engineer; these obligation determine how the sanction is applied in case of violation.

4.1.2 Preferences over Contracts on Functional Goals

In this section, we present the RdC algorithm for adjusting norms according to preferences, shown in Figure 5. Preferences establish an order over requirements, and therefore over goals in the ES. The preference order defined over functional goals corresponds to a preference order over contracts. We do not, however, carry over the very

Contract from Functional Goal

`Input :` One functional goal FG such that: (i) FG has not been transformed previously using this algorithm; (ii) there are no functional goals in ES for which FG is the supergoal.

`Output:` One contract **C** including norms for ensuring that logical conditions of FG must or can be brought about.

`begin`

`1` **For each** $a_i, 1 \leq i \leq n$ in FG.Satisfy **do**

`2` Create obligation $\omega_i = \langle \mathsf{O}_{a:r}\phi \wedge \bigwedge_{k=1}^{r} \gamma_k, t_d, \bigwedge_{k'=0}^{r'} \psi'_{k'}, \bigwedge_{k''=0}^{r''} \psi''_{k''} \rangle$ where $\phi = a_i$ and $\bigwedge_{k''=0}^{r''} \psi''_{k''} = a_i$ and t_d is recorded automatically.

`3` **If** FG.Type is Conditional **then**

`4` **For each** $b_j, 1 \leq j \leq m$ **do** $r' = m$ and $\psi'_{k'} = b_j$

`5` **End If.**

`6` **If** FG.Type is Free **then**

`7` every b_j is empty and every $\psi'_{k'}$ remains unchanged.

`8` **End If.**

`9` **If** FG.Type is Compulsory **then**

`10` Create sanction $\omega_i^{sanct} = \langle \mathsf{O}_{a:auth}\phi \wedge \bigwedge_{k=1}^{r} \gamma_k, t_d, \bigwedge_{k'=0}^{r'} \psi'_{k'}, \bigwedge_{k''=0}^{r''} \psi''_{k''} \rangle$ where $\bigwedge_{k'=0}^{r'} \psi'_{k'} = \neg a_i$ and $\bigwedge_{k''=0}^{r''} \psi'_{k''} = a_i$, and the VO engineer is asked to provide $\mathsf{O}_{a:auth}\phi \wedge \bigwedge_{k=1}^{r} \gamma_j$.

`11` **End If.**

`12` **If** FG.Type is Optional **then**

`13` no sanctions are defined for FG.

`14` **End If.**

`15` Create contract $\mathbf{C}_{\mathsf{FG}} = \langle \langle r_1, \{\omega_i | 1 \leq i \leq n\} \rangle, \langle r_2, \{\omega_i^{sanct} | 1 \leq i \leq n\} \rangle \rangle$ where:

`16` **If** FG.BelongsTo is Role **then** r_1 is the name of that role and r_2 is an authority role $auth$. **End If.**

`17` **If** FG.BelongsTo is Dependency **then** r_1 is the name of the depender role and r_2 is an authority role $auth$. **End If.**

`end`

Figure 3. RdC Algorithm for Deriving a Contract from a Functional Goal.

$$\omega_1 = \langle \mathsf{O}_{a:Banker} isPaperLog(pL,d), (\cdot), received(d) \wedge approved(d), isPaperLog(pL,d) \rangle$$

$$\omega_2 = \langle \mathsf{O}_{a:Banker} isElectronicLog(pL,d), (\cdot), received(d) \wedge approved(d), isElectronicLog(pL,d) \rangle$$

$$\omega_1^{sanct} = \langle \mathsf{O}_{a:BankAuth}\phi', (\cdot), \neg isPaperLog(pL,d), isPaperLog(pL,d) \rangle$$

$$\omega_2^{sanct} = \langle \mathsf{O}_{a:BankAuth}\phi'', (\cdot), \neg isElectronicLog(pL,d), isElectronicLog(pL,d) \rangle$$

Figure 4. Norms Obtained from the Functional Goal Given in Figure 2.

notion of preference order onto the normative specification. We instead use an approach that does not involve extending the conceptualizations at the normative level. Overall, we know that a preference for a functional goal A over B means that honoring the contract on A is more desirable than honoring the contract on B. In absence of preference orders to establish relative desirability at the normative level, we must rely on activation and expiration constraints in norms. Namely, we can place activation constraints on norms of contract B so that those norms are activated (i.e., agents go about honoring the contract on B) only if the contract on A cannot be honored. Consequently, we can constrain the activation of norms in B to cases when norms in A cannot be honored. There is, however, no absolute criterion for knowing whether a contract cannot be honored. We therefore leave it to the VO engineer to choose a set of *critical* obligations in the contract on A that, once violated, mean that the contract on A cannot be honored, and that the contract on B is to be activated. In summary, if we have a preference order $A >> B >> C$, then if the contract obtained on A is not honored (i.e., critical obligations in the contract on A are violated), we activate the contract on B, and if the contract on B is not honored, we activate the contract on C. To activate the contract on B, we introduce additional activation constraints to those already defined within the norms in B: if the additional constraints hold, the contract on A is not honored.

The algorithm in Figure 5 considers each preference pair in a preference order (line `1` in Figure 5). Given a pair of functional goals, we take the more preferred functional goal FG_l, and consider individually each of the critical obligations appearing in the contract on that goal. For each critical obligation (`4`–`6`), we have the violation of the obligation's ϕ as an additional activation condition to each of the norms appearing in the contract on the less preferred goal FG_{l+1} (`5`). We also (`6`) add expiration conditions so that the contract on the less preferred goal is not activated when the critical obligations on the more preferred one are honored. By doing so, we ensure that the contract on the less preferred goal will only be considered if all critical obligations on the more preferred goal are violated. The algorithm in Figure 5 ensures that the norms on each less preferred options in the given preference order are activated only when norms on more preferred options cannot be honored. The algorithm always terminates as all of the **for each** loops move through finite sets, and always process one element at a time.

5 RELATED WORK

Frameworks and methodologies for the engineering of multi-agent systems [1, 11, 10] start from high-level goals of the system, then identify operations to achieve these goals, and design components that will perform the operations. They do not use the autonomy and adaptability inherent in agents to facilitate the engineering and development of MAS.

Preferences over Norms from Functional Goals

`Input :` One preference order (from ES and among functional goals) that has not been transformed previously using this algorithm. Assume for simplicity that the order involves w goals, and is of the form $\mathsf{FG}_1.\mathsf{UID} >> \mathsf{FG}_2.\mathsf{UID} >> \mathsf{FG}_3.\mathsf{UID} >> \ldots >> \mathsf{FG}_w.\mathsf{UID}$, and that a "preference pair" from that order is $\mathsf{FG}_l.\mathsf{UID} >> \mathsf{FG}_{l+1}.\mathsf{UID}$, for $1 \leq l \leq w-1$.

`Output :` Per preference pair, a norm on each less preferred goal in the preference pair, updated with constraints ensuring that the norms in the contract activate only if the norm on the more preferred goal cannot be honored.

`begin`

`1` **For each** preference pair $\mathsf{FG}_l.\mathsf{UID} >> \mathsf{FG}_{l+1}.\mathsf{UID}$, $1 \leq l \leq w-1$ **do**

`2` Choose a set of *critical* obligations in the contract on FG_l;

`3` **For each** critical obligation $\omega_i = \langle \mathsf{O}_{a:r}\phi \wedge \bigwedge_{k=1}^{r} \gamma_k, (\cdot), (\cdot), (\cdot) \rangle$ **do**

`4` **For each** norm $\langle (\cdot), (\cdot), \bigwedge_{k'=0}^{r'} \psi'_{k'}, \bigwedge_{k''=0}^{r''} \psi''_{k''} \rangle$ in the contract on FG_{l+1} **do**

`5` Replace $\bigwedge_{k'=0}^{r'} \psi'_{k'}$ by $\psi'_{r'+1} \wedge \bigwedge_{k'=0}^{r'} \psi'_{k'}$ where $\psi'_{r'+1} = \neg\phi$;

`6` Replace $\bigwedge_{k''=0}^{r''} \psi''_{k''}$ by $\psi''_{r''+1} \wedge \bigwedge_{k''=0}^{r''} \psi''_{k''}$ where $\psi''_{r''+1} = \phi$;

`end`

Figure 5. RdC Algorithm for Deriving Norms from a Preference Order over Functional Goals.

In our approach, the standard MAS engineering process is more efficient since it needs to describe only the desired overall system behavior and relies on the capabilities and properties of individual agents to assemble complete software systems, negotiate their roles therein, and operate according to the given normative specification. Our approach therefore relies on the premise that functionality is available in the form of individual autonomous agents, designed, developed, and maintained by, and distributed across many organizations (e.g., Google, Microsoft, etc.). The VO engineer therefore need not specify and implement individual agents, but instead determine how to regulate their interactions so that requirements are satisfied to the most desirable extent.

Our work has been influenced by the Tropos methodology [1], which features rich requirements models. Tropos does not, however, integrate preferences, priorities, and norms. Precise specification in RdC relies on a less expressive though computationally more attractive formalism. The role of a domain ontology is implicit in Tropos, while it is explicit in RdC, again due to the focus on VO. Tropos does not focus on governed MAS, but instead assumes that agents are implemented according to the requirements. RdC automates some activities that are manual in Tropos: given already designed agents, RdC governs their behavior through the conversion of requirements to norms.

6 CONCLUSIONS

We introduced Requirements-driven Contracting (RdC), which combines research into rich requirements engineering conceptualizations with research on norm-based specifications of multi-agent systems. The framework allows the specification of rich requirements models and the automatic derivation of executable normative specifications that express the obligations, permissions, and prohibitions regulating behaviors of agents participating in multi-agent systems.

Future work will address several points. First, automated resolution of conflicts between norms at our governance level is available [9], where first-order term unification is employed to find out if and how norms overlap in their influence. By integrating this work with RdC, we will be able to address inconsistencies in an automated manner at the lower, governance level. More work is necessary for combining inconsistency detection and resolution at both the ES and governance levels. Second, we rely on supervised interaction to ensure supervision of agent behavior; more elaborate role structures can be obtained by introducing additional role relationships (e.g., if an agent occupies one role, it cannot occupy some other). Third, additional normative concepts, such as that of power and loyalty (i.e., matching between individual and MAS goals), must be studied. Calculating loyalty would facilitate the allocation of resources for supervision, so that, e.g., higher initial trust may be assigned to agents with higher loyalty values.

REFERENCES

[1] Jaelson Castro, Manuel Kolp, and John Mylopoulos, 'Towards requirements-driven information systems engineering: the tropos project', *Information Systems*, **27**(6), 365–389, (2002).

[2] Andrés García-Camino, Juan-Antonio Rodríguez-Aguilar, Carles Sierra, and Wamberto Vasconcelos, 'A Rule-based Approach to Norm-Oriented Programming of Electronic Institutions', *ACM SIGecom Exchanges*, **5**(5), 33–40, (January 2006).

[3] Ivan J. Jureta, Martin Kollingbaum, Stephane Faulkner, John Mylopoulos, and Katia Sycara, 'Requirements-driven contracting for norm-governed multi-agent systems', Technical report, University of Namur, (October 2007; Also available online: http://www.jureta.net/papers/RDC.pdf).

[4] Martin J. Kollingbaum, Wamberto W. Vasconcelos, Andrés García-Camino, and Timothy J. Norman, 'Conflict Resolution in Norm-Regulated Environments via Unification and Constraints', in *DALT*, (2007).

[5] T.J. Norman, A. Preece, S. Chalmers, N.R. Jennings, M. Luck, V.D. Dang, T.D. Nguyen, V. Deora, J. Shao, W.A. Gray, and N.J. Fiddian, 'Agent-based Formation of Virtual Organisations', *Knowledge Based Systems*, **17**, 103–111, (2004).

[6] Olga Pacheco and José Carmo, 'A role based model for the normative specification of organized collective agency and agents interaction', *Autonomous Agents and Multi-Agent Systems*, **6**(2), 145–184, (2003).

[7] Katia P. Sycara, Seth Widoff, Matthias Klusch, and Jianguo Lu, 'Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace', *Autonomous Agents and Multi-Agent Systems*, **5**(2), 173–203, (2002).

[8] W. Vasconcelos, 'Norm Verification and Analysis in Electronic Institutions', in *AAMAS 2004 Workshop Declarative Agent Languages and Technologies (DALT 2004)*, (2004).

[9] Wamberto Vasconcelos, Martin J. Kollingbaum, and Timothy J. Norman, 'Resolving conflict and inconsistency in norm-regulated virtual organizations', in *Proceedings of AAMAS*, (2007).

[10] Javier Vázquez-Salceda, Virginia Dignum, and Frank Dignum, 'Organizing multiagent systems', *Autonomous Agents and Multi-Agent Systems*, **11**(3), 307–360, (2005).

[11] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge, 'Developing multiagent systems: The gaia methodology', *ACM Trans. Softw. Eng. Methodol.*, **12**(3), 317–370, (2003).

[12] Pamela Zave and Michael Jackson, 'Four dark corners of requirements engineering', *ACM Trans. Softw. Eng. Methodol.*, **6**(1), (1997).