Operational Modelling of Agent Autonomy: Theoretical Aspects and a Formal Language

Gerhard Weiß[1], Felix Fischer[1,2], Matthias Nickles[1], and Michael Rovatsos[3]

[1] Department of Informatics, Technical University of Munich, 85748 Garching, Germany
{weissg, nickles}@in.tum.de
[2] Department of Informatics, University of Munich, 80538 Munich, Germany
fischerf@tcs.ifi.lmu.de
[3] School of Informatics, The University of Edinburgh, Edinburgh EH8 9LE, United Kingdom
mrovatso@inf.ed.ac.uk

Abstract. *Autonomy* has always been conceived as one of the defining attributes of intelligent agents. While the past years have seen considerable progress regarding theoretical aspects of autonomy, and while autonomy has been identified as an enabler for new computing paradigms such as grid computing, (web-)service-oriented computing or ubiquitous computing, autonomy as a *software property* is still miles away from implementation. Because of the *legal responsibility* of designers or users for the actions of autonomous software, the implementation of autonomy will require rigorous modelling and verification, so as to ensure maximum dependability. We take a first step in this direction by introducing a formal language ASL (Autonomy Specification Language) that allows for a precise specification of the activities to be carried out by a set of agents, the deontic constraints imposed on these activities, and the implications of activity execution on particular constraints (i.e., constraint dynamics). Agent autonomy is implicit in an ASL specification as the degrees of freedom left to the agents for the execution of activities.

1 Introduction

Since the inception of distributed artificial intelligence, *autonomy* has always been conceived as one of the defining attributes of intelligent agents. In the past years, particular interest has been paid to the theoretical aspects of autonomy and related concepts (like the control of and cooperation between agents), and considerable progress has been made in formally defining these [10, 5]. In addition to that, the increasing complexity of software in domains like e/m-commerce, telecommunications, logistics, knowledge management, and simulation of social and economic processes on the one hand and the identification of autonomy as an enabler for emerging information processing paradigms such as grid computing, (web-)service-oriented computing or ubiquitous computing on the other have given rise to a more general interest in autonomy as a *software property*. Nevertheless, software systems that tap the full potential of intelligent agents and have autonomy as a *real* property[1] rather than just a catchy label are still miles

[1] This means decision and action choice for working and interacting towards a design objective even under critical and unexpected circumstances and without substantial human support or intervention.

J.P. Müller and F. Zambonelli (Eds.): AOSE 2005, LNCS 3950, pp. 1–15, 2006.

away from implementation. The main reason for this is obvious: while (technically) each piece of software can be given the autonomy to act on its own, it will always be the designers or users who are *responsible* for its actions in a legal sense. Hence, the only way towards the implementation of autonomy is via a *systematic* process of *rigorous modelling and verification*, so as to ensure maximum dependability of systems that are given the permission to act autonomously. Without this dependability, it is unlikely that autonomously acting agents will be broadly used in industrial, commercial and scientific applications.

We respond to this challenge and take a first step by introducing a formal language ASL (Autonomy Specification Language) that allows for a precise specification of the activities to be carried out by a set of agents, the deontic constraints imposed on these activities, and the implications of activity execution on particular constraints (i.e., constraint dynamics). Agent autonomy is implicit in an ASL specification as the degrees of freedom left to the agents for the execution of activities, so that its type and degree can be precisely tailored to the task at hand. ASL further allows for the automatic detection and handling of norm conflicts, such that conflicts can either be resolved at design time or appropriate measures can be taken regarding their runtime settlement. What distinguishes ASL from existing role- and norm-based models of agent interaction is its operational character and its expressiveness and flexibility particularly w.r.t. agent autonomy.

The remainder of this paper is structured as follows. Section 2 introduces ASL and gives a formal definition of its syntax. Throughout this section, the expressiveness and flexibility of ASL is illustrated in the context of an agent-based electronic trading platform. Section 3 identifies different types of conflicts in an autonomy specification and proposes strategies for their identification and resolution. Section 4 then discusses the features of ASL, compares it to related work and points to some shortcomings and future improvements.

2 The Autonomy Specification Language ASL

The basic view underlying ASL is that agents are embedded in a social frame that regulates their behaviour. This social frame, henceforth called *role space*, is composed of a set of *roles* which are available to the agents and through which they can try to achieve individual and joint objectives. An agent may own several roles at the same time, and the same role may be owned by several agents. In the context of this paper, roles serve as a means for specifying desired behaviour and for achieving behavioural predictability, but *not* to make sure that agents never exhibit unexpected and undesirable behaviour (which would simply be impossible if autonomy is taken seriously). In particular, roles may not fully specify or constrain the behaviour of potential owners, but leave room for individuality (so that different agents may fill in the same role differently, put emphasis on different aspects, etc.).

Formally, a role in ASL consists of a set of *activities* to which *norms* and *sanctions* are attached. As the owner of a role, an agent is exposed to all the norms and sanctions attached to the role-specific activities. ASL distinguishes between three different types of norms (namely permissions, obligations, and interdictions) and two types of

sanctions (reward and punishment). While norms correspond to behavioural expectations held by agents against each other in their capacities as role owners, sanctions denote (potential) consequences of norm-conforming and norm-violating behaviour. Hence, through norms and sanctions, a system designer can explicitly specify the limits within which an agent is supposed to act autonomously, and how these limits are enforced.

2.1 Notational Preliminaries

The syntax of ASL will be given as a set of production rules in extended Backus-Naur form (more precisely, these rules resemble a context-free grammar G, and this grammar generates the language $L(G)$ of valid ASL specifications). For the sake of readability, *nonterminals* (to be replaced) and ASL-specific **keywords** and special symbols (which both are terminal symbols) are written in different fonts.

2.2 Basic Language Constructs

Role Spaces. The most general abstraction employed by ASL is that of a role space composed of several roles to be played by the individual agents in their attempt to achieve their goals. This is captured by the nonterminal *role-space-spec*[2] and the production rule

role-space-spec ::= **role space** *role-space-id* { *role-spec*$^+$ }

where *role-space-id* is an identifier[3] composed of letters "L" and digits "D" and beginning with a letter, i.e.,

role-space-id ::= L { L | D }*

role-space-id (i.e. any result of its replacement) is referred to as a *role space identifier*. The nonterminal *role-spec*, which allows for the specification of roles as sets of activities, can be replaced according to the rule

role-spec ::= **role** *role-id* { *activity-spec*$^+$ }

where *role-id* is a *role identifier* and *activity-spec* is given by the rule

activity-spec ::= *basic-activity-spec* | *activating-activity-spec* | *deactivating-activity-spec* | *request-activity-spec*

The four nonterminals on the right hand side of this rule, corresponding to the different kinds of activities in ASL, are treated in section 2.3.

Example 1. Consider an agent-based electronic supply chain management system, for which the system designers have identified five roles "European supplier", "US supplier", "European assembly manager","US assembly manager", and "member of the board of directors". In ASL, this role structure can be written as

[2] Hence, *role-space-spec* is the starting symbol of the grammar G that generates ASL.

[3] All the different kinds of identifiers used throughout this paper are assumed to be defined in this way, individual identifiers are further assumed to be unique.

role space eSUPPLY
{ **role** EUROsupplier { ... } **role** USsupplier { ... }
role EUROamg { ... } **role** USamg { ... } **role** MBdir { ... } }

where the "..." remain to be filled with the appropriate activity specifications.

Variables. In ASL, variables can be specified explicitly according to the production rule

variable-spec ::= *variable-id* **of type** *variable-type* [*variable-range*]

where *variable-id* is an identifier and *variable-type* is a data type, i.e.,

variable-type ::= { **nat** | **int** | **real** | **bool** | **char** | **string** | **identifier** }

All types but **identifier** are standard primitive types known from various high-level programming languages. The type **identifier**, which encompasses all legal identifiers and has no operations defined on it, serves to enable a designer to effectively refer to specific roles and activities (details on these follow below). Optionally, variable domains can be restricted explicitly by giving possible (ranges of) values after the type in square brackets (e.g. [1..100] or [EUROamg, USamg]).

Status Statements, Norms, and Sanctions. In ASL, each role is defined through a set of characteristic activities. Attached to each activity of each role is at least one *status statement* that specifies the norms and sanctions an agent playing the role is exposed to with respect to this particular activity. ASL distinguishes three types of norms – permission (indicated by the keyword **p**), obligation (**o**), and interdiction (**i**) to carry out the activity – and two types of sanctions – reward (**re**) and punishment (**pu**) – that apply in the case of norm conformance and norm deviation, respectively.

As we have already said at the beginning of section 2, it is unrealistic to assume that agents as autonomous entities do always act in accordance with available norms (especially in *open* environments characterised by a changing population of heterogeneous, self-interested agents). Instead, agents may ignore or violate norms, be it intentional or not. ASL takes care of this fact by enabling designers to explicitly specify the consequences of norm-conforming and norm-deviating behaviour in terms of positive and negative sanctions (i.e., reward and punishment). In other words, norms alone do not impose any limitations on possible agent behaviour (since this is impossible due to our definition of autonomy), they rather work indirectly via the agent's internal reasoning about the attached sanctions, making certain behaviours (which may be undesirable from the designer's point of view) undesirable for the agent. Hence, it is the responsibility of the system designer to devise a set of norms that prevent undesirable behaviour and the appropriate sanctions to enforce these norms. In addition to that, norms can be coupled to logical conditions that specify the circumstances under which they are valid and apply.

Alternatively, the three types of norms (in combination with the sanctions attached to them) can be viewed as different ways to specify the boundaries of agent autonomy: while obligations and interdictions state which activities are outside an agent's range of behavioural choice and control, permissions state which activities are within. Putting sanctions aside, an agent may, but need not 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 such an activity solely depends on his own decision about how to pursue his goals.[4]

Returning to the ASL syntax, a designer can distinguish between two different types of status statements (i.e. norm-sanction pairs) attached to an activity:

- *independent* status statements (keyword **ind**) an agent becomes subject to as a direct consequence of entering the role to which the activity belongs; and
- *dependent* status statements (keyword **dep**) an agent as owner of the respective role only becomes subject to if they are explicitly "activated" by another agent (through the execution of special *activating activities*, details on which are given in section 2.3).

Hence, dependent status statements allow for the specification of *adjustable autonomy* [9], and the status statements attached to activating activities resemble a kind of "meta-autonomy" (i.e. autonomy w.r.t. influencing others' autonomy), and so on. Formally, status statements are given by the following rule:

status-statement-spec ::= < { **ind** | **dep** *role-id* } > : *norm-spec* [+ *sanction-spec*]

The *norm specification* is defined as

norm-spec ::= **norm** < { **p** | **o** | **i** } > <*condition*>

where *condition* is a standard Boolean expression over the variables of the activity to which the status statement is attached (evaluating to **true** or **false**) and denotes when the norm is actually valid. The *sanction specification* is given by

sanction-spec ::= **sanc** < { **re** | **pu** } > <*sanction-ref*>

Details on *sanction-ref* will be given at the end of section 2.3, for now it shall suffice to view *sanction-ref* as a (unique) identifier referring to a particular sanction. The following examples shall illustrate the use of status statements.

Example 2. Consider a status statement <**ind**> : **norm** <**p**> < **true**> attached to an activity Deliver of the role EUROsupplier (a complete specification of this activity will be provided in example 3 in section 2.3). Accordingly, each agent acting as EUROsupplier is permitted (as indicated by **p**) to carry out this activity (i.e., to deliver material) under any circumstances (as *condition* is **true**) and without any sanction coupled to this permission. Being an independent status statement (**ind**), an agent becomes subject

[4] In fact, for truly autonomous agents (which only judge norms by the personal consequences of attached sanctions) the distinction between different types of norms does not increase the expressiveness of ASL, since assigning both a positive and a negative sanction to each activity would suffice to fully specify the range of behavioural choice. This is an interesting similarity to deontic logic, where each of the operators can be defined via the respective other, and we will return to this aspect in the following section in the context of requests.

to this permission automatically when entering the role EUROsupplier. Further assume that the Deliver activity contains ⟨**dep** EUROamg⟩ ; **norm** ⟨**o**⟩ ⟨material = "steel"⟩ + **sanc** ⟨**pu**⟩ ⟨ChargeFine(500)⟩ as a second status statement. As indicated by "**dep** EUROamg", this status statement can be activated by agents acting as European assembly manager (how this can be done is described in the following section). Through this activation, a European supplier (more precisely, an agent owning the role EUROsupplier) becomes obliged (**o**) to fulfil all requests for delivering steel (from now on, and no matter what quantity of steel is requested). Moreover, this status statement says that a violation of this obligation results in a punishment (**pu**) in the form of a $500 fine (as indicated by "ChargeFine(500)").

Assuming a Closed World. A well known assumption in AI (and the modelling realm in general) is that of a *closed world*, stating that everything that cannot be shown to be true is assumed to be false. ASL adopts this principle in that every activity not *explicitly* declared as being permitted, obligatory or interdicted (under certain conditions), is *implicitly* assumed to be interdicted (under these conditions).[5] In software engineering terms, this corresponds to the *least privileges* and *complete mediation* design principles for secure software. The former principle states that users and programs should be endowed with as few privileges as possible, and the latter states that only those activities – more specifically, those data accesses – being explicitly allowed should in fact be executable. Obviously, implicit interdiction also requires an implicit sanction to be effective, which we assume to be the "grounding" sanction described in the following section.

2.3 Activity Specifications

Around the status statements defined in the previous section, we will now introduce the ASL syntax for four different types of activities, namely *basic*, *activating*, *deactivating* and *request*. The nonterminal symbols corresponding to these different types are `basic-activity-spec`, `activating-activity-spec`, `deactivating-activity-spec`, and `request-activity-spec`, respectively.

Basic Activities. All activities that concern the handling of resources and events are referred to as basic activities. Examples for resources to be handled are time, money, or data, and examples for events are the access to a database, the delivery of goods, the execution of a negotiation protocol, or the response to an environmental chance. In ASL, basic activities are specified according to the production rule

```
basic-activity-spec ::= act activity-id(variable-id*)
                        { variable-spec* ;
                          status range status-statement-spec+ }
```

where `activity-id` is an identifier. The activity takes a (possibly empty) list of parameters and contains a specification of all these variables and any additional (e.g. global)

[5] It should be noted that while practically there is no difference between implicit and explicit interdictions, the latter can be used *deliberatively* – through the execution of activating activities – to "override" permissions and obligations.

ones referred to by the activity specification. At the core of the activity specification is a nonempty set of status statements, the activity's *status range*.

Example 3. Consider the following basic activity specification as part of the role US-supplier:

```
act Deliver (material, quantity)
{ material of type string["steel", "silver", "gold", "platinum"],
  quantity of type nat[1 .. 1000];
  status range
  <ind> : norm <o> <quantity ≥ 100> + sanc <pu> <ChargeFine(500)>
  <dep USamg> : norm <p> <quantity < 100>
  <dep MBdir> : norm <i> <quantity > 50 and material = "silver"> +
                sanc <pu> <WithdrawRole> }
```

According to the independent status statement of this activity, a US supplier must (**o**) fulfil any delivery request with a quantity of at least 100. If this obligation is violated, the responsible US supplier has to pay a fine (more precisely, the agent who violated this norm in his capacity as US supplier). What's implicit in this independent status statement is that delivery of quantities *below* 100 is forbidden, but due to the first dependent status statement a US assembly manager can permit a US supplier to obey such requests (for any kind of material given in the variable specification). The second dependent status statement says that a member of the board of directors (MBdir) can forbid (**i**) a US supplier to fulfil requests for delivering more than 50 units of silver. An agent is no longer allowed to act as US supplier if he violates this interdiction (indicated by "WithdrawRole").

Activating and Deactivating Activities. As we have already mentioned, ASL explicitly captures adjustable autonomy (i.e. autonomy that changes over time) and meta-autonomy (i.e. autonomy w.r.t. influencing others' autonomy) by means of so-called activating and deactivating activities, which serve to activate and deactivate dependent status statements and thus dynamically expose role owners to certain norms and sanctions. The ASL syntax of activating activities is given by the rule

```
activating-activity-spec ::= act activity-id
                             activate activity-id of role-id
                             { variable-spec*;
                               status-range-spec;
                               impact status-statement-spec+ }
```

The first *activity-id* is a unique identifier for the activating activity, while the second *activity-id* and the *role-id* identify the activity being affected. The status statements included in *impact-spec* are those statements of that activity that are activated (i.e. the same that occur in the corresponding dependent status statement). Deactivating activities (nonterminal *deactivating-activity-spec*) are specified analogously with **activate** replaced by **deactivate** (the meaning of this should be clear).

Obviously, a sound ASL specification should include one corresponding activating activity for each dependent status statement in order to ensure that each such statement can be activated (and also a deactivating activity if it should be possible to deactivate it

afterwards). Compared to that, independent status statements are inherently active and they concern agents immediately upon entering a role. Finally, it should be emphasised that activating and deactivating activities apply at the role rather than the individual agent level (i.e., a status statement can only be activated for *all* agents acting as owners of a particular role).

Example 4. Consider the basic activity Deliver of a US supplier as defined in example 3. According to the first dependent status statement of this activity, a US supplier can be permitted by a US assembly manager to fulfil delivery requests under certain circumstances. Consequently, within the role USamg there should be an activating activity corresponding to this "permissive" status statement. Assume that this activating activity is given by the following specification:

act PermitDeliver
activate Deliver **of** USsupplier
{ EcoSituation **of type string**["poor" , "medium" , "excellent"] ;
status range
≤**ind**≥ : **norm** ≤**p**≥ ≤**true**≥
≤**dep** MBdir≥ : **norm** ≤**o**≥ ≤EcoSituation = "poor"≥ + **sanc** ≤**re**≥ ≤EarnBonus(500)≥
impact
≤**dep** USamg≥ : **norm** ≤**p**≥ ≤quantity $<$ 100≥ }

As desired, the **impact** part includes the first status statement (i.e., "≤**dep** USamg≥ ...") of the Deliver activity of a US supplier, thus clearly identifying both the activity to be affected and the effect of executing the activating activity (i.e., US assembly managers are granted the permission to deliver less than 100 pieces of material). The respective deactivating activity (for example called ForbidDeliver) will only differ by the keyword **activate** replaced by **deactivate** and will have just the opposite effect (in this case revoking the above permission). A pair of corresponding activating and deactivating activities hence facilitates the exertion of full control over the adjustable autonomy inherent in a dependent status statement. The semantics of the status range is the same across the different activity types (basic, activating and deactivating). Hence, according to the independent status statement, a US assembly manager is permitted (**p**) to execute this activating activity (hence to permit US suppliers to fulfil deliver requests with an order volume lower than 100) without any restrictions (**true**). According to the dependent status statement, a US assembly manager can be obliged (**o**) by a member of the board of directors (MBdir) to carry out this activating activity, provided that the economic situation is rated as poor. By following this obligation, a US assembly manager earns a bonus.

Request Activities. ASL allows a designer to explicitly specify requests for carrying out activities through so-called *request activities*. Request activities may be viewed as requests for behaving cooperatively by executing the requested activity. This not only allows for modelling autonomy w.r.t. issuing requests, but also enables a precise definition of the notion of "not executing an action a" often found in deontic frameworks, namely as "not executing a (immediately) *when requested*". The ASL syntax of request activities is defined quite similar to that of (de)activating activities by the rule

```
request-activity-spec ::= act activity-id
                          request activity-id of role-id
                          { variable-spec*;
                            status-range-spec }
```

with nonterminals as defined above. Again, the first *activity-id* serves to identify the request activity, while the second together with the *role-id* refers to the activity being requested. Observe that the parameters are determined by the activity being requested and need not be specified again. Possible restrictions on the parameters can be expressed by means of the request activity's status range.

Example 5. Assume that the following request activity specification forms part of the role USamg:

```
act RequestDeliver
request Deliver of USsupplier
{ material of type string["steel", "silver", "gold", "platinum"],
  quantity of type nat[1 .. 1000];
  status range
  <ind> : norm <p> <quantity ≤ 200>
  <dep MBdir> : norm <i> <material = gold> + sanc <pu> <WithdrawRole> }
```

According to this, a US assembly manager (i.e., an agent in his capacity as a US assembly manager) is permitted under certain conditions (as given in the status range) to request US suppliers to deliver certain types of material (namely, steel, silver, gold and platinum). The independent status statement says that a US assembly manager is permitted to order up to 200 units of material. According to the dependent status statement, once activated through a member of the board of directors, a US assembly manager is interdicted to request the delivery of gold.

An important feature w.r.t. the expressiveness and flexibility of ASL is that activities of any type can be subject to both (de)activating and request activities. In particular, this means that ASL allows for the formulation of "crossed" and "self-referential" constructs such as requests for requests, requests for disallowing certain activities (i.e. requests for carrying out activating or deactivating activities) and so on.

2.4 Modelling Sanctions and Autonomy Dynamics

So far, we have not given a formal definition of the nonterminal *sanction-ref* introduced on page 5 and have rather referred to sanctions by some abstract identifiers. By means of request activities, we are now able to introduce a natural yet much more expressive model of sanctioning. This can be done by defining a basic activity for every action that is to be executed as the result of a sanction (like paying a fine, for example), which is obligatory for every role it is part of. However, the corresponding request activity (which is required to put this obligation into practise) may not normally be executed, but is triggered *automatically* upon norm violation.[6]

[6] More precisely, this resembles an executive authority that constantly monitors all active norms and is allowed to execute the corresponding request activity – and does so – in case of a norm violation.

For sanctions to be of any use in the presence of really autonomous agents, failure to execute a sanctioning activity (which has become obligatory by the "triggered" request) will again have to be sanctioned, until ultimately some *grounding sanction* is reached (e.g. role withdrawal, as used in some of the above examples).[7] To enable the use of sanctioning activities in a status statement, we finally define

sanction-ref ::= *activity-id*(*variable-id**)

Example 6. Consider the following definition of a basic activity PayFine as part of the role USsupplier. It takes the amount of the fine as a parameter and is grounded in role withdrawal.

act PayFine (amount)
{ amount **of type int** ;
status range
≤**ind**≥ : **norm** ≤**o**≥ ≤true≥ ± **sanc** ≤**pu**≥ ≤WithdrawRole≥ }

The corresponding request activity (invoked automatically if USsupplier violates certain norms) then forms part of the role specification for the executive authority:

act ChargeFine
request PayFine **of** USsupplier
{ **status range**
≤**ind**≥ : **norm** ≤**p**≥ ≤true≥ }

Besides sanctioning, activities that are triggered automatically upon norm conformance or violation can also be used for modelling a wide variety of autonomy dynamics like, for example, alternatives in norms, reciprocal norms, or contrary-to-duty obligations. For example, the obligation to do either X or Y can be modelled by means of deactivating activities that remove the obligation for either of the two as soon the other one is performed (i.e. as a reward). As an example for contrary-to-duties, consider a contract according to which a seller is obliged to deliver some goods, and a buyer is obliged to pay a certain price (not necessarily after the goods have been delivered). However, if the buyer fails to pay for the goods, the seller must no longer deliver them (in addition to the buyer being fined). This situation can be modelled by means of a deactivating activity which impacts the seller's obligation (to deliver) and is triggered as a punishment for violating the obligation to pay. What is particularly interesting about this model of a contract is that the buyer's refusal to pay for the goods explicitly excuses the seller from delivering. The formalisation of these two examples in ASL is left to the interested reader as an exercise.

3 Autonomy-Induced Conflicts

Since ASL does not impose any limitations whatsoever on the different status statements in an activity (e.g., regarding their number or kind), the corresponding norms

[7] By this, we implicitly assume that (at least) this grounding sanction can always be enforced. The existence of such a grounding sanction is crucial for retaining control over any system in which autonomy is involved.

may be inconsistent. To this end, we will now define three basic types of autonomy-induced conflicts in terms of such inconsistencies and show how these can be detected and resolved at design time.

It should be noted that in the context of this paper the term *conflict* is used to denote conflicts between norms (as these, and possible other conflicts caused directly by them, are the conflicts that can be treated on the level of an ASL specification). The (low-level, design-time) conflict resolution strategies presented here do not address exactly the same problems as the (high-level, runtime) ones usually investigated in the context of agents, like negotiation, mediation, arbitration, etc. (see, e.g. [8, 13]). They should hence be seen as a supplement (able to completely avoid certain high-level conflicts) rather than an alternative.

3.1 Types of Conflicts

In the following, let

$\mathcal{S}1$ = <*status-type1*> : **norm** <*norm-type1*> <*condition1*> ...
$\mathcal{S}2$ = <*status-type2*> : **norm** <*norm-type2*> <*condition2*> ...

be two status statements that are part of the status range of an activity $\mathcal{A}$. $\mathcal{S}1$ and $\mathcal{S}2$ are then said to constitute a *potential conflict* if and only if

(i) $\mathcal{S}1$ and $\mathcal{S}2$ have one of the following three norm constellations:
- *norm-type1* = **o** and *norm-type2* = **i** ("OI conflict")
- *norm-type1* = **p** and *norm-type2* = **o** ("PO conflict")
- *norm-type1* = **p** and *norm-type2* = **i** ("PI conflict") and

(ii) it can happen that *condition1* and *condition2* evaluate to **true** at the same time (i.e., both $\mathcal{S}1$ and $\mathcal{S}2$ are applicable for a particular request).

A potential conflict of type OI turns into an actual conflict of this type, if both $\mathcal{S}1$ and $\mathcal{S}2$ are activated and a request for executing $\mathcal{A}$ is available for which both $\mathcal{S}1$ and $\mathcal{S}2$ are applicable. As mentioned above, permissions imply decision choice on the part of an agent, so the situation is somewhat different for conflicts of types PO and PI. A potential conflict of type PO turns into an *actual* PO conflict if additionally the agent being requested to execute $\mathcal{A}$ *prefers* to *not* execute $\mathcal{A}$ (i.e. to not fulfil the request, which is in accordance with the permission $\mathcal{S}1$) while at the same time being obliged to ($\mathcal{S}2$). Similarly, a potential conflict of type PI turns into an actual conflict of this type if additionally the requested agent prefers to execute $\mathcal{A}$ (i.e. to fulfil the request in accordance with the permission $\mathcal{S}1$) while at the same time being interdicted to do so ($\mathcal{S}2$).

Example 7. First, consider the independent status statement and the second dependent status statement of the Deliver activity specified in example 3 as part of the role USsupplier. Since the conditions of both evaluate to **true** for a request of at least 100 units of silver, they constitute a potential OI conflict. This can also be understood as a conflict between the roles USsupplier (as the independent status statement becomes active automatically through entering this role) and MBdir. An example for a potential PO conflict is given by the two status statements of the activating activity PermitDeliver given in example 4, where both the condition of the independent status statement and that of the

dependent statement evaluate to **true** if EcoSituation = poor. This conflict can also be seen as a conflict between the roles USamg (which includes the activity) and MBdir (through which the dependent status statement can be activated). Finally, the two dependent status statements of the Deliver basic activity constitute a potential PI conflict, as both are activated through a request for delivering x units of silver where $51 < x < 100$. This may also be understood as a conflict between the roles USamg and MBdir, through which the two status statements can be activated.

3.2 Conflict Detection

As only the status statements of a single activity may lead to conflicts in the above sense, their detection at design time reduces to a pairwise comparison of status statements and can be fully automatised by means of the following, rather simplistic, algorithm:

```
for each role R ∈ role-space-spec {
  for each activity A ∈ R {
   for each S1 ∈ status range of A {
    for each S2 ∈ status range of A \ {S1} {
     if (norm-type1 and norm-type2 are of type OI, PO or PI) {
      test whether there is a variable assignment that satisfies
      both condition1 and condition2 } } } }
```

For conditions encoded in propositional logic (or first order logic with finite domains), the innermost tests are decidable, and at most $n \cdot m^2$ of the tests are required, where n is the total number of activities for all roles and m is an upper bound for the number of status statements included in the status range of a particular activity. However, a single test may take time exponential in the number of variables shared by *condition1* and *condition2*.

3.3 Conflict Resolution

Given that all potential conflicts in an ASL specification can be identified, we will now present three specific strategies for the resolution of such conflicts. All of them are based on specifying at design time which of two (or more) conflicting norms will actually be enforced.

- *Norm ordering*: define an order (a reflexive, antisymmetric, transitive relation) $\prec_N$ on the three norms **o**, **i** and **p**, determining which of two norms overrules the other in case of a conflict. This ordering can be *partial* (e.g., **i** $\prec_N$ **o** and **p** $\prec_N$ **o**) or *total* (e.g., **i** $\prec_N$ **o** $\prec_N$ **p**).
- *Role ordering*: define a (total or partial) order $\prec_R$ on roles, determining which of two roles involved in a conflict dominates the other. This strategy is often found in human organisations (where the decisions of one role may be overruled by a superior), and it makes sense because, as we have seen above, a conflict between two status statements can always be attributed to the roles to which the status statements belong or by which they have been activated.
- *Status statement ordering*: impose an order $\prec_S$ on conflicting status statements. Again, this order can be *total* (in this case meaning that all pairs of *conflicting* status statements are ordered) or *partial*.

These strategies differ significantly w.r.t. their granularity. For example, norm ordering is rather unspecific, but fair in the sense that it is uniform across all roles. On the other hand, status statement ordering allows for responding to conflicts in a direct and highly specific manner, but at the risk of resulting in a very heterogeneous collection of relationships between norms. For instance, consider four status statements $\mathcal{S}1$ to $\mathcal{S}4$ with $\texttt{norm-type1} = \texttt{norm-type4} \neq \texttt{norm-type2} = \texttt{norm-type3}$, where $\mathcal{S}1$ is in conflict with $\mathcal{S}2$ and $\mathcal{S}3$ is in conflict with $\mathcal{S}4$. Irrespective of the individual norm types (but also in a possibly counterintuitive way), these statements can be ordered according to $\mathcal{S}1 \prec_S \mathcal{S}2$ and $\mathcal{S}3 \prec_S \mathcal{S}4$. Role ordering lies somewhere in between the other two strategies, but has the additional appeal of being the most "natural" approach.

Most importantly, both total norm ordering and total status statement ordering are guaranteed to resolve *all* OI/PO/PI conflicts (while role ordering obviously doesn't help to resolve conflicts between one and the same role). The same effect can be achieved by appropriately combining different partial orderings. Such a combination is appealing as it allows for balancing the specificities of different conflict resolution strategies, but has to be done carefully because of potential "meta-conflicts" between the strategies. For example, norm ordering and status statement ordering may put certain status statements into a different order. Such meta-conflicts can be resolved at design time by imposing an order (i.e., a meta-strategy) on the strategies (or strategy types) themselves.

Example 8. Again consider the two dependent status statements included in the status range of the basic activity Deliver defined in example 3. As described above, these statements constitute a potential PI conflict, which can be resolved by imposing an order on permissions and interdictions (i.e., $\mathsf{p} \prec_N \mathsf{i}$ or $\mathsf{i} \prec_N \mathsf{p}$). Now assume that the first dependent status statement (i.e., "$\underline{\leq}$**dep** USamg$\underline{\geq}$: **norm** $\underline{\leq \mathsf{p} \geq}$ ...") should "override" the second one (i.e., "$\underline{\leq}$**dep** MBdir$\underline{\geq}$: **norm** $\underline{\leq \mathsf{i} \geq}$ ..."), while in all other cases the decisions of a member of the board of directors should overrule that of a US assembly manager. This can be realised by imposing the desired order on the two status statements (i.e., "$\underline{\leq}$**dep** USamg$\underline{\geq}$ ..." $\prec_S$ "$\underline{\leq}$**dep** MBdir$\underline{\geq}$ ...") and on the roles (i.e., MBdir $\prec_R$ USamg) and by combining these two orderings according to the meta-strategy $S \prec R$.

4 Discussion

After this extensive treatment of the ASL syntax, we will now summarise the essential features of ASL, compare it to related work and point to some shortcomings that call for further research.

Features of ASL. From the engineering point of view, ASL offers two main benefits. First, it is a highly expressive language that enables designers to specify agent autonomy at a very precise level. Consequences of both norm-conforming and norm-deviating behaviour can be captured by means of positive and negative sanctions. Instead of making any assumptions about norm conformance or deviation, this exerts control on agents via their internal reasoning without limiting their autonomy. In a way, ASL is *neutral* w.r.t. autonomy (i.e. neither biased for nor against it). The fact that no assumptions whatsoever (e.g. mentalistic or based on social commitments) are made about the type or internal structure of agents is also reflected in the fact that ASL focuses on the role rather than the individual agent level. Context sensitivity of activities and norms

(and thus adjustable autonomy) can be captured by means of activating and deactivating activities, which may either be executed at will by other agents or follow implicitly in case of conformance with or deviation from certain norms. Request activities can be used to explicitly model cooperation and coordination between agents. Finally, nested activity constructs of arbitrary complexity can be formalised in a natural way, such as requests for requests or requests for activating activities. A second key feature of ASL is that it allows for the detection and resolution of autonomy-induced conflicts already at design time. To this end, different types of conflicts and different strategies for their resolution have been identified. While this does not render high-level conflict resolution techniques usually investigated in the context of agents, like negotiation, mediation or arbitration (see, e.g. [8, 13]) unnecessary, it makes the most of what can already be done a design time. To have at least a partial alternative to the high-level strategies is important, because the former are not always applicable in real-world contexts (e.g., due to limited communication bandwidth, knowledge, or time available to identify potential compromises and put them into practice).

Related Work. There are several existing approaches for modelling the interaction of autonomous agents, mainly in the area of electronic institutions and organisations. [3] introduces an abstract, normative, role-based model for interactions between autonomous agents within an organisation. This model uses a deontic temporal logic to formalise contracts about agents' capabilities and obligations. [12] presents a framework for the normative specification of electronic organisations of autonomous agents at different levels of abstraction. [11] uses a special deontic and action logic, with includes "acting in a role" as first-order concept, to devise and reason about role-based models of groups of autonomous agents. While both ASL and the above approaches (as well as several others, e.g. [2, 4, 6, 14, 15]) use deontic concepts to specify (the boundaries of) autonomous behaviour, there are three main differences. Firstly, ASL has been built top-down for maximum expressiveness and flexibility, especially w.r.t. agent autonomy. Secondly, it lends itself very well to an operational or procedural interpretation, which is useful when an abstract specification is to be transformed into a concrete (i.e. implementable) agent system. Thirdly, ASL includes a notion of autonomy-induced conflict, and allows for handling such conflicts and hence reducing the inherent contingency of autonomous systems already at design time. There also exists a close relationship between ASL and policy specification languages, in particular the Ponder language [1]. Ponder is a declarative, strongly-typed, and object-oriented language for the specification of security policies and for policy-based management of computer networks and distributed systems [7]. It is fully implemented and supported by a number of tools.

Future Work. In this respect, part of our future research will be concerned with a more detailed investigation of the fundamental relationship between agent autonomy and security policies in general and the languages ASL and Ponder in particular. Unlike Ponder, ASL as defined in this paper does not include the usual (object-oriented) constructs for role modelling (inheritance, composition, etc.) and assignment to individual agents. While this does not limit the expressiveness of ASL, it would be rather cumbersome to have certain activities (like the "sanctioning" activity PayFine) that are part of a large number of roles.

On the conceptual side, we see two main shortcomings of ASL in its current form. First, it would be desirable to introduce *explicit* time and hence allow for the specification of deadlines as temporal constraints on norms (i.e. the time interval between a request, the execution of the corresponding activity and the initiation of a possible sanction) or other temporal aspects of autonomy (e.g. norms that are valid only at a certain time). Second, giving a formal (e.g. possible worlds) semantics to ASL will provide a proper theoretical grounding and ultimately pave the way for model checking the autonomy-related properties of a system. Our current research addresses these issues to further improve the expressiveness of ASL and support the engineering of autonomy as a property of dependable software systems.

References

1. N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *Proceedings of the 2nd International Workshop on Policies for Distributed Systems and Networks*, volume 1995 of *Lecture Notes in Computer Science*, Bristol, UK, 2001. Springer.
2. F. Dignum. Autonomous agents with norms. *Artificial Intelligence and Law*, 7:69–79, 1999.
3. V. Dignum. *A model for organizational interaction: based on agents, founded in logic*. PhD thesis, Utrecht University, The Netherlands, 2004.
4. M. Esteva. *Eletronic institutions: from specification to development*. PhD thesis, IIIA, Spain, 2003.
5. H. Hexmoor, C. Castelfranchi, and R. Falcone. *Agent autonomy*, volume 7 of *Multiagent Systems, Artificial Societies, and Simulated Organizations (MASA)*. Kluwer Academic Publishers, 2003.
6. 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.
7. E. Lupu and M. Sloman. Towards a role based framework for distributed systems management. *Journal of Network and Systems Management*, 5(1):5–30, 1997.
8. H.-J. Müller and R. Dieng, editors. *Computational conflicts. Conflict modeling for distributed intelligent systems*. Springer, Berlin, 2000.
9. D. Musliner and B. Pell. Agents with adjustable autonomy. Papers from the AAAI spring symposium. Technical Report SS-99-06, AAAI Press, Menlo Park, CA, 1999.
10. M. Nickles, M. Rovatsos, and G. Weiß, editors. *Agents and computational autonomy. Potential, risks, and solutions*, volume 2969 (Hot Topics) of *Lecture Notes in Artificial Intelligence*, Berlin, Germany, 2004. Springer.
11. 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 (JAAMAS)*, 6(2):125–184, 2003.
12. J. Salceda. *The role of norms and electronic institutions in multi-agent systems applied to complex domains*. PhD thesis, Technical University of Catalonia, Spain, 2003.
13. 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.
14. H. Verhagen. *Norm Autonomous Agents*. PhD thesis, Department of System and Computer Sciences, The Royal Institute of Technology and Stockholm University, 2000.
15. 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)*, pages 105–112, 2003.