Specifying the Intertwining of Cooperation and Autonomy in Agent-based Systems

Gerhard Weiß*, Matthias Nickles*, Michael Rovatsos†, Felix Fischer*

*Computer Science Department
Technical University Munich
80290 München, Germany
{weissg,nickles,fischerf}@cs.tum.edu

†School of Informatics
University of Edinburgh
Edinburgh EH8 9LE, UK
mrovatso@inf.ed.ac.uk

Abstract

Cooperation and autonomy are two antagonistic core variables of agent-based systems, and a key challenge in designing such systems is to balance these variables appropriately. This article describes a specification schema called RNS2 that has been developed in response to this challenge. Being formally grounded in deontic logic and informally inspired by social role theory, RNS2 shows several useful features which together make it unique and distinct from related approaches. In particular, RNS2 is highly expressive and allows to specify the intertwining of cooperation and autonomy at a very precise level, and it facilitates the automated detection and resolution of autonomy-induced cooperation conflicts at design time. The application of RNS2 is illustrated in the context of an agent-based supply chain management system.

Keywords. Intelligent agents, multiagent systems, cooperation, autonomy, control, RNS2.

1 Introduction

Cooperation and autonomy are two antagonistic core design variables for agent-based systems. While cooperation is closely related to issues such as interdependence, sharing and integration, autonomy has much to do with issues such as independence, differentiation and separation (Keidel, 1995). At the heart of the antagonistic nature of cooperation and autonomy is that they concern different types of control: cooperation has its focus on exerting control on others and on acting

under *external control* as a member of a team; against that, autonomy has its focus on freely choosing between actions and on acting independently and under *internal control* (i.e., self-control). A highly critical and challenging task in building an agent-based system is to balance its internal and external control regime, as this regime directly affects the system's overall effectiveness and efficiency. This raises the strong need for techniques that support developers of agent-based systems in precisely and systematically specifying the *intertwining of internal and external control of agents*. Moreover, such techniques should enable developers to effectively deal with *autonomy-induced cooperation conflicts* between agents. Without the availability of such techniques it is highly unlikely that agent-based systems get broadly accepted as a standard technology in real-world industrial and commercial applications.

This article describes a formal specification schema called RNS2 that has been developed in response to this need. (RNS2 stands for "Roles, Norms, Sanctions", with the "2" indicating that RNS2 is based on the specification approach called RNS we proposed in (Nickles et al., 2002; Weiß et al., 2003).[1]) RNS2, which is formally grounded in deontic logic (e.g., Meyer and Wieringa, 1993) and informally inspired by social role theory (e.g., Biddle and Thomas, 1966), captures the notion of cooperation in terms of requests for carrying out, or refraining from, certain activities and allows to handle cooperation conflicts at design-time. The basic view underlying RNS2 is that agents are embedded in a social frame that regulates their behavior. This social frame, called role space, is composed of *roles* which are available to the agents and through which an agent can try to achieve individual and joint objectives either alone or in cooperation with other agents. An agent may own several roles and a role may be owned by several agents at the same time. Roles serve as a means for specifying desired behavior and for achieving behavioral predictability, but *not* as a means for making sure that agents do never exhibit unexpected and undesirable behavior. In particular, roles do not fully constrain individual behavior, but leave room for individuality (i.e., agents may fill in the same role differently by putting emphasis on different aspects). According to RNS2, a role consists of activities to which *norms* and *sanctions* are attached. As the owner of a role an agent is exposed to all norms and sanctions attached to the role-specific activities. RNS2 distinguishes three types of norms (permissions, obligations, and interdictions) and two types of sanctions (reward and punishment). While norms correspond to behavioral expectations held by agents against other agents in their capacity as role owners, sanctions correspond to potential consequences of norm-conforming and norm-violating behavior. With that, through norms and sanctions a system designer can explicitly specify the limits within which an agent is supposed to act autonomously.

[1] RNS2 significantly extends and improves RNS in several respects. A key difference is that RNS2 allows for the automated handling of cooperation conflicts, while RNS does not due to syntactic and semantic ambiguities.

The article is organized as follows. Section 2 gives an overview of related work. Section 3 describes the schema RNS2. Section 4 focusses on RNS2-enabled design-time identification and resolution of autonomy-induced cooperation conflicts between agents. In sections 3 and 4 key concepts and aspects of RNS2 are illustrated in the context of an agent-based supply chain management system. Section 5 discusses the benefits and shortcomings of RNS2. Section 6 concludes the article with more general considerations on balancing agent cooperation cooperation and autonomy.

2 The Schema RNS2

2.1 Basic Constructs

The following basic notation is used throughout this article. RNS2-specific syntactic keywords and special symbols are written in underlined TYPEWRITER FONT, and *italic font* is used to indicate RNS2-specific variables. The vertical bar, |, separates alternatives, and separated alternatives are enclosed by angle brackets, ⟨ and ⟩.

2.1.1 Role Space

RNS2 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 comma-separated list of character strings called role identifiers that uniquely identify roles. 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_spec_list* **}**

where *activity_spec_list* is a list of specifications of the activities being part of this role. Details on how to specify activities are shown in section 2.2.

Example. For an agent-based electronic supply chain management system the following four roles (among others not described here) are identified by the system designers: USsupplier ("supplier located in the US"); EUROsupplier ("supplier located in Europe"); AssemblyMg ("assembly manager"); MemBoardDir ("member of the board of directors of the overall management system"); and SpaceMg ("role

space manager", being responsible for activities affecting the structure and contents of the role space). In RNS2, this is captured by:

ROLE SPACE eSUPPLY
{ USsupplier , EUROsupplier , AssemblyMg , MemBoardDir , SpaceMg }

2.1.2 Domain- and Problem-Specific Variables

A variable that is specific to the application (i.e., the domain and the problem) under consideration is specified through

variable-id[*domain-elem-list*]

where *variable-id* is a variable identifier and *domain-elem-list* is either a comma-separated list of domain values (strings and numbers) or a numerical interval.

Example. Assume that the eSUPPLY system is restricted to specific kinds of material, namely, steel, silver, gold and platinum. Using RNS2, this restriction corresponds to the following variable specification:

material[steel, silver, gold, platinum]

As another example, assume that within eSUPPLY material can only be ordered and delivered up to a certain limit (say, 2000 units). This could be captured by the numerical-interval variable:

quantity[0 .. 2000] .

2.1.3 Status Range

RNS2 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 the case of norm conformity and violation. Norms can be coupled with conditions (i.e., Boolean expressions) so that it is possible to specify the circumstances under which they apply. Norms and sanctions serve as different means for specifying the boundaries of agent autonomy: while obligations and interdictions allow to state which activities are outside an agent's range of behavioral choice and control, permissions allow to state which activities are within this range. In other words, an agent may, but needs not to execute a permitted activity – the execution is neither mandatory (as in the case of an obligation) nor forbidden (as in the case of an interdiction). Whether or not an agent executes a permitted activity solely depends on his own decision about how to pursue his goals. By enabling a designer to explicitly specify sanctions, RNS2 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 (by chance or intentionally). 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. Status statements are said to be *attached to an activity*. A key feature of RNS2 is that it facilitates the explicit modeling and specification of *cooperation requests*, that is, requests for (refraining from) executing particular activities. This feature induces the distinction of two kinds of status statements attached to an activity:

- *Independent* status statements (indicated by the keyword IND) – these are status statements (and thus norm-sanction pairs) an agent becomes subject to as a direct consequence of entering the role to which the activity belongs.
- *Dependent* status statements (DEP) – these are status statements an agent as the owner of the role to which the activity belongs becomes subject to as a direct consequence of another agent's explict request for executing or refraining from this activity. In other words, dependent status statements are activited through ("depend on") cooperation requests.

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

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

where *status_type* $\in \{$IND, DEP *role_id*$\}$ discriminates between IS and DS statements, *norm_type* $\in \{\mathbf{P}, \mathbf{O}, \mathbf{I}\}$, *condition* is a Boolean expression making it possible to formulate conditioned norms, *sanction_type* $\in \{\mathbf{RE}, \mathbf{PU}\}$, and *sanction* is an identifier referring to 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* $= \{\mathbf{P}\}$: "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 (assume that the DS statement is attached to activity α and that this activity is part of role ρ):

- *norm_type* = {**P**}: "If an agent owning the role *role_id* requests to execute activity α from an agent owning role ρ, then the requested agent is *permitted* (by the requesting agent and through the request) to execute α provided that the condition *condition* is fulfilled. The sanction associated with this permission is of type *sanction_type* and is given by *sanction*." Hence, in this case the cooperation request induces a conditioned permission or, to say it the other way round, the conditioned permission is assigned (or "activated") through the cooperation request.

- *norm_type* = {**O**}: "If an agent owning the role *role_id* requests to execute α, then the requested agent is *obliged* to execute α 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* to 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 cooperation requests (e.g., from different agents) do have different normative and sanctioning impacts on a requested agent.

Examples. Assume that the status statement

<IND> : NORM <**P**> <NO> + SANC <NO> <NO>

is attached to the activity "deliver material" which is part of the role USsupplier. (A formal specification of a "deliver" activity is provided in 2.2). In that case, each agent acting as an USsupplier is permitted (as indicated by **P**) to carry out this

activity (i.e., to deliver material) without any restriction (as *condition* is NO) and with no sanction coupled to this permission (as *sanction type* is NO).[2] As this is an independent status statement (IND), an agent becomes subject to this permission automatically when entering the role USsupplier. As a second example, assume that

<DEP AssemblyMg> : NORM <**I**> <material = steel> + SANC <**PU**> <pay_fine>

is a further status statement attached to the deliver activity of USsupplier. As indicated by "<DEP AssemblyMg>", this status statement depends on a cooperation request by an assembly manager, and as indicated by "<**I**><material = steel>", this is a request to cooperate by *not* delivering steel. Moreover, this status statement says that a violation of this request to not deliver (i.e., of the interdiction to deliver) results in a punishment (**PU**) in form of a fine to be payed (as indicated by the identifier "pay_fine").

2.2 Activities

As said above, each role is specified through a list of activities. According to RNS2, four types of activities are distinguished:

- Basic activities, that is, resource and event handling activities (*Type I*).
- Cooperation requests, that is, requests for (not) 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 any of the types II, III, and IV. This means, in particular, that RNS2 allows to formulate crossed and self-referential constructs such as requests for cooperation requests, requests for sanction and norm changes, changes of norms attached to 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, called *referential constructs*, are provided below. In the following, the four activity types are described in detail. Figure 1 overviews the RNS2-specific activity types.

[2] Generally, 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).

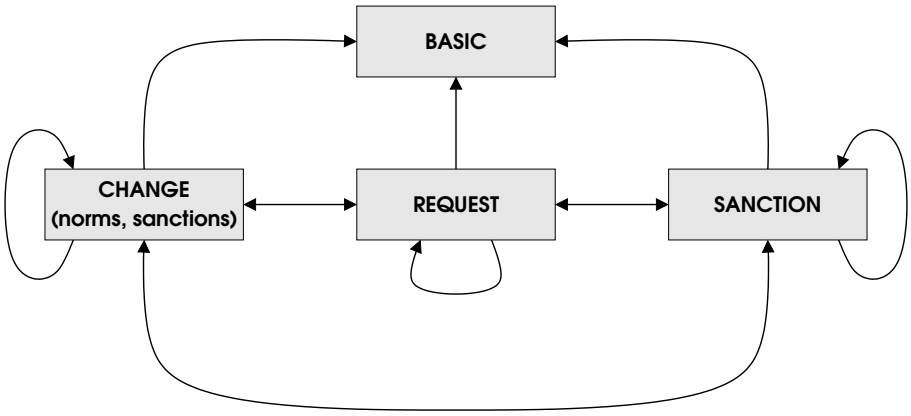

Figure 1: RNS2 activity types and their relationships.

2.2.1 Basic Activities

Basic 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 RNS2 specification of this type of activities has the general form

ACT BASIC *activity_id* (*variable_id_list*)
{ VAR *variable_spec_list* ;
STATUS RANGE *status_statement_list* }

where *variable_id_list* is a comma-separated list of variable identifiers and *variable_spec_list* is a comma-separated list of variable specifications (namely, of all variables employed in this activity). The first line of any activity specification, i.e., "ACT ... (...)", is called the *activity header*, and the part enclosed in {.} is called the *activity body*.

Example. Assume that one of the basic activities of the role USsupplier is:

ACT BASIC deliver (material , quantity)
{ VAR
material[steel, silver, gold, platinum] ,
quantity[0 .. 2000] ;
STATUS RANGE
<IND> : NORM <**P**> <NO> + SANC <NO> <NO>
<DEP AssemblyMg> : NORM <**I**> <material = steel> + SANC <**PU**> <pay_fine>
<DEP EACH> : NORM <**O**> <quantity ≤ 100> + SANC <**PU**> <withdraw_role>
}

The first and the second status statement already have been explained above (see the examples at the end of 2.1.3). The third status statement, "<DEP EACH> ..."

says that a request from any agent – no matter what role the requesting agent plays (as indicated by the keyword EACH) – to deliver *quantity* units of *material* must (**O**) be fulfilled, provided that the requested quantity is not above 100.[3] 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 this obligation.

2.2.2 Cooperation Requests

Cooperation requests are specified as follows:

```
ACT REQUEST activity_id
    ( ⟨ agent_id_list | EACH ⟩ ;
      ⟨ role_id_list | EACH ⟩ ;
      ⟨ NOT activity_id | activity_id (variable_id_list) ⟩ )
    { VAR variable_spec_list ;
      STATUS RANGE status_statement_list
      NORMATIVE IMPACT norm_specification_list }
```

The activity header says that cooperation 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*. Thereby *agent_id_list* and *role_id_list* are lists of comma-separated agent identifiers and role identifiers, respectively. The header also identifies the activity being subject to the cooperation request (*activity-id*). 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). The activity body consists of a status range part (STATUS RANGE, as defined above) and a normative impact part (NORMATIVE IMPACT). The normative impact part is a list of conditioned norms which the cooperation request imposes on the requested activity.

Examples. 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 ManagerDeliverReq1
    ( MrsMeyer ; USsupplier ; deliver ( material, quantity ) )
    { VAR
        material[steel, silver, gold, platinum] ,
        quantity[0 .. 2000] ;
      STATUS RANGE
        <IND> : NORM <P> <NO> + SANC <NO> <NO>
      NORMATIVE IMPACT
        NORM <O> <quantity ≤ 100> }
```

The activity header says that this cooperation request can be directed toward the US supplier Mrs. Meyer w.r.t. her deliver activity. In other words, an assembly manager

[3] Generally, the keyword EACH used as an instantiation of *role_id* matches *all* roles within the role space under consideration.

can request Mrs. Meyer in her capacity as a US supplier to deliver a certain quantity (up to 2000 units) of some material (steel, silver, gold or platinum). According to the status range, an assembly manager is permitted to carry out this cooperation request without any restriction. Moreover, no positive or negative sanction is associate with this permission. According to the normative impact part, it is obligatory for the requested agent (i.e., Mrs. Meyer) to carry out the requested deliver activity, provided that the deliver quantity is not greater than 100.[4] (Note that the induced conditioned norm, i.e., <**O**> <quantity $\leq$ 100>, is part of a DS statement within the status range of the deliver activity of the role USsupplier.)

As a second example, assume that the following request activity specification is also part of the role AssemblyMg:

ACT REQUEST ManagerNotDeliverReq1
(EACH ; USsupplier, EUROsupplier ; NOT deliver)
{ VAR
material[steel, silver, gold, platinum] ;
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 to not execute the deliver activity, provided that the material to be not delivered is steel and the current price of steel is rated as poor. (In other words, this IS statement says that an assembly manager is permitted to request a US supplier to not deliver steel, if the current pricing of steel is 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 the case of not following this obligation, an assembly manager receives an official reprimand. Within the role MemBaordDir several cooperation requests can correspond to this DS statement. An example of such a corresponding cooperation request by a member of the board of directors is:

[4]Because of this condition it would be also possible to specify the variable "quantity" as quantity[0 .. 100].

ACT REQUEST DirectorReqOfRequest1
(EACH ; AssemblyMg ;
ManagerNotDeliverReq1 (EACH ; EUROsupplier ; NOT deliver))
{ STATUS RANGE
<IND> : NORM <**O**> <decided_by_board> + SANC <**PU**> <board_exclusion>
NORMATIVE IMPACT
NORM <**O**> <NO> }

According to the activity head, this cooperation request can be directed to each agent acting as an assembly manager. The requested activity (ManagerNotDeliverReq1) is also a cooperation request, and so this example also illustrates how *nested requests* (i.e., "requests for requests for ...") can be formulated in RNS2. Note that within the request activity "DirectorReqOfRequest1" the requested activity "ManagerNotDeliverReq1" is restricted to European suppliers, while in general "ManagerNotDeliverReq1" concerns US suppliers, too.

2.2.3 Sanctioning Activities

Through sanctioning activities it is possible to specify explicitly what agents are empowered to sanction other agents in response to norm-conform and norm-violating behavior. Activities of this type are specified as follows:

ACT SANCTION *activity_id*
(*agents* ; *roles* ; *activities* ; *statement*)
{ STATUS RANGE *status_statement_list* }

where *agents*, *roles*, *activities* and *statement* are defined as

⟨ *agent_id_list* | EACH ⟩
⟨ *role_id_list* | EACH ⟩
⟨ *activity_id_list* | EACH ⟩
⟨ *status-statement* | EACH ⟩

respectively, with *agent_id_list*, *role_id_list* and *status-statement* being defined as above and *activity_id_list* being a comma-separated list of activity identifiers. The sanction included in *status-statement* (or *all* sanctions in the case of EACH) is the one whose execution is empowered through the sanctioning activity. In other words, the sanction included in the status statement is the one that "becomes reality" through the execution of the sanctioning activity.

Examples. Assume that all sanctioning activities are bundled within a special role called SpaceMg ("role space manager"), and that one of the sanctioning activities is defined as:

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

According to the activity head, this sanctioning activity concerns each agent playing any role which includes the activity "deliver". Moreover, the status statement included in the activity head uniquely identifies the type of sanction covered by this sanctioning activity: to punish (through widthdrawing role ownership) the violation of the obligation to fulfill deliver requests with an order volume ≤ 100. Now consider the status range of this sanctioning activity. The IS statement says that an agent owning the role SpaceMg is unconditionally permitted to execute this sanction, and that he earns some bonus in the case he does (i.e., actually sanctions). The DS statement says that the sanctioning activity may become subject to an unconditioned interdiction as the result of an "interdiction request" by a member of the board of directors; a role space manager violates this interdiction at the risk of being punished by the withdrawal of role ownership.

As a more hypothetical example that illustrates the expressiveness of RNS2, assume that the following sanctioning activity is part of the role SpaceMg:

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

According to this activity, each agent owning the role SpaceMg is unconditionally permitted to sanction every norm conformance/violation in accordance to the sanction attached to the norm.

2.2.4 Change Activities

Change activities affect the status range of activities. Three types of change activities are distinguished: delete (DEL), replace (REP) and add (ADD) a status statement being part of the status range of an activity. Change activities are specified as follows:

```
ACT CHANGE activity_id
    ( roles ; activities ; change-cmd )
    { STATUS RANGE status_statement_list }
```

where *roles* and *activities* are as defined above and *change-cmd* is defined as

```
⟨ ADD status-statement |
  DEL status-statement |
  REP status-statement BY status-statement ⟩
```

Change activities imply the distinction between the *initial* status range (i.e., the status range at design time) and potential *future* status ranges (i.e., status ranges that derive from the initial status range through change activities) of an activity. The set of all potential future status ranges of an activity $\mathcal{A}$, together with initial status range of $\mathcal{A}$, is called the *status range set* of $\mathcal{A}$ and is denoted by $SRS(\mathcal{A})$ in the sequel. ($SRS(\mathcal{A})$ is finite, if the number of change activities is so.)

Example. Assume that the following change activity specification is part of the role SpaceMg:

```
ACT CHANGE
    ( USsupplier, EUROsupplier ; deliver ;
      REP <IND> : NORM <P> <NO> + SANC <NO> <NO>
      BY  <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>
    }
```

According to the activity head, an agent acting as a role space manager is permitted to replace the IS statement "NORM <**P**> ..." by the IS statement "NORM <**O**> ..." within the deliver activity being part of the roles USsupplier and EUROsupplier. According to the DS statement in the status range, an agent owning the role MemBoardDir may forbid this change activity (i.e., each member of the board of directors is authorized to request to not execute it); the consequence of violating this interdiction is the exclusion from the role space. Generally, RNS2 enables to formulate requests on change activities, and, reversely, it enables to formulate changes of status statements belonging to request activities.

3 RNS2-enabled Automated Conflict Handling

RNS2 allows to formally distinguish three fundamental classes of autonomy-induced cooperation conflicts in agent systems and, in particular, to detect and resolve them already at design time.

3.1 Norm-based Conflicts

Norm-based conflicts are cooperation conflicts that result from inconsistent status statements. More precisely, let $\mathcal{S}_1$ and $\mathcal{S}_2$ refer to two status statements that are part of the status range of an activity of any type ("basic", "request", "sanctioning", "change"), and let $NT(\mathcal{S}_i)$ and $NC(\mathcal{S}_i)$ denote the *norm type* and the *norm condition* of $\mathcal{S}_i$, respectively ($i \in \{1,2\}$). $\mathcal{S}_1$ and $\mathcal{S}_2$ are said to constitute a norm-based conflict if (i) and (ii) do hold:

(i) $NT(\mathcal{S}_1) \neq NT(\mathcal{S}_2)$, that is, $\mathcal{S}_1$ and $\mathcal{S}_2$ show one of the following three norm constellations ($i, j \in \{1,2\}$ with $i \neq j$):

- $NT(\mathcal{S}_i) = \mathbf{O}$ and $NT(\mathcal{S}_j) = \mathbf{I}$ ("*OI conflict*")
- $NT(\mathcal{S}_i) = \mathbf{P}$ and $NT(\mathcal{S}_j) = \mathbf{O}$ ("*PO conflict*")
- $NT(\mathcal{S}_i) = \mathbf{P}$ and $NT(\mathcal{S}_j) = \mathbf{I}$ ("*PI conflict*")

(ii) it can happen that $NC(\mathcal{S}_1)$ and $NC(\mathcal{S}_2)$ evaluate to TRUE at the same time (i.e., both $\mathcal{S}_1$ and $\mathcal{S}_2$ are valid simultaneously).

PO and **PI** conflicts are *weak* in the sense that their incidence during runtime can be suppressed by the agents. This is because a permission implies behavioral choice

on the side of the agents, and a permitted activity may be executed (which would be in accordance with an obligation to carry out that activity) or may not be executed (which would be in accordance with an interdiction). Against that, **OI** conflicts are *hard*, as neither obligations nor interdictions leave room for action choice.

The detection of norm-based conflicts reduces to a pairwise comparison of status statements. For a given RNS2 specification the following pseudo-notation algorithm can be applied to detect all norm-based conflicts *at design time*:

```
for each role R
{ for each activity A ∈ R
  { for each status range SR ∈ SRS(A)  /* SRS = status range set */
    { for each S1 ∈ SR
      { for each S2 ∈ SR \ S1
        { if  NT(S1) ≠ NT(S2)
          then if NC(S1) = NC(S2) = TRUE
               then test whether NC(S1) and NC(S1)
                    may be fulfilled simultaneously } } } } }
```

According to this algorithm, at most $l \cdot m \cdot n^2$ tests are required, where l is the total number of activities for all roles, m is an upper bound for the cardinality of the status range sets (i.e., for $|SRS(\mathcal{A})|$), and n is an upper bound for the number of status statements included in the activities' status ranges. The complexity of the single tests depends on the mathematical logic used to formulate the norm conditions. In particular, if propositional logic is used then the algorithm is guaranteed to terminate for all RNS2 specifications (as this logic is decidable), and if first-order predicate logic is used then the algorithm is guaranteed to terminate for all RNS2 specifications that are free of norm-based conflicts. With that, RNS2 allows to map the problem of detecting norm-based conflicts to the well known decidability problem in mathematical proof theory.

Examples. The two DS statements within the status range of the above defined basic activity "deliver" constitute an **OI** conflict, as they may be activated simultaneously. The two status statements included in the status range of the above defined cooperation request activity "ManagerNotDeliverReq1" constitute a **PI** conflict.

3.2 Sanction-based Conflicts

Sanction-based conflicts are cooperation conflicts that are induced by sanctioning activities concerning the same norm-conform or norm-violating behavior. More precisely, let

ACT SANCTION *activity_id_1* (*agents_1* ; *roles_1* ; *activities_1* ; *statement_1*) { ... }

ACT SANCTION *activity_id_2* (*agents_2* ; *roles_2* ; *activities_2* ; *statement_2*) { ... }

be two sanctioning activities referred to as $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. $\mathcal{A}_1$ and $\mathcal{A}_2$ are said to constitute a sanctioning conflict if (i) to (iv) do hold:

(i) *agents_1* $\cap$ *agents_2* $\neq \emptyset$

(ii) *roles_1* $\cap$ *roles_2* $\neq \emptyset$

(iii) *activities_1* $\cap$ *activities_2* $\neq \emptyset$

(iv) *statement_1* $\cap$ *statement_2* $\neq \emptyset$

where $\cap$ is the standard set intersection with $x \cap \underline{\texttt{EACH}} = x$. If $\mathcal{A}_1$ and $\mathcal{A}_2$ belong to the same role, then the conflict is called an *intra-role* conflict, otherwise it is called an *inter-role* conflict. (Against that, norm-based conflicts are always intra-role conflicts.) According to (iv) two sanctioning activities are *not* conflictive if they sanction conformity to, or violation of, the same norm *differently*. For instance, assume that the status range of a basic "deliver" activity includes the two status statements

<DEP AssemblyMg> : NORM <**I**> <material = steel> + SANC <**PU**> <pay_fine>
<DEP AssemblyMg> : NORM <**I**> <material = steel> + SANC <**PU**> <withdraw_role>

If both status statements are activated through the corresponding cooperation requests, delivering steel violates the interdiction in both statements. As an effect, two different sanctions are applicable. According to (iv), the corresponding sanctioning activities are *not* in conflict although they concern the same "offense", namely, the violation of an interdiction. Analogously, according to (iv) *none* of the following two pairs of status statements induces a sanction-based conflict:

<DEP AssemblyMg> : NORM <**I**> <material = steel> + SANC <**PU**> <pay_fine>
<DEP SpaceMg> : NORM <**I**> <material = steel> + SANC <**PU**> <pay_fine>

and

<DEP AssemblyMg> : NORM <**I**> <material = steel> + SANC <**PU**> <pay_fine>
<DEP SpaceMg> : NORM <**I**> <material $\neq$ gold> + SANC <**PU**> <withdraw_role>

If it should be desirable in a given application to treat these kinds of "multiple sanctioning" as conflicts, then this can be easily realized by replacing (iv) through the refined condition (iv)*:

(iv)* $NT(\textit{statement\_1}) = NT(\textit{statement\_2})$ and
$NC(\textit{statement\_1})$ and $NC(\textit{statement\_2})$ may be TRUE at the same time

where NT and NC are as defined above.

The following algorithm can be applied at design time to detect all intra- and inter-role sanctioning conflicts (let SSA denote the set of all sanctioning activities in all roles):

```
for each sanctioning activity A1 ∈ SSA
{ for each sanctioning activity A2 ∈ SSA \ A1
  { test whether A1 and A2 fulfill the conditions (i) to (iv)/(iv)* } }
```

As with norm-based conflicts, the (semi-)decidability of sanction-based conflicts (and thus the efficiency with which these conflicts can be identified) depends on the type of mathematical logics used for formulating the norm conditions.

3.3 Change-based Conflicts

Change-based conflicts are cooperation conflicts that are induced by change activities. Let $\mathcal{S}$ and $\mathcal{T}$ be two status statements and let

ACT CHANGE *activity_id_1* (*roles_1* ; *activities_1* ; *change-cmd_1*) { ... }

ACT CHANGE *activity_id_2* (*roles_2* ; *activities_2* ; *change-cmd_2*) { ... }

be two change activities referred to as $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. $\mathcal{A}_1$ and $\mathcal{A}_2$ constitute a change-based conflict if:

(i) *roles_1* $\cap$ *roles_2* $\neq \emptyset$

(ii) *activities_1* $\cap$ *activities_2* $\neq \emptyset$

(iii) *change-cmd_1* and *change-cmd_2* show one of the following three constellations:

- *change-cmd_1* = "ADD $\mathcal{S}$" and *change-cmd_2* = "DEL $\mathcal{S}$"
- *change-cmd_1* = "DEL $\mathcal{S}$" and *change-cmd_2* = "REP $\mathcal{S}$ BY $\mathcal{T}$"
- *change-cmd_1* = "ADD $\mathcal{S}$" and *change-cmd_2* = "REP $\mathcal{S}$ BY $\mathcal{T}$"

If $\mathcal{S}$ and $\mathcal{T}$ belong to the same role, then the conflict is an *intra-role* conflict; otherwise, it is an *inter-role* conflict.

An algorithm for detecting all change-based conflicts at design time is the following (let SCA denote the set of all change activities):

```
for each change activity A1 ∈ SCA
{ for each change activity A2 ∈ SCA \ A1
  { test whether A1 and A2 fulfill the conditions (i) to (iii) } }
```

This algorithm terminates after $n \cdot (n-1)$ tests, where n is the total number change activities.

3.4 Strategies for Resolving Cooperation Conflicts

Design-time conflict detection enables design-time conflict resolution. One approach to resolve conflicts is to *modify* the roles and the activities so that the conflicts disappear in a system specification.[5] Another, very powerful approach offered by RNS2 is to apply the following domain- and problem-independent resolution strategies:

- "*Norm ordering*" (N-ordering for short) – define a preference order, $\prec_N$, on the three norms **I**, **O** and **P**, determining which of two norms overrules the other in the case of a conflict. This ordering can be *partial* (e.g., "$\mathbf{I} \prec_N \mathbf{O}$ and $\mathbf{P} \prec_N \mathbf{O}$") or *total* (e.g., "$\mathbf{I} \prec_N \mathbf{O} \prec_N \mathbf{P}$").

[5] A general difficulty with such modifications is that they run the risk of resulting in specifications that do not reflect the real system requirements.

conflict types		resolution strategies			
		N-ordering	R-ordering	A-ordering	S-ordering
norm-based		√			√
sanction-based	intra-role			√	
	inter-role		√		
change-based	intra-role			√	
	inter-role		√		

Table 1: Conflict types and resolution strategies. "√" indicates which conflict types can be resolved through which resolution strategy.

- "*Role ordering*"(R-ordering) – define a precedence order, $\prec_R$, on the roles, determining which of two roles involved in a conflict dominates the other. R-ordering can be partial or total.
- "*Activity ordering*" (A-ordering) – define a precedence order, $\prec_A$, on conflicting activities (e.g., on the cooperation requests or the change activities), determining which of two conflicting activities is preferred to the other. A-ordering can be *complete* (i.e., all pairs of conflicting activities are ordered) or *incomplete*.
- "*Status statement ordering*" (S-ordering) – impose a preference order, $\prec_S$, on conflicting status statements. This ordering can be complete or incomplete.[6]

These strategies differ in the level of concreteness and specificity at which they resolve conflicts. This level increases from N-ordering over R-ordering and A-ordering to S-ordering. Moreover, as summarized by table 1 the strategies differ in the types of conflicts they resolve. As this table also shows, it is possible to resolve *all* norm-, sanction- and change-based cooperation conflicts through applying the four strategies in appropriate combination.

4 Discussion

RNS2 offers several desirable key features. First, RNS2 is strongly expressive and allows to capture cooperation, autonomy and the interplay of these two design variables at a highly precise level. Expressiveness and precision derive from the following features:

- Through its concept of (positive and negative) sanctions RNS2 enables a developer to explicitly specify consequences of both norm-conforming and norm-deviating behavior. The importance of specifying these consequences results

[6]In the case of two conflicting DS statements, S-ordering corresponds to an ordering of the corresponding cooperation requests.

from the fact that autonomy, taken seriously, implies *autonomy against norms* (Conte et al., 1999) – 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 RNS2 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 RNS2 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, RNS2 allows to explicitly capture *context sensitivity* of norms and thus of autonomy.

- RNS2 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, RNS2 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.

- RNS2 is neutral w.r.t. autonomy, that is, it is neither biased in favor of nor against autonomy and so supports a developer in specifying autonomy at any level considered as appropriate.

Second, it facilitates the automated detection and resolution of potential cooperation conflicts at design time. The RNS2-specific conflict resolution strategies constitute an effective alternative to the high-level run-time conflict-resolution strategies – negotiation, mediation, and arbitration – typically considered in the agents area (see, e.g., Müller and Dieng, 2000; Tessier et al., 2000). To have an alternative to these high-level strategies is important, because they are not always applicable in real-world contexts (e.g., due to the lack of communication bandwidth or due to the lack of knowledge needed for identifying potential compromises).

Third, RNS2 is particularly qualified for requirements elicitation and analysis. Using RNS2 in the early phase of agent-oriented systems development has several advantages: in particular, 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, and it can be used as a basis for clarifying security issues and issues of legal liability posed by agents acting autonomously (on behalf of human users).

Fourth, RNS2 is domain- and application-independent. This means that RNS2 can be used to specify the kind and degree of cooperation and autonomy the agents should show in their behavior, regardless of the type of application and the domain in which the application runs.

RNS2 leaves room for further improvement. We think a deficiency of RNS2 is the lack of support of explicitly capturing advanced relationships between the individual roles, such as generalization, aggregation, peer and inheritance. Such a support is necessary to enable designers and developers to build agent-based systems that are structurally and functionally transparent. In solving this deficiency, the broad body of available work on the organizational design of agent-based systems is likely to be very helpful (e.g., Bond, 1990; Castelfranchi, 1995; Do et al., 2003; Drogoul and Collinot, 1998; Kolp et al., 2001; Moss et al., 1996; Zambonelli et al., 2001).

Another main deficiency we see is that RNS2 in its current form does not support the specification of temporal constraints on cooperation and autonomy. For instance, RNS2 does not allow to state explicitly that two activities (belonging to the same or to different roles) should not be executed at the same time, that a cooperation request expires after a certain period of time, and that a norm or a status statement attached to some activity is only valid within a certain time interval. Various approaches to continuous and discrete time/event modeling are available in traditional (distributed) systems theory, and we are confident that RNS2 can be extended appropriately with the help of these approaches.

5 Related Work

To our knowledge RNS2 is the first available formalism designed to explicitly capture the complex interplay of agent cooperation and autonomy. There are, however, several approaches that are closely related to RNS2 in that they aim at, or are appropriate to, a formal specification of autonomous agent behavior. Generally, RNS2 differs from these approaches in its combination of desirable features. In particular, none of these approaches is comparable to RNS2 w.r.t. expressiveness and automated design-time resolution of autonomy-induced cooperation conflicts. Differences in detail are pointed out in the following.

An approach called OperA that shows several interesting parallels to RNS2 is described in (Dignum, 2004). OperA is a very powerful formal framework for the description of agent societies in the context of open knowledge management. Like RNS2, OperA is strongly based on the concepts of roles, norms and sanctions and considers agents as black boxes (i.e., no demands on the architecture and the intentions of the individual agents are made). OperA allows to express cooperation requests, but only at the level of communication ontologies. Moreover, in OperA cooperation requests per se have no normative impact on the requested agents; in-

stead, norms are strictly attached at the level of roles. Unlike RNS2, OperA does not deal with norm changes, that is, the set of norms is static. Similar to RNS2, OperA covers certain agent-role and role-role conflicts. OperA allows to specify (as part of so-called interaction contracts) how conflicts should be resolved during runtime, although OperA is rather unspecific about conflict resolution. In particular, OperA does not deal with design-time and automated conflict resolution.

Another approach showing interesting parallels to RNS2 is described in (Lopez y Lopez et al., 2002). The focus of this work is on norm compliance and on the question what motivations an agent might have to comply with norms. Like RNS2, this approach is based on the view that agents as autonomous entities may decide to *not* act in accordance with norms. Moreover, similar to RNS2 this approach considers the issue of positive and negative sanctions. A 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, RNS2 does not make restrictive assumptions on "things occurring within agents", but concentrates on the role level.

Another approach showing interesting parallels to RNS2 is presented in (Lupu and Sloman, 1997). 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 RNS2, this approach employs the role concept and supports a specification of context sensitivity. Main differences are that this approach does assume that agents always do behave norm-conforming (thus sanctioning is not considered) and that the specification of dynamic norm (and sanction) changes is not supported.

Two related works from the area of electronic institutions are (Salceda, 2003) (here a formal framework called HARMONIA for modeling institutions is presented) and (Esteva, 2003) (here a language called ISLANDER for specifying institutions is described). Similar to RNS2, HARMONIA and ISLANDER employ the concepts of norms and roles. Moreover, similar to RNS2 HARMONIA (but not ISLANDER) takes care of the need for explicitly representing sanctions as a consequence of norm-violating activities. Unlike HARMONIA and ISLANDER, RNS2 does not deal with the issue of agent-agent communication. Unlike RNS2, HARMONIA and ISLANDER do not address the problem of dynamically changing norms.

A related approach rooted in deontic logic is described in (Pacheco and Carmo, 2002). This approach concentrates on collective agency and offers, similar to RNS2, a normative system perspective. An important difference is that RNS2, though employing norm concepts known from deontic logic, works on the basis of standard propositional or predicate logic, rather than a deontic calculus (axioms and inference rules). 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 RNS2 are (Barbuceanu et al., 1999; Dignum, 1999). Unlike RNS2, these approaches do neither support the specification of dynamic changes in norms and sanctions nor do they support the specification of referential activities. Moreover, these approaches are not role-based, that is, they do not offer roles as an abstraction level. Instead, norms and sanctions are directly attached to agents and assumptions are made on agent-internal (cognitive) processes.

6 Conclusion

Although it is commonly agreed that cooperation and autonomy are equally important to the concept of intelligent agents, the field has primarily dealt with cooperation as a design variable. The reason for this onesidedness is that cooperative behavior has been always understood as a functional property that can be realized constructively, whereas autonomous behavior has been usually assumed to be a non-functional property an agent possesses "automatically". (Interestingly, this functional/non-functional view of cooperation and autonomy is also predominant in the field of conventional distributed computer systems.) Based on practical experiences with agent-based applications, since some years this assumption gives away to the insight that it is highly problematic to expect the desired kind and level of autonomous behavior to emerge as a byproduct during system development and application. As a consequence of this insight, autonomous behavior is more and more understood as a constructive/functional rather than an emergent/non-functional agent property. We think this shift in the way autonomy is viewed, which is also reflected by a steadily growing number of respective publications (see, e.g., Hexmoor et al., 2003; Nickles et al., 2004), is a necessary condition for successfully meeting the challenge of appropriately intertwining cooperation and autonomy.

This article described a formalism for expressing the intertwining of agent cooperation and autonomy. What is needed in addition to such formalisms are *methodologies* that support and guide developers in the process of determining the right intertwining and the right levels of cooperation and autonomy. To provide such methodologies is not trivial and calls for extensive research efforts, because they must take into consideration a number of performance-critical criteria (and the dependencies among them) that are affected by the "more-cooperative-versus-more-autonomous design" decision. On the one hand, these are IT- and application-specific criteria such as communication bandwidth, failure modes, fault tolerance, security, administrative constraints and development costs. On the other hand, these are agent-specific criteria such as communication languages and protocols, agent architectures and organization structures, and learning and planning abilities. A promising starting point for developing such "intertwining methodologies" appear to be analysis and design methods available in the area of agent-oriented software engineering (e.g., Bergenti et al., 2004; Weiß, 2002).

References

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.

F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors. *Methodologies and software engineering for agent systems*. Kluwer Academic Press, Boston et al., 2004.

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

A.H. Bond. A computational model for organizations of cooperating intelligent agents. In *Conference on Office Information Systems (COIS'90)*, 1990.

C. Castelfranchi. Commitments: from individual intentions to groups and organizations. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 41–48, 1995.

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.

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

V. Dignum. *A model for organizational interaction: Based on agents, founded in logic*. PhD thesis, Institute of Information and Computing Sciences, Utrecht University, 2004.

T.T. Do, M. Kolp, and A. Pirotte. Social patterns for designing multi-agent systems. In *Proceedings of the 15th International Conference on Software Engineering and Knowledge Engineering (SEKE-2003)*, 2003.

A. Drogoul and A. Collinot. Applying an agent-oriented methodology to the design of artificial organizations: a case study in robotic soccer. *Autonomous Agents and Multi-Agent Systems*, 1(1):113–129, 1998.

M. Esteva. *Eletronic institutions: from specification to development*. PhD thesis, IIIA, Spain, 2003.

H. Hexmoor, C. Castelfranchi, and R. Falcone. *Agent autonomy*, volume 7 of *Multiagent Systems, Artificial Societies, and Simulated Organizations (MASA)*. Kluwer Academic Publishers, 2003.

R.W. Keidel, editor. *Seeing organizational patterns.* Berrett-Koehler Publishers, San Francisco, 1995.

M. Kolp, J. Castro, and J. Mylopoulos. Organizational patterns for early requirements analysis. In *Fifth IEEE International Symposium on Requirements Engineering (RE01)*, 2001.

F. Lopez y Lopez, M. Luck, and M. d'Inverno. Constraining autonomy through norms. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, 2002.

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.

J.-J. Meyer and R.J. Wieringa, editors. *Deontic logic in computer science. Normative system specification.* John Wiley and Sons, Chichester, 1993.

S. Moss, H. Gaylard, S. Wallis, and B. Edmonds. SDML: A multi-agent language for organizational modelling. CPM Report 97-19, Centre for Policy Modelling, Manchester Metropolitan University, United Kingdom, 1996.

H.J. Müller and R. Dieng, editors. *Computational conflicts. Conflict modeling for distributed intelligent systems.* Springer-Verlag, Berlin, 2000.

M. Nickles, M. Rovatsos, and G. Weiß. A schema for specifying computational autonomy. In *Proceedings of the Third International Workshop on Engineering Societies in the Agents' World (ESAW-02)*, Lecture Notes in Computer Science, Vol. 2577, pages 82–95. Springer-Verlag, 2002.

M. Nickles, M. Rovatsos, and G. Weiß, editors. *Agents and computational autonomy. Potential, risks, and solutions*, volume 2969 (Hot Topics) of *Lecture Notes in Computer Science*, Berlin u.a., 2004. Springer-Verlag.

O. Pacheco and J. Carmo. A role based model for the normative specification of organized collective agency and agents interaction. *Autonomous Agents and Multi-Agent Systems*, 2002. to appear.

J.V. Salceda. *The role of norms and electronic institutions in multi-agent systems applied to complex domains.* PhD thesis, Technical University of Catalonia, Spain, 2003.

C. Tessier, L. Chaudron, and H.-J. Müller, editors. *Conflicting agents. Conflict management in multiagent systems*, volume 1 of *Multiagent Systems, Artificial Societies, and Simulated Organizations (MASA).* Kluwer Academic Publishers, 2000.

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

G. Weiß, M. Rovatsos, M. Nickles, and C. Meinl. Capturing agent autonomy in roles and XML. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003)*, pages 105–112, 2003.

F. Zambonelli, N.R. Jennings, and M. Wooldridge. Organisational abstractions for the analysis and design of multi-agent systems. In P. Ciancarini and M.J. Wooldridge, editors, *Agent-oriented software engineering. Proceedings of the First International Workshop (AOSE-2000)*, Lecture Notes in Artificial Intelligence, Vol. 1957, pages 235–252. Springer-Verlag, 2001.