Capturing Agent Autonomy in Roles and XML

Gerhard Weiß
Institut für Informatik, TUM
Boltzmannstraße 3
85748 Garching, GERMANY
weissg@in.tum.de

Michael Rovatsos
Institut für Informatik, TUM
Boltzmannstraße 3
85748 Garching, GERMANY
rovatsos@in.tum.de

Matthias Nickles
Institut für Informatik, TUM
Boltzmannstraße 3
85748 Garching, GERMANY
nickles@in.tum.de

ABSTRACT

A key question in the field of agent-oriented software engineering is how the kind and extent of autonomy owned by computational agents can be appropriately captured. As long as this question is not answered convincingly, it is very unlikely that agent-oriented software (having "autonomy" as a real property rather than just a catchy label) gets broadly accepted in industry and commerce. In particular, in order to be of practical value an answer to this question has to come in form of concrete techniques which enable developers of agent-oriented software to precisely capture the scope of behavioral freedom and self-control they want to concede to a computational agent. This paper describes two such techniques. First, a formal schema called RNS for specifying the boundaries of autonomous agent behavior. This schema is conceptually grounded in sociological role theory, and employs the concepts of role, norm and sanction to capture agent autonomy. What makes RNS particularly valuable and distinct from related autonomy specification approaches is, among other things, its strong expressiveness and high precision. Second, a software tool called XRNS which enables developers to easily generate RNS-based autonomy specifications in XML format. Encoded in XML, these specifications are easily accessible to all stakeholders in an agent-oriented software under development, and can be even processed directly by XML enabled computational agents.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Requirements/Specifications—*languages, tools*

General Terms

Design, Theory

Keywords

agent-oriented software engineering, computational autonomy, specification, RNS, XRNS, XML

1. INTRODUCTION

Software engineering has always been targeted on the development of software whose behavior is fully determined and predictable under all circumstances. This does hold for both paradigms which have dominated the field so far: the classical paradigm of structure orientation with its focus on modularization, data abstraction, abstract data types, etc.; and the currently predominant paradigm of object orientation with its emphasis on messages, classes, inheritance, polymorphism, etc. Since some years a subfield is emerging in software engineering, known as agent-oriented software engineering, which is targeted on a significantly different type of software, namely, software possessing at least to some extent action and decision choice (e.g., see [4, 8, 15] for related proceedings and [9, 14] for surveys). This subfield's guiding principle is agent orientation, where an agent is understood as a computational entity capable of flexible, social, and autonomous behavior. Through its emphasis on autonomy as a key property of computational agency, agent orientation remarkably differs from both structure and object orientation in that it gives up the traditional request for a complete behavioral determination and predicability: within certain boundaries agents are empowered to act and decide under self-control. An obvious key question coming up with agent orientation is how the kind and the extent of autonomy owned by different agents can be tailored appropriately. Thereby "appropriately" means that agent autonomy should be neither unnecessarily cut down (as this would result in agents being not really distinct from ordinary objects) nor unnecessarily admitted (as this would result in an increased risk of undesirable or even chaotic behavior). To answer this question is perhaps the most critical precondition for a broad, industrial and commercial acceptance of agent-oriented software. In particular, in order to be of practical value this answer has to come in form of concrete techniques – methods, formalisms, tools, and so forth – which enable and support software developers in precisely capturing and describing the scope of self-control they want to concede to a computational agent in a given application.

This paper describes research which aims at developing such techniques. More specifically, the contribution of this paper is twofold. First, it presents a formal schema called RNS (standing for "Roles, Norms, Sanctions") which allows for a highly precise specification of a computational agent's autonomy. The basic view underlying RNS is that agents act as owners of roles in order to pursue their goals. As role owners, agents are exposed to certain norms (permissions, obligations, and interdictions), and through behaving in conformity with or in deviation from norms they become exposed to certain sanctions (rewards or punishments). Second, the paper describes a software tool called XRNS which is based on RNS and which enables developers to easily generate RNS-based autonomy specifications in XML format. XRNS is implemented in Java and is based on standard technology such as MySQL™ and JDBC™.

The paper is organized as follows. Sections 2 and 3 describe RNS and XRNS, respectively. Section 4 discusses these two techniques. Section 5 overviews related work. Section 6 concludes the paper with general considerations on the importance of autonomy engineering.

2. RNS

2.1 Basic Constructs

RNS, which is largely inspired by sociological role theory (e.g., [2]), requires to specify the autonomy of computational agents in terms of roles which are available to these agents and through which these agents can try to achieve their goals. 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] Conceptually, a role serves as a behavioral guideline which helps to achieve behavioral predictability without excluding behavioral freedom. Formally, a role is treated as a collection of activities, and for each role identifier, *role_id*, being included in `ROLE SPACE` there must be a role specification in the form

`ROLE` *role_id* **{** *activity_id_list* **}**

where *activity_id_list* is a list of identifiers of the activities being part of this role. For each activity identifier there must be a corresponding activity specification as detailed in section 2.2. Throughout this paper selected facets of an *agent-based supply chain management system* is considered as an application scenario, where the role space includes the following roles (among others not described here): USsupplier; EUROsupplier; AssemblyMg (= assembly manager); MemBoardDir (= member of the board of directors of the overall management system); and SpaceMg (= manager of the role space itself, being responsible for activities affecting the structure and contents of the role space).

RNS 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. By enabling a designer to explicitly specify sanctions, RNS takes care of the fact that generally (and in particular in open applications) agents as autonomous entities do not necessarily act in accordance with the available norms, but may ignore and violate them. Based on the distinction of different types of norms and sanctions, 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 RNS 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:

$$\langle status\_type\rangle \ \mathtt{:}\ \underbrace{\underbrace{\underline{\mathtt{NORM}}\ \langle norm\_type\rangle\ \langle condition\rangle}_{\text{norm specification}} + \underbrace{\underline{\mathtt{SANC}}\ \langle sanction\_type\rangle\ \langle sanction\rangle}_{\text{sanction specification}}}_{\text{norm-sanction pair}}$$

where *status_type* ∈ {`IND`, `DEP` *role_id*} discriminates among IS and DS statements, *norm_type* ∈ {**P**, **O**, **I**}, *condition* is a Boolean expression making it possible to formulate conditioned norms, *sanction_type* ∈ {**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*."

[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.

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.

2.2 Activities

As mentioned above, for each activity identifier included in a ROLE specification there must a corresponding activity specification. According to RNS, 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 RNS 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 activities concerning the management and consumption of resources and events. 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*), and run-protocol(*English-auction*). 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 ≤ 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* (resp. *agent_id*) indicates that *all* roles (resp. 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. Generally, the keyword EACH serves as a wildcard, and expressions including it are called *templates*.

Execution Requests. These activities are specified as follows:

```
ACT REQUEST activity_id
    ( 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 ManagerReq1
    ( EACH ; USsupplier, EUROsupplier ; NOT deliver ( material, quantity ) )
  { STATUS RANGE
      <IND> : NORM <P> < (material = steel) AND (rating(material) = poor)> +
              SANC <NO> <NO>
      <DEP MemBoardDir> : NORM <O> <NO> + SANC <PU> <reprimand>
    NORMATIVE IMPACT
      NORM <I> <material = steel>
  }
```

The keyword EACH in the activity header 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.) The status range includes an IS and a DS statement. The IS statement says that an agent as an assembly manager is permitted to request the non-execution of the deliver activity, provided that the material to be not delivered is steel the current price of steel is rated as poor. The DS statement means that this request activity becomes obligatory for an agent owing the role AssemblyMg if it is requested by an agent owning the role MemBoardDir ("Member of Board of Directors"); in case of not following this obligation, an assembly manager receives an official reprimand.

The specification of the corresponding request activity of the MemBoardDir role could look like this:

```
ACT REQUEST DirectorReq1
  ( EACH ; AssemblyMg ;
    REQUEST ManagerReq1
          ( EACH ; USsupplier, EUROsupplier ; NOT deliver ( material, quantity ) )
  )
  { STATUS RANGE
      <IND> : NORM <O> <decided_by_board> + SANC <PU> <board_exclusion>
    NORMATIVE IMPACT
      NORM <O> <NO>
  }
```

This example also indicates how *nested requests* (i.e., "requests for requests for requests for ...") look like in RNS notation.

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

```
ACT SANCTION activity_id ( 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. For an example of a sanctioning activity, again consider the supply chain management system. Here we assume that all sanctioning activities are bundled within a special role called RoleMg ("Role Manager"), with one of these activities being defined as follows:

```
ACT SANCTION DeliverPunish3 ( EACH ; EACH ; deliver ; NORM <O> <quantity ≤ 100> )
  { STATUS RANGE
      <IND> : NORM <P> <NO> + SANC <RE> <earn_bonus>
      <DEP MemBoardDir> : 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 the role SpaceMg 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 MemBoardDir; violating such an interdiction is punished through the withdrawal of role ownership.

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 as part of the role SpaceMg is

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

saying that each agent being a space manager 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 activity_id ( role_id_list ; activity_id_list ; status_statement )
    { STATUS RANGE status_statement_list
      [ STATUS IMPACT
        add status_statement ]
    }

ACT DEL activity_id ( role_id_list ; activity_id_list ; status_statement )
    { STATUS RANGE status_statement_list
      [ STATUS IMPACT
        delete status_statement ]
    }
```

```
ACT REP activity_id ( 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 replace 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>
      <DEP MemBoardDir> : NORM <I> <NO> + SANC <PU> <space_exclusion>
  }
```

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 of this specification by the second one. According to the DS statement, an agent owning the role MemBoardDir may forbid this delete activity (i.e., is authorized to request to not execute it); the consequence of violating this interdiction is the exclusion from the role space. Generally, RNS enables to formulate requests on change activities, and, reversely, it enables to formulate changes of status statements belonging to request activities.

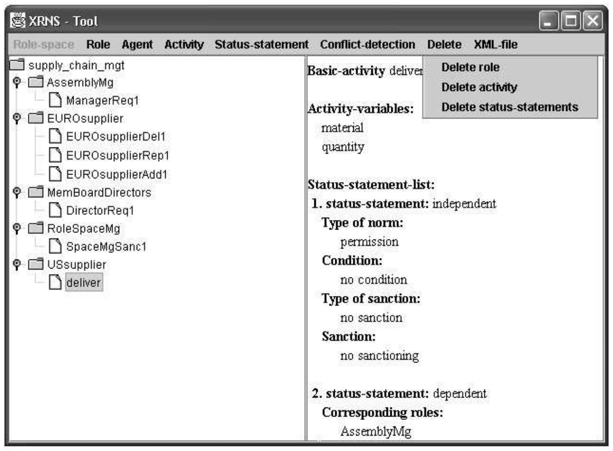
3. XRNS

XRNS (version 2.0) is a tool which supports a developer in creating RNS-based autonomy specifications and which transforms these specifications into a valid XML document. XRNS takes roles (rather than a complete role space or an individual activity) as the basic unit of transformation, that is, it generates an XML document for each role. The tool is implemented in Java^{TM} 1.3, and its GUI is realized with Java^{TM} SWING. MySQL^{TM} (version 3.23.42), the most popular open source database, is used to store and efficiently manage all data provided by a developer. The JDBC^{TM} API (version 1.12) is used to enable database access; JDBC^{TM} ("Java Data Base Connection") is the industry standard for connectivity between Java and a wide range of databases. As a JDBC enabling database driver, MM.MySQL (version 2.0.4) is used; this is the official and most popular JDBC driver for MySQL^{TM}. The transformation of the RNS-based specifications is realized via the Xerces Java Parser (version 1.4.0).

3.1 User Interface

The figure 1 gives an impression of the GUI of XRNS. The GUI consists of an "XRNS window" having a menu bar through which all available XRNS functions can be easily accessed in a pull-down style. In this figure, the left side of the XRNS window shows a part of the tree-structured role space for the supply chain management system described in 2.2; and the right side shows a mouse-click selected detail of this role space, namely, the deliver activity of the USsupplier role. At a glance, the functionality of XRNS is as follows:

- "Role space" menu: creation of a new role space.
- "Role" menu: creation of a new role.



supply_chain_mgt
AssemblyMg
ManagerReq1
EUROsupplier
EUROsupplierDel1
EUROsupplierRep1
EUROsupplierAdd1
MemBoardDirectors
DirectorReq1
RoleSpaceMg
SpaceMgSanc1
USsupplier
deliver

Basic-activity deliver

Activity-variables:
material
quantity

Status-statement-list:
1. status-statement: independent
Type of norm:
permission
Condition:
no condition
Type of sanction:
no sanction
Sanction:
no sanctioning

2. status-statement: dependent
Corresponding roles:
AssemblyMg

Figure 1: The GUI of XRNS.

- "Agent" menu: specification of agent identifiers.
- "Activity" menu: creation of a new activity of any of the four activity types distinguished according to RNS.
- "Status-statement" menu: creation of a new status statement list, and extension of an existing one.
- "Conflict-detection" menu: detection of conflicts among norms.
- "Delete" menu: deletion of a role, of an activity within a role, and of a status statement within an activity.
- "XML-file" menu: transformation of a role description to XML.

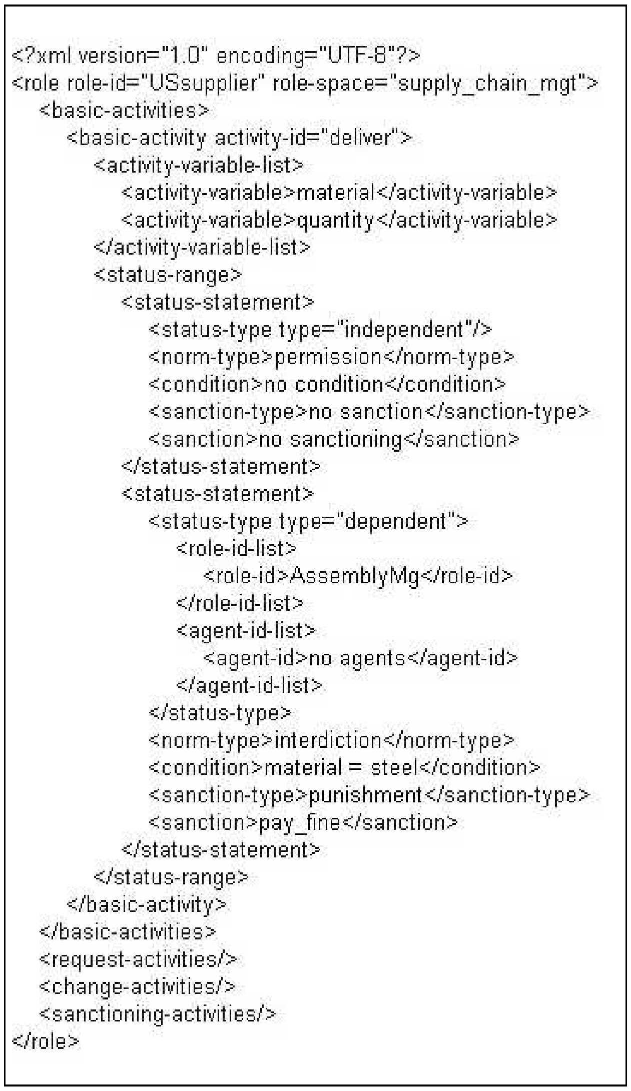
The conflict-detection function is under development, and is not further described here. Currently XRNS just detects some elementary conflicts among obligations and interdictions. (Conflict detection is very extensive as it requires e.g. a complete pairwise comparison of all available status statements, and this makes the use of efficient database technology mandatory.) Not included in the menu bar are implicit functionalities of XRNS, such as the verification of activity name consistency.

3.2 Generated XML Documents and DTD

The figure 2 gives an example of an XML document generated by XRNS. This document captures the RNS-based specification of the USsupplier role described in 2.[2] As this example shows, the XML tag naming is chosen so that it closely reflects the RNS notation.

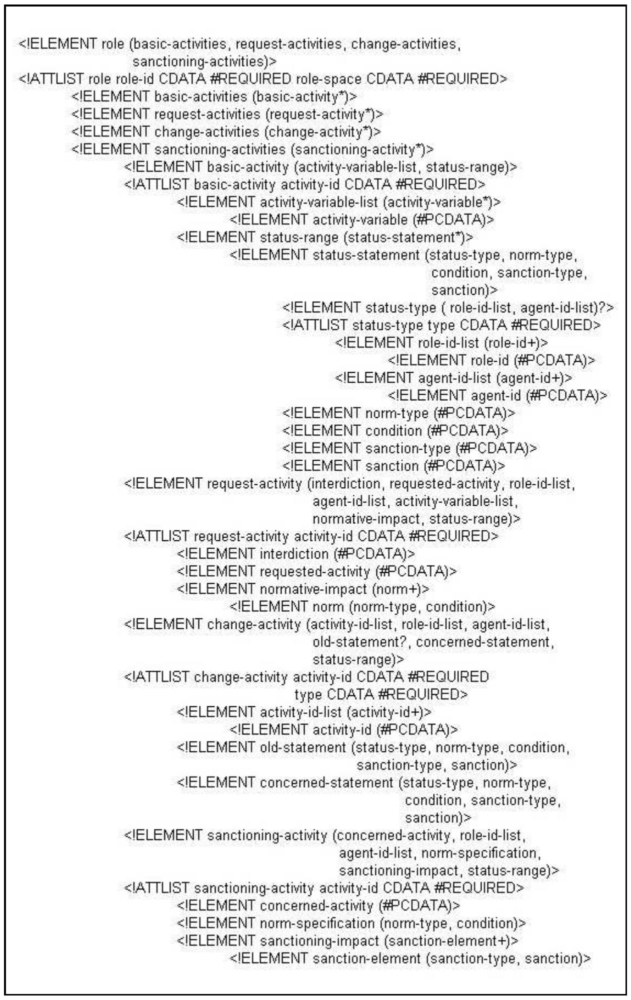
The DTD (Document Type Definition) defining the legal elements of XRNS-generated XML document is shown in the figure 3. This DTD is also used to ensure that the data provided by a developer are of legal type. Moreover, this DTD can be used by computational agents themselves to validate an XML encoded RNS-based role specification.

[2]Actually, a DS statement is omitted to keep the example document short. Note that this document corresponds to the activity description on the right side of the figure 1.



```
<?xml version="1.0" encoding="UTF-8"?>
<role role-id="USsupplier" role-space="supply_chain_mgt">
   <basic-activities>
      <basic-activity activity-id="deliver">
         <activity-variable-list>
            <activity-variable>material</activity-variable>
            <activity-variable>quantity</activity-variable>
         </activity-variable-list>
         <status-range>
            <status-statement>
               <status-type type="independent"/>
               <norm-type>permission</norm-type>
               <condition>no condition</condition>
               <sanction-type>no sanction</sanction-type>
               <sanction>no sanctioning</sanction>
            </status-statement>
            <status-statement>
               <status-type type="dependent">
                  <role-id-list>
                     <role-id>AssemblyMg</role-id>
                  </role-id-list>
                  <agent-id-list>
                     <agent-id>no agents</agent-id>
                  </agent-id-list>
               </status-type>
               <norm-type>interdiction</norm-type>
               <condition>material = steel</condition>
               <sanction-type>punishment</sanction-type>
               <sanction>pay_fine</sanction>
            </status-statement>
         </status-range>
      </basic-activity>
   </basic-activities>
   <request-activities/>
   <change-activities/>
   <sanctioning-activities/>
</role>
```

Figure 2: XML encoding of the USsupplier role, generated by XRNS.



```
<!ELEMENT role (basic-activities, request-activities, change-activities,
                sanctioning-activities)>
<!ATTLIST role role-id CDATA #REQUIRED role-space CDATA #REQUIRED>
        <!ELEMENT basic-activities (basic-activity*)>
        <!ELEMENT request-activities (request-activity*)>
        <!ELEMENT change-activities (change-activity*)>
        <!ELEMENT sanctioning-activities (sanctioning-activity*)>
                <!ELEMENT basic-activity (activity-variable-list, status-range)>
                <!ATTLIST basic-activity activity-id CDATA #REQUIRED>
                        <!ELEMENT activity-variable-list (activity-variable*)>
                                <!ELEMENT activity-variable (#PCDATA)>
                        <!ELEMENT status-range (status-statement*)>
                                <!ELEMENT status-statement (status-type, norm-type,
                                                            condition, sanction-type,
                                                            sanction)>
                                        <!ELEMENT status-type ( role-id-list, agent-id-list)?>
                                        <!ATTLIST status-type type CDATA #REQUIRED>
                                                <!ELEMENT role-id-list (role-id+)>
                                                        <!ELEMENT role-id (#PCDATA)>
                                                <!ELEMENT agent-id-list (agent-id+)>
                                                        <!ELEMENT agent-id (#PCDATA)>
                                        <!ELEMENT norm-type (#PCDATA)>
                                        <!ELEMENT condition (#PCDATA)>
                                        <!ELEMENT sanction-type (#PCDATA)>
                                        <!ELEMENT sanction (#PCDATA)>
                <!ELEMENT request-activity (interdiction, requested-activity, role-id-list,
                                            agent-id-list, activity-variable-list,
                                            normative-impact, status-range)>
                <!ATTLIST request-activity activity-id CDATA #REQUIRED>
                        <!ELEMENT interdiction (#PCDATA)>
                        <!ELEMENT requested-activity (#PCDATA)>
                        <!ELEMENT normative-impact (norm+)>
                                <!ELEMENT norm (norm-type, condition)>
                <!ELEMENT change-activity (activity-id-list, role-id-list, agent-id-list,
                                           old-statement?, concerned-statement,
                                           status-range)>
                <!ATTLIST change-activity activity-id CDATA #REQUIRED
                                          type CDATA #REQUIRED>
                        <!ELEMENT activity-id-list (activity-id+)>
                                <!ELEMENT activity-id (#PCDATA)>
                        <!ELEMENT old-statement (status-type, norm-type, condition,
                                                 sanction-type, sanction)>
                        <!ELEMENT concerned-statement (status-type, norm-type,
                                                       condition, sanction-type,
                                                       sanction)>
                <!ELEMENT sanctioning-activity (concerned-activity, role-id-list,
                                                agent-id-list, norm-specification,
                                                sanctioning-impact, status-range)>
                <!ATTLIST sanctioning-activity activity-id CDATA #REQUIRED>
                        <!ELEMENT concerned-activity (#PCDATA)>
                        <!ELEMENT norm-specification (norm-type, condition)>
                        <!ELEMENT sanctioning-impact (sanction-element+)>
                                <!ELEMENT sanction-element (sanction-type, sanction)>
```

Figure 3: DTD for validating XML documents produced by XRNS.

4. DISCUSSION

4.1 Benefits

RNS. The RNS schema is appealing for several reasons: it is based on a relatively simple notation and syntax; it is domain- and application independent; it is 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); it is grounded in sociological role theory; and, in particular, 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* [5] – an agent as an autonomous entity can not be guaranteed to always act in accordance with all available norms.

- 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 the case of open applications, because here it often is not known in advance which agents will enter into the running system.

XRNS. Because of the properties of XML, the XML-encoded specifications generated by XRNS are rather comfortable to read by humans and can be easily processed by computers. Moreover, available as XML documents these specifications can be exchanged across computing platforms, human languages, and applications, and can be used with a great number of available tools and utilities (e.g., for searching, extracting, translation, etc.). From these benefits in general, the following key advantages derive in particular:

- XRNS makes "autonomy" easily accessible to all stakeholders of the agent-oriented software under development, including analysts, designers, programmers, potential end users, and the customers themselves. Such an accessibility is invaluable w.r.t. the identification and refinement of the kind and level of autonomy an agent should possess, and fosters an early detection of misconceptions of autonomous behavior. XRNS thus supports the validation and verification of autonomy-related software properties through all phases of the developmental process.

- XRNS makes the autonomy specifications directly accessible to XML-enabled computational agents themselves. In particular, this makes it much easier to built open agent interaction forums in which the individual agents behave as desired. For instance, an "XML speaking agent" who participates in an open supply chain for the first time can orientate herself much easier and perhaps even without any human intervention if the roles she is allowed to fill within this supply chain are described in XML.

- XRNS supports the distributed development of autonomous software, as well as the reuse of RNS-based specifications across agent-oriented software systems and applications.

Another important feature of XRNS which contributes to its practical usage is that it is based on industrial standard software, such as JDBC^{TM} and MM.MySQL (resp. MySQL^{TM} Connector/J).

4.2 Deficiencies and Open Issues

Though RNS/XRNS are appealing for a number of reasons, they also leave room for improvement. The two most critical deficiences of RNS we identify are the following. First, RNS in its current version 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 yet 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.

Another important issue left for future research is the suitability of RNS/XRNS for interaction protocol specification and design. A remarkable difference between RNS-type specification and standard protocol-type specification is in the potential normative impact of requests: in the case of RNS, a request can 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., is not an inherent consequence of the utterance of a request), but only by the – explicit – acceptance of a request. We believe that both "request-obligation perspectives" do make sense, though probably in different contexts and on different levels of systems modeling. Obviously, an identification and characterization of these contexts and levels is needed, as well as an investigation of possibilities to unify both perspectives by extending RNS and/or standard interaction protocols accordingly.

As regards XRNS, a possible improvement is to use XML schemas instead of DTDs to gain advantages such as support for data types.

5. RELATED WORK

Among the available work on agent autonomy (e.g., see the collections [6, 12]), there are several approaches which are closely related to RNS in that they also aim at a norms-based specification of autonomous behavior [10, 11, 13, 1, 7]. 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 [10]. 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 [11]. 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 [13]. 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 [1, 7]. Unlike RNS, however, these approaches do neither support the specification of dynamic changes in norms and sanctions nor do they capture complex activity specification. Moreover, these approaches are not role-based; instead, norms and sanctions are directly attached to agents and assumptions are made on agent-internal (cognitive) processes.

As regards XRNS, a related work is described in [3]. Here a system called XRole for the XML-based definition of roles is described. XRole, however, does not concentrate on autonomy issues like XRNS does, but has its focus on agent interaction.

6. CONCLUDING REMARKS

To take the property of autonomy serious is essential to agent orientation and the field of agent-oriented software engineering. Among the commonly voiced objections to agent orientation are at least two which are directly related to this property: "everything an agent does can be done by an object as well", and "agents are just a risky source of chaotic system behavior." As long as these autonomy-related objections are not proved to be without substance, computational agency will hardly become a broadly accepted technology – despite the fact there *are* a number of seminal application domains which actually call for autonomous software (e.g., telecommunications, e/m-commerce, mobile robots, and supply chain management). Characteristic to these domains is that they are too complex to be fully predictable in all relevant aspects; as a consequence, software is needed which is autonomous enough to continue its work even under unforeseeable circumstances. Today it is common practice in the field to keep autonomous behavior minimal in order to avoid uncontrollable behavior, by applying rigid interaction protocols, preset organization patterns, and so forth [9]. We think, however, that this practice is not satisfactory, as it tends to make objects out of agents and to give up the benefits of autonomy too rashly. Instead, for the reasons mentioned above it is our feeling that the field needs to invest much more effort on the engineering of computational autonomy in order to ensure that agent orientation eventually becomes widely accepted. The approach described in this paper is intended to contribute to this effort.

Acknowledgements. The work reported here has been supported by Deutsche Forschungsgemeinschaft (DFG) under contract Br609/11-2.

7. ADDITIONAL AUTHORS

Christian Meinl (Institut für Informatik, TUM, Boltzmannstraße 3, 85748 Garching, meinl@in.tum.de).

8. REFERENCES

[1] 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.

[2] B.J. Biddle and E.J. Thomas, editors. *Role theory: Concepts and research.* John Wiley & Sons, Inc., New York, London, Sydney, 1966.

[3] G. Cabri, L. Leonardi, and F. Zambonelli. XRole: XML roles for agent interaction. In *Proceedings of the 3rd International Symposium "From Agent Theories to Agent Implementations (AT2AI-02)*, 2002. to appear.

[4] P. Ciancarini and M. Wooldridge, editors. *Agent-oriented software engineering. Proceedings of the First International Workshop (AOSE-2000).* Lecture Notes in Computer Science, Vol. 1957. Springer-Verlag, 2001.

[5] 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.

[6] R. Conte and C. Dellarocas, editors. *Social order in multiagent systems.* International Book Series on Multiagent Systems, Artificial Societies, and Simulated Organizations. Kluwer Academic Publishers, 2001.

[7] F. Dignum. Autonomous agents with norms. *Artificial Intelligence and Law*, 7:69–79, 1999.

[8] F. Giunchiglia, J. Odell, and G. Weiß, editors. *Agent-oriented software engineering III. Proceedings of the Second International Workshop (AOSE-2002).* Lecture Notes in Computer Science. Springer-Verlag, 2003. to appear.

[9] N.R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] G. Weiß. Agent orientation in software engineering. *Knowledge Engineering Review*, 16(4):349–373, 2002.

[15] M. Wooldridge, G. Weiß, and P. Ciancarini, editors. *Agent-oriented software engineering II. Proceedings of the Second International Workshop (AOSE-2001).* Lecture Notes in Computer Science, Vol. 2222. Springer-Verlag, 2002.