A Schema for Specifying Computational Autonomy

Matthias Nickles, Michael Rovatsos, and Gerhard Weiß

Institut für Informatik, Technische Universität München,
D-85748 Garching, Germany
{nickles,rovatsos,weissg}@in.tum.de

Abstract. A key property associated with computational agency is autonomy, and it is broadly agreed that agents as autonomous entities (or autonomous software in general) have the capacity to become an enabling technology for a variety of complex applications in fields such as telecommunications, e/m-commerce, and pervasive computing. This raises the strong need for techniques that support developers of agent-oriented applications in specifying the kind and level of autonomy they want to ascribe to the individual agents. This paper describes a specification schema called RNS ("Roles, Norms, Sanctions") that has been developed in response to this need. The basic view underlying RNS is that agents act as owners of roles in order to attain their individual and joint goals. As a role owner an agent is exposed to certain norms (permissions, obligations and interdictions), and through behaving in conformity with or in deviation from norms an agent becomes exposed to certain sanctions (reward and punishment). RNS has several desirable features which together make it unique and distinct from other approaches to autonomy specification. In particular, unlike other approaches RNS is strongly expressive and makes it possible to specify autonomy at a very precise level. Moreover, RNS is domain- and application-independent, and is of particular value for agent-oriented requirements elicitation and analysis.

1 Introduction

A key property associated with computational agency is autonomy. As an autonomous entity, an agent possesses action choice and is able to act under self-control within the bounds of its design objectives. Compared to other basic properties usually associated with agency such as behavioral flexibility (covering both reactivity and pro-activeness) and high-level interactivity (based on communication and negotiation with the purpose of cooperation or competition), it is autonomy that makes agent orientation most distinct from traditional software and systems development paradigms. These paradigms, relying on concepts such as objects or components, simply are not intended to capture the notion of computational autonomy – to the contrary, they can even be said to be intended to prevent autonomous component behavior. The past years have witnessed a rapidly growing interest in various aspects of computational autonomy, as it is

also indicated by an increasing number of related research efforts (e.g., see [1, 4, 5, 8] for work explicitly dealing with autonomy). This interest is largely based on the insight that agents as autonomous entities – or autonomous software in general – do have the capacity to become an enabling technology for a broad and important class of applications, namely, applications that run in complex socio-technical environments which are open, dynamic, networked, time-critical, and/or decentralized and thus possess a critical mass of inherent unpredictability. Due to this unpredictability a full behavioral specification of application software is not possible in many cases, and this is the point where autonomously acting entities come into play which are able to act in a desired manner not only in anticipated environmental situations but also in situations that were unforeseeable at design time. Well known examples of application domains calling for a deployment of "autonomous technology" are telecommunications, logistics, e/m-commerce, supply chain management, and pervasive and ubiquitous computing.

Among the most critical activities in engineering agent-based applications is the specification of the kind and level of autonomy owned by the different agents. This specification can fail in two opposite ways, both making it unlikely that a resulting application meets its requirements: on the one hand, if this specification is too rigid then necessary action choice is suppressed and "objects are made out of agents"; and on the other hand, if this specification is too generous then unnecessary action choice is admitted and "agents are made out of objects". This *autonomy specification dilemma* raises the strong need for supporting a developer in specifying the agents' action choice. More specifically, what is needed are specification techniques – methods, formalisms, tools, languages, and so forth – which enable and force developers to *precisely* state what degrees of behavioral freedom they want to ascribe to the individual agents. This paper describes a specification schema called RNS (standing for "Roles, Norms, Sanctions") which has been developed in response to this need. This schema employs the concepts of roles, norms (permissions, obligations, and interdictions) and sanctions (reward and punishment) to capture autonomy. RNS, which is particularly suited for agent-oriented requirements elicitation and analysis, possesses several desirable features which together make it unique and distinct from related approaches to autonomy specification. In particular, although RNS is based on a relatively simple and easy-to-understand notation and syntax, it is very expressive and enables a developer to specify agent autonomy with a very high precision. Moreover, RNS is general enough to be not bound to any specific application or application domain.

The paper is structured as follows. Section 2 describes RNS in detail; this includes a general characterization (2.1) and a detailed technical presentation (2.2 and 2.3). This section also gives a number of illustrating examples of all basic aspects of RNS. Finally, Section 3 discusses pros and cons of RNS and compares RNS to related approaches.

2 The RNS Schema

2.1 Informal Characterization

RNS employs the concepts of role, norm and sanction as known from sociological role theory (e.g., [3]) to specify autonomy. The basic view underlying RNS is that autonomous agents are embedded in a social frame which regulates – guides and constrains – their behavior. This social frame, called role space, is composed of roles which are available to the agents and through which the agents can try to achieve individual and joint objectives. An agent may own several roles at the same time, and the set of roles owned by an agent may dynamically vary over time. Conceptually roles are viewed as a means for specifying desired behavior and for achieving behavioral predictability, and *not* as a means for making sure that agents do never exhibit unexpected and undesirable behavior. Roles in RNS are not intended to fully constrain individual behavior; instead, they leave room for individuality (agents may fill a role differently by putting emphasis on different aspects). Somewhat more specifically, according to RNS a role consists of at least one activity to which norms together with sanctions are attached. RNS distinguishes three types of norms (permissions, obligations, and interdictions) and two types of sanctions (reward and punishment). Whereas norms specify behavior expectations held by agents against other agents (in their capacity as role owners), sanctions specify potential consequences of norm-conforming and norm-deviating behavior. Sanctions, in some sense, serve as a means for controlling autonomy. By enabling a designer to explicitly specify sanctions, RNS takes care of the fact that generally (and especially in open applications) agents as autonomous entities do not necessarily behave in conformity with available norms, but may also ignore and violate them (be it intentionally or not).

2.2 Basic Concepts and Constructs

The RNS schema requires to analyze and specify systems in terms of roles which are available to the agents and through which the agents can try to achieve their objectives. The set of available roles is called a *role space*. A role space is specified in the form

ROLE SPACE *role_space_id* { *role_id_list* }

where *role_space_id* is a character string uniquely identifying the role space under consideration and *role_id_list* is a list of character strings called role identifiers that uniquely identify roles.[1] *Roles* are viewed as collections of specific activities, and for each role identifier, *role_id*, there must be a role specification in the form

ROLE *role_id* { *activity_spec_list* }

[1] Syntactic keywords are written in underlined TYPEWRITER FONT, and *italic font* is used to indicate variables. Expressions enclosed in brackets [.] are optional. Brackets of the form <.> are part of the RNS syntax. As the reader will see in the examples provided below, most of the variables also can be instantiated with specific keywords such as EACH.

where *activity_spec_list* is a list of specifications of all activities being part of the role *role_id*. How activity specifications look like is described in section 2.3.

The RNS schema distinguishes three types of norms – permissions (**P**), obligations (**O**), and interdictions (**I**) – and two types of sanctions – reward (**RE**) and punishment (**PU**) – that apply in case of norm conformity and deviation. Based on these distinctions, a *status range* is attached to each activity which describes activity-specific norms and associated sanctions. More specifically, a status range specification is of the form

`STATUS RANGE` *status_statement_list*

where *status_statement_list* is a list of so called *status statements* each describing a norm-sanction pair that is specific to the activity to which the status range is attached. A key feature of the RNS schema is that it facilitates the explicit modeling and specification of *requests* for (refraining from) executing particular activities. This feature induces the distinction of two kinds of norm-sanction pairs attached to an activity:

- norm-sanction pairs an activity is subject to, no matter whether the execution or omission of the activity is requested or not by some agent. Norm-sanction pairs of this kind are, so to say, independent of any requests for (not) executing the activity to which they are attached. Norm-sanction pairs of this kind, and the status statements describing them, are called *independent* and are indicated by the keyword `IND`.
- norm-sanction pairs an activity becomes subject to as a consequence of a request for (not) executing it. Such norm-sanction pairs are, so to say, induced by (i.e., do become active as an effect of) explicit requests for activity execution or omission. Agents requesting the (non-)execution of an activity are called role senders. Norm-sanction pairs of this kind, and the status statements describing them, are called *dependent* and are indicated by the keyword `DEP`.

The common syntax of independent status (IS) statements and dependent status (DS) statements is as follows:

$$\underbrace{\underbrace{<\textit{status\_type}> : \texttt{NORM} <\textit{norm\_type}> <\textit{condition}>}_{\text{norm specification}} + \underbrace{\texttt{SANC} <\textit{sanction\_type}> <\textit{sanction}>}_{\text{sanction specification}}}_{\text{norm-sanction pair}}$$

where *status_type* $\in$ {`IND`, `DEP` *role_id*} discriminates among IS and DS statements, *norm_type* $\in$ {**P**, **O**, **I**}, *condition* is a Boolean expression making it possible to formulate conditioned norms, *sanction_type* $\in$ {**RE**, **PU**}, and *sanction* is an expression specifying a sanction of type *sanction_type*. Though syntactically almost identical, IS and DS statements differ significantly in their semantics. First consider IS statements, that is, statements of the form

<IND> : NORM <*norm_type*> <*condition*> + SANC <*sanction_type*> <*sanction*>

Dependent on *norm_type*, such a statement attached to an activity reads as follows:

- *norm_type* = {**P**}: "An agent owning the role of which this activity is part of is *permitted* to execute this activity provided that the condition *condition* is fulfilled. The sanction associated with this permission is of type *sanction_type* and is given by *sanction*."
- *norm_type* = {**O**}: "An agent owning the role of which this activity is part of is *obliged* to execute this activity provided that the condition *condition* is fulfilled. The sanction associated with this obligation is of type *sanction_type* and is given by *sanction*."
- *norm_type* = {**I**}: "An agent owning the role of which this activity is part of is *forbidden* to execute this activity provided that the condition *condition* is fulfilled. The sanction associated with this interdiction is of type *sanction_type* and is given by *sanction*."

Against that, DS statements, that is, statements of the form

<DEP *role_id*> : NORM <*norm_type*> <*condition*> + SANC <*sanction_type*> <*sanction*>

read as follows:

- *norm_type* = {**P**}: "If an agent owning the role *role_id* requests to execute this activity (from an agent owning the role of which this activity is part of), then the requested agent is *permitted* (by the requesting agent) to execute it (i.e., she may execute it) provided that the condition *condition* is fulfilled. The sanction associated with this permission is of type *sanction_type* and is given by *sanction*."
- *norm_type* = {**O**}: "If an agent owning the role *role_id* requests to execute this activity, then the requested agent is *obliged* (by the requesting agent) to execute it (i.e., she must execute it) provided that the condition *condition* is fulfilled. The sanction associated with this obligation is of type *sanction_type* and is given by *sanction*."
- *norm_type* = {**I**}: "If an agent owning the role *role_id* requests to *not* execute this activity, then the requested agent is *forbidden* (by the requesting agent) to execute it (i.e., she must not execute it) provided that the condition *condition* is fulfilled. The sanction associated with this interdiction is of type *sanction_type* and is given by *sanction*."

DS statements make it possible to capture situations in which requests (e.g., from different agents) for executing an activity do have different normative and sanctioning impacts on the requested agent. In other words, DS statements allow to model situations in which requests even for the very same activity induce different norms and sanctions. With that, the RNS schema is highly sensitive to normative and sanctioning contexts.

2.3 Activity Types

According to the RNS schema, four types of activities are distinguished:

- Basic activities, that is, resource and event handling activities (*Type I*).
- Request activities, that is, requests for executing activities (*Type II*).
- Sanctioning activities, that is, activities that result in a punishment of behavior deviating from available obligations and interdictions, as well as activities that result in a rewarding of behavior going conform with permissions, obligations and interdictions (*Type III*).
- Change activities, that is, activities that result in changes of status statements being part of the status range of an activity of any type (*Type IV*). As a status statement consists of a norm specification and a sanction specification, change activities can be also characterized as activities that result *(i)* in changes of norms attached to an activity and/or *(ii)* in changes of sanctions associated with such norms.

Each of these four types of activities may be subject to (or "the target of") an activity of types II, III, and IV. This means, in particular, that the RNS schema allows to formulate "crossed and self-referential" constructs such as requests for requests, requests for sanction and norm changes, changes of norms attached to norm-changing activities (as well as requests for such changes), and changes of sanctions attached to sanction-changing activities (as well as requests for such changes). Examples of such constructs, which we call *activity reference constructs*, are provided below. In the following, the four activity types are described in detail.

Resource and Event Handling Activities. These activities are highly domain- and application-specific. Two types of resources are distinguished, namely, consumable ones (e.g., time, money, and any kind of raw material to be processed in a manufacturing process) and non-consumable ones (e.g., data, protocols, and communication support services such as blackboard platforms and translation systems). Examples of such activities are

provide(*CPU_time*), deliver(*material,quantity*), access(*database*),
run-protocol(*joint-planning*), kick-ball(*position*), acknowledge-receipt(*data*).

The RNS specification of this type of activities has the general form

ACT *activity_id* (*activity_variable_list*)
{ STATUS RANGE *status_statement_list* }

where *activity_variable_list* is a list of variables specific to the activity *activity_id*. The first line of any activity specification, starting with the keyword ACT, is called an *activity header*, and the part enclosed in **{.}** is called an *activity body*. Here is an example of a specification of a basic activity. Assume there is a role with identifier USsupplier, and that one of its basic activities is specified as follows:

ACT deliver (*material,quantity*)
{ STATUS RANGE
<IND> : NORM <**P**> <NO> + SANC <NO> <NO>
<DEP EACH> : NORM <**O**> <*quantity* $\leq$ 100> + SANC <**PU**> <withdraw_role>
<DEP AssemblyMg> : NORM <**I**> <*material* = steel> + SANC <**PU**> <pay_fine>
}

The keyword EACH used as an instantiation of *role_id* (*agent_id*) indicates that *all* roles (agents) are concerned, and the keyword NO used as an instantiation of

condition (of *sanction_type* and *sanction*) indicates that the norm is unconditioned (that there is no associated sanction). With that, in this example the IS statement says that an agent owning the role of which the deliver activity is part of is permitted to deliver. The first DS statement says that a request from each agent (no matter what role she owns within the role space under consideration) for executing this deliver activity induces the obligation to deliver, provided that the requested quantity is not above 100. Furthermore, the statement says that the requested agent must withdraw the role USsupplier (i.e., is not longer allowed to act as a USsupplier) in the case of violating such an induced obligation. The second DS statement says that the delivery of steel, if requested by an agent owning the role AssemblyMg ("Assembly Manager"), is forbidden; not acting in accordance with this interdiction is punished by some fine.

Execution Requests. These activities are specified as follows:

ACT REQUEST (*agent_id_list* ; *role_id_list* ; [NOT] *activity_id* (*activity_variable_list*))
{ STATUS RANGE *status_statement_list*
NORMATIVE IMPACT *norm_specification_list* }

The activity header says that requests can be directed towards any agent who is referred to in *agent_id_list* and who owns at least one of the roles listed in *role_id_list*. The header also identifies the activity being subject to the request. The keyword NOT is optional and is to be used only in the case of interdiction (i.e., in the case of requests for not executing some activity). *norm_specification_list* specifies the normative impact of the request on the requested agent(s) through a list of norm specifications. As already introduced above, these specifications are of the form

NORM <*norm_type*> <*condition*>

Note that every norm specification included in a normative impact specification of a request activity, together with the identifier of the role of which the request activity is a part, unambiguously points to a single or (if there are multiple sanctions – rewards and punishments – associated with the induced norm) several DS statements.

As an illustrating example based on the delivery activity specified above, consider the following request activity specification being part of the role AssemblyMg:

ACT REQUEST (EACH ; USsupplier, EUROsupplier ; NOT deliver (*material*, *quantity*))
{ STATUS RANGE
<IND> : NORM <**P**> < (*material* = steel) AND (rating(*material*) = poor)> +
SANC <NO> <NO>
NORMATIVE IMPACT
NORM <**I**> <*material* = steel>
}

The keyword EACH says that the request can be directed towards each agent owning the roles USsupplier or EUROsupplier. (If *role_id_list* were also instantiated with EACH, then this would mean that each agent – without any role restriction –

can be requested to deliver.) Generally, the keyword EACH serves as a wildcard, and expressions including it are called *templates*.

A potential, legal occurrence or "call" of this request activity (which, for instance, could be part of interaction protocols) during run time is the following:

REQUEST (Dr_Meyer, Mr_Black ; USsupplier ; NOT deliver (steel, [0 . . . 500]))
(i.e., the USsuppliers Dr_Meyer and Mr_Black are requested to not accept steel delivery orders with an ordering volume less than or equal to 500 units)

As a variant of this example, consider the following specification (again assuming that the specified request activity is part of the AssemblyMg role):

ACT REQUEST (EACH ; USsupplier, EUROsupplier ; NOT deliver (*material, quantity*))
{ STATUS RANGE
<IND> : NORM <**P**> < (*material* = steel) AND (rating(*material*) = poor)> +
SANC <NO> <NO>
<DEP MemBoardDirectors> : NORM <**O**> <NO> + SANC <**PU**> <reprimand>
NORMATIVE IMPACT
NORM <**I**> <*material* = steel>
}

The status range of this variant includes a DS statement, meaning that this request activity becomes obligatory for an agent owing the role AssemblyMg if it is requested by an agent owning the role MemBoardDirectors ("Member of Board of Directors"). The specification of the corresponding request activity of the MemBoardDirectors role could look like this:

ACT REQUEST
(EACH ; AssemblyMg ;
REQUEST (EACH ; USsupplier, EUROsupplier ; NOT deliver (*material, quantity*)))
{ STATUS RANGE
<IND> : NORM <**O**> <decided_by_board> + SANC <**PU**> <board_exclusion>
NORMATIVE IMPACT
NORM <**O**> <NO>
}

With that, the RNS schema offers the possibility to formulate "requests for requests for requests for . . .", that is, *nested requests*.

Sanctioning Activities. Activities of this type are specified as follows:

ACT SANCTION (*agent_id_list* ; *role_id_list* ; *activity_id* ; *norm_spec*)
{ STATUS RANGE *status_statement_list*
SANCTIONING IMPACT *sanction_specification_list* }

where *norm_spec* is a norm specification and *sanction_specification_list* is a list of sanction specifications, that is, a list of specifications of the form

SANC <*sanction_type*> <*sanction*>

The sanctioning impact part specifies all sanctions that "become reality" through the execution of the sanctioning activity. Here is a simple example of a sanctioning activity, based on the "deliver" activity specification above:

```
ACT SANCTION ( EACH ; EACH ; deliver ; NORM <O> <quantity ≤ 100> )
    { STATUS RANGE
        <IND> : NORM <P> <NO> + SANC <RE> <earn_bonus>
        <DEP RoleSpaceMg> : NORM <I> <NO> + SANC <PU> <withdraw_role>
      SANCTIONING IMPACT
        SANC <PU> <withdraw_role>
    }
```

The two occurrences of EACH indicate that the sanctioning activity concerns each agent and each role of which the activity with identifier "deliver" is part of. The IS statement says that an agent owning a role of which this sanctioning activity is part of is unconditionally permitted (i.e., may) to execute this sanction, and that she earns some bonus in the case she does (i.e., actually makes use of her permission). The DS statement says that the sanctioning activity may become subject to an unconditioned interdiction, namely, as the result of an "interdiction request" by an agent owning the role with identity RoleSpaceMg ("Role Space Manager"); violating such an interdiction is punished through the withdrawal of role ownership.

An example of a real-time occurrence (instantiation) of this sanction specification is

```
ACT SANCTION ( Dr_Meyer ; USsupplier ; deliver ; NORM <O> <quantity ≤ 100> )
```

A further example of a sanction specification illustrating the expressiveness of RNS is the following. Under the assumption that each norm violation is punished by the withdrawal of role ownership, the "most general" specification of a sanction activity that can be constructed is

```
ACT SANCTION ( EACH ; EACH ; EACH ; EACH )
    { STATUS RANGE
        <IND> : NORM <P> <NO> + SANC <NO> <NO>
      SANCTIONING IMPACT
        SANC <PU> <withdraw_role>
    }
```

saying that each agent owning a role of which this activity specification is part of is unconditionally permitted to sanction any norm violation through the withdrawal of role ownership.

Change Activities. Change activities affect the status range of activities. Three types of change activities are distinguished: DEL (delete), REP (replace), and ADD (add). The specification of these activities is as follows:

```
ACT ADD ( role_id_list ; activity_id_list ; status_statement )
    { STATUS RANGE status_statement_list
      [ STATUS IMPACT
        add status_statement ]
    }
```

```
ACT DEL ( role_id_list ; activity_id_list ; status_statement )
    { STATUS RANGE status_statement_list
      [ STATUS IMPACT
        delete status_statement ]
    }
ACT REP ( role_id_list ; activity_id_list ; status_statement_1 ; status_statement_2 )
    { STATUS RANGE status_statement_list
      [ STATUS IMPACT
        replace status_statement_1 by status_statement_2]
    }
```

The status impact parts are optional as they are of explanatory nature only. Here is an example of a specification of a change activity:

```
ACT REP
    ( USsupplier, EUROsupplier ; deliver ;
      <IND> : NORM <P> <NO> + SANC <NO> <NO> ;
      <IND> : NORM <O> <NO> + SANC <PU> <pay_fine> )
    { STATUS RANGE
        <IND> : NORM <P> <NO> + SANC <NO> <NO>
    }
```

An agent owning a role which includes this activity specification is permitted to replace, within each deliver activity being part of the roles USsupplier and EUROsupplier, the first status statement given in the activity header by the second one. (Note that the first IS statement in the header and the IS statement in the status range part are syntactically identical.)

Another example of change activity specification is the following:

```
ACT DEL
    ( USsupplier ; deliver ;
      <IND> : NORM <P> <NO> + SANC <NO> <NO> )
    { STATUS RANGE
        <IND> : NORM <P> <NO> + SANC <NO> <NO>
        <DEP USdirector> : NORM <I> <NO> + SANC <PU> <withdraw_role>
    }
```

This change activity concerns the deletion of the status statement

```
<IND> : NORM <P> <EACH> + SANC <EACH> <EACH>
```

attached to the deliver activity of the USsupplier role (as specified in the activity header). As can be seen from the status range of the delete activity specification, an agent owning the role USdirector may forbid this delete activity (i.e., is authorized to request to not execute it). Generally, RNS enables to formulate requests on change activities, and, reversely, it enables to formulate changes of status statements belonging to request activities.

Finally, here is an example of a specification of an add activity:

```
ACT ADD ( EACH ; EACH ;
        <DEP President> : NORM <I> <NO> + SANC <PU> <role_space_exclusion> )
    { STATUS RANGE
        <IND> : NORM <P> <NO> + SANC <NO> <NO>
    }
```

The owner of a role containing this activity specification is permitted to add the specified DS statement to every activity being part of any role (in the role space). Once added to an activity (more precisely, to the status range attached to an activity), an agent being in the role of President may unconditionally forbid this activity, where the consequence of violating this interdiction is the exclusion from the role space.

3 Discussion

A key feature of the RNS schema is its domain- and application independence. This means that the schema can be used to specify the kind and degree of autonomy which the agents should possess, no matter what specific application is considered and no matter in what domain this application runs. RNS is particularly suited for requirements elicitation and analysis, and thus its primary use lies in the early phase of agent-oriented systems development. More specifically, there are several potential advantages resulting from a use of RNS during requirements elicitation and analysis: it both enables and forces a developer to carefully reflect on the role of autonomy and autonomous behavior; it helps to avoid misunderstandings among analysts, stakeholders and programmers when communicating about agent autonomy; it facilitates the detection of potential faults caused by autonomous behavior; and, last but not least, it may serve as a basis for clarifying legal liability issues posed by agents acting autonomously on the behalf of human users.

Another key feature of RNS is that it is strongly expressive and enables a highly precise specification of agent autonomy. Expressiveness and precision derive from the following features:

- Through its concept of (positive and negative) sanctions RNS enables a developer to explicitly specify consequences of both norm-conforming and norm-deviating behavior. The importance of specifying these consequences results from the fact that autonomy, taken seriously, implies *autonomy against norms* [4] – an agent as an autonomous entity can not be guaranteed to act in accordance with all available norms under all circumstances.
- Through its concept of change activities RNS supports the explicit modeling and specification of potential *dynamic changes* in norms and sanctions and thus in behavioral autonomy.
- Through its concept of a status range RNS enables a developer to specify different normative impacts on the same activity. This makes it possible to cope with situations in which the normative status of an activity depends on the request context, that is, on who requested the activity under what

condition. With that, RNS allows to explicitly capture *context sensitivity* of norms and thus of autonomy.
- RNS supports the specification of complex activities through various activity reference constructs. While some possible reference constructs (e.g., "a request for requesting a certain resource handling activity") may be only of marginal interest in an application at hand, others (e.g., "a request for sanctioning a norm violation") may be of particular importance.
- As it is based on the role concept, RNS does not imply constraints on the type and structure of the individual agents. Instead, it enables a developer to abstract from architectural aspects of agency. This is of particular importance in view of open applications.

Further appealing features of the RNS schema are the following: it is based on a relatively simple and intuitively clear notation and syntax; it is fully neutral w.r.t. autonomy (i.e., it is neither biased in favor of nor against autonomy and so supports a developer in specifying any autonomy level she considers as appropriate); and it is grounded in sociological role theory.

It is important to see that RNS operates on the agent and role level, although it also may be of use for norms-based interaction protocol specification; however, this is an open issue that remains to be investigated. A difference between RNS-type specification and protocol-type specification seems to be in the potential normative impact of requests: in the case of RNS, a request may induce an obligation; against that, in the case of (speech act-based) interaction protocols it is typically assumed that an obligation is not induced by a request per se (i.e., from the pure fact that something is requested), but by the – explicit – acceptance of a request.

There are several approaches which are closely related to RNS in that they also aim at a norms-based specification of autonomous behavior [6, 7, 9, 2, 5]. As elucidated below, what makes RNS distinct from all these approaches is the expressiveness and precision with which it allows to capture autonomy.

An approach which shows several interesting parallels to RNS is described in [6]. The focus there is on norm compliance and on the question what motiviations an agent might have to comply with norms. Like RNS, this approach is based on the view that agents as autonomous entities may decide to not act in accordance with norms; moreover, similar to RNS this approach considers the issue of positive and negative sanctions. The main difference is that this approach does make several strong and in some sense restrictive assumptions on the cognitive structure and processes within the individual agents (e.g., by treating sanctions as the agents' goals and by defining autonomy in terms of motivations hold by agents). Against that, RNS does not make restrictive assumptions on "things occuring within agents", but concentrates on the role level.

Another approach showing interesting parallels to RNS is presented in [7]. This approach focuses distributed systems management through policies. A policy in this approach is understood as a behavior-influencing information being

located outside of the managers themselves, and is specified in terms of normative concepts (authorizations and obligations). Similar to RNS, this approach employs the role concept and supports a specification of context sensitivity. The main differences are that this approach does assume that agents always do behave norm-conforming (thus sanctioning is not considered), that complex activity specification is not supported, and that the specification of dynamic norm (and sanction) changes is not supported.

A logic-based approach related to RNS is described in [9]. This approach concentrates on collective agency and offers, similar to RNS, a normative system perspective. One important difference is that RNS, in contrast to this approach with its roots in deontic logic, does not rely on inter-definability of permissions and obligations (i.e., $P(x) =_{def} \neg O \neg x$). Another important difference is that this approach does neither consider the possibility of norm-deviating behavior nor the issue of dynamic norm change activities.

Other logic-based approaches related to RNS are described in [2] (dealing with norms-based coordination) and [5] (dealing with norms-based capturing of autonomous agents from a more general perspective). Like RNS, these approaches employ the concepts of permissions, obligations and interdictions and consider sanctions on norm-deviating behavior (though only negative sanctions). Unlike RNS, the approaches do not support the specification of dynamic changes in norms and sanctions and do not capture complex activity specification. Another difference is that these approaches are not role-based; instead, norms and sanctions are directly attached to agents and assumptions are made on agent-internal (cognitive) processes.

4 Conclusion

It should be clear that RNS in its current form leaves room for improvement. The two most critical deficiences of RNS we identify are the following. First, RNS does not support developers in explicitly specifying information and control relationships among roles such as generalization, aggregation, inheritance, peer, superior-subordinate, and so forth. Without support of such an explicit specification is it difficult (especially for large-scale applications) to obtain transparency of the overall system and its internal organizational structure. Second, RNS does not support developers in identifying and avoiding conflicts among norms (e.g., permission and interdiction of the same activity). Especially for large-scale applications such a support is extremely important as a means for avoiding poor system behavior resulting from normative conflicts. Both deficiences are of particular relevance w.r.t. a coherent and consistent system perspective and both require to extend RNS appropriately. What needs to be done in a first step thus is to define clear and useful conceptualizations of role-role relationships and normative conflicts. Encouraged by the above mentioned advantages of RNS we are currently concentrating on this first step. Moreover, we are also working on a software tool which supports RNS-based systems specification during require-

ments elicitation and analysis; a key service this tool is planned to offer is the automated detection of norm-based conflicts.

Acknowledgements. This work has been supported by Deutsche Forschungsgemeinschaft (DFG) under contract Br609/11-2. We would like to thank the reviewers for their valuable comments.

References

1. R. Alterman. Rethinking autonomy. *Minds and Machines*, 10(1):15–30, 2000.
2. M. Barbuceanu, T. Gray, and S. Mankovski. The role of obligations in multiagent coordination. *Journal of Applied Artificial Intelligence*, 13(2/3):11–38, 1999.
3. B.J. Biddle and E.J. Thomas, editors. *Role theory: Concepts and research*. John Wiley & Sons, Inc., New York, London, Sydney, 1966.
4. R. Conte, C. Castelfranchi, and F. Dignum. Autonomous norm acceptance. In J.P. Müller, M.P. Singh, and A. Rao, editors, *Intelligent Agents V. Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98)*, Lecture Notes in Artificial Intelligence Vol. 1555, pages 99–112. Springer-Verlag, 1999.
5. F. Dignum. Autonomous agents with norms. *Artificial Inteligence and Law*, 7:69–79, 1999.
6. F. Lopez y Lopez, M. Luck, and M. d'Inverno. Constraining autonomy through norms. In *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS'2002)*, 2002.
7. E. Lupu and M. Sloman. Towards a role based framework for distributed systems management. *Journal of Network and Systems Management*, 5(1):5–30, 1997.
8. D. Musliner and B. Pell (Cochairs). Agents with adjustable autonomy. Papers from the AAAI spring symposium. Technical Report SS-99-06, AAAI Press, Menlo Park, CA, 1999.
9. O. Pacheco and J. Carmo. A role based model for the normative specification of organized collective agency and agents interaction. *Journal of Autonomous Agents and Multi-Agent Systems*, 2002. to appear.