Abstract

Although rule-based modeling has been successful in many areas, many rule languages have severe weaknesses, especially when trying to model large systems of rules, some of which describe both human and computer behavior. In these situations, the current rule languages easily become too restricted, both in expressiveness and the ability to deal with inconsistencies.

Deontic operators are proposed to address the problem of expressiveness in rule languages. They enable a distinction between facts and norms, as well as priorities of actions. The suggestion contains five deontic operators: obligation, recommendation, permission, objection and prohibition.

The proposal is a personal extension of the TEMORA project on rule-based information systems development, including deontic operators in the temporal rule language ERL.

1 Introduction to Rule-based modeling

Rule-based modeling is becoming increasingly popular. As stated by Twining and Miers[35], “One reason why the notion of ‘rule’ is such an important one not only in law, but in fields as varied as linguistics, sociology, anthropology, education, psychology and philosophy, is that there is hardly any aspect of human behavior that is not governed or at least guided by rules.” Also within the field of information systems development, rule-based approaches are receiving increasing attention. Several advantages have been experienced with a declarative, rule-based approach to information systems modeling:

- Problem-orientation: With an explicit specification of assumptions, rules and constraints, the analyst has freedom from technical considerations to reason about true application problems [10]. This facilitates a problem-oriented approach [4, 11].

- Maintenance: The declarative approach makes possible a one place representation of every rule and fact, whereas a more traditional operational specification has a poorer locality of change [30]. Rule maintenance is thus simpler when rules are explicitly represented.

- Knowledge enhancement: Rules are not always explicitly given. According to Stamper [33] “Every organization, in as far as it is organized, acts as though its members were confronting to a set of rules only a few of which may be explicit ¹. These are the essential features of the organization that the analyst must uncover.” Several researchers look upon IS specification as a process of rule reconstruction [8], i.e. specification is not only to represent rules that are already known, but also to uncover rules which are not yet part of a shared organizational reality.

On the other hand, several critical remarks have been made towards existing rule-based approaches. Some of the most notable critique regard limited expressiveness, flexibility, comprehensibility and poor performance.

In this paper we will concentrate on expressiveness. The most common rule-based languages have several rigid limitations as to what they can and cannot express:

- just true and false: every statement must be either true or false, there is nothing in between. This problem has been addressed, however, by the occurrence of multi-valued and fuzzy logics.

¹Our italics.
• just facts and constraints: generally, one can only express that some state is obligatory or prohibited in the information base [37]. Hence, it is impossible to distinguish between plain axioms or facts of the domain (e.g. that the age of a person is never below zero) and norms, which should not, but may well, be broken (e.g. that all employees should be above 18 years old).

• no organizational connection: Rules usually have the forms state implies state or state implies action, not stating who is responsible for performing the action or achieving the state. In human organizations, it is often the case that an action which is prohibited to one group of persons can be permitted for another, and similarly, actions which are obligatory tend to have certain actors responsible for performing them [20].

With the rule language developed in TEMPORA [24] as a starting point, we will address expressiveness with the conceptual extension of deontic operators in the rules.

The rest of the paper is structured as follows: Section 2 describe the conceptual basis and underlying architecture of TEMPORA. In section 3 we discuss the use of deontic operators in the TEMPORA rule language. An intuitive semantical interpretation of the deontic operators in the scope of the TEMPORA runtime-model in section 4. Section 5 gives an overview of related work, and section 6 concludes the paper, positioning our work in relation to other work being done on deontic logic in computer science and giving an overview of further work to be done within our approach.

2 Introduction to TEMPORA

We will here give a brief overview of the main concepts of TEMPORA. See [34] for further detail.

2.1 Conceptual Basis

TEMPORA has the following languages for conceptual modeling:

• ERT (Entity, Relationship, Time) [25, 34], a graphical language for the modeling of static aspects. Modeling constructs are: Entity classes, relationship classes, and value classes. The language also contains generalization and aggregation, and derived entities and relationships, as well as some temporal extensions particular for ERT. A grouping mechanism exist to enhance visual abstraction of ERT models.

• PID (Process Interaction Diagram) [9, 34], a graphical language for the modeling of dynamic aspects being an extension of data flow diagrams. Modeling concepts are: processes, stores, external agents, timers, control and data flows and ports.

• ERL (External Rule Language) [24, 34], a formal language for expressing the rules of an organization. Both static and dynamic aspects including the temporal dimension are possible to express. The ERL is based on first-order temporal logic, with the addition of syntax for querying the ERT model.

The general structure of a rule is as follows:

WHEN trigger IF condition THEN consequence

where trigger and condition are optional and consequence is either an action or a state.

An ERL-rule must be categorized as being a constraint, a derivation rule, or an action rule. Constraints express conditions on the ERT database which must not be violated. Derivation rules express how new information can be derived from information that already exists. Action rules express which actions to perform under what conditions.

Whereas constraints and derivation rules applies to views of the ERT-model, action rules are usually linked to atomic processes in the PID, giving the execution semantics for the processes as described in Krogstie et al. [19]. Processes is further combined into transactions.

2.2 Runtime Architecture

The information captured by using the above languages is translated into TEQUEL, an internal language running over a Rule Manager [34]. Roughly speaking, a TEQUEL specification is a set of rules on the form [19]:

formula about the future <= formula about the past

These rules are evaluated with respect to a particular state in a historical database, yielding a number of formulae about the future which must be made true, if not already true. The TEMPORA rule language thus becomes a kind of executable temporal logic, with a declarative past and an imperative future. Within the historical database there are two flows of time. The first is the flow of time along which the database
changes, and the second is the flow of time which is modeled within each version of the database. Each individual version of the database can be imagined as a sequence of intervals. These have a minimal time-span of the length of a tick.

3 Intra-rule Deontic Operators

Although the inclusion of triggers and temporal operators have given the rule language a higher expressiveness than many traditional rule languages, it still suffers from many of the weaknesses mentioned in section 1. In this section we will present two extensions that go beyond the TEMPORA rule language, namely

- introducing deontic operators in the rules, and
- connecting the rules to roles (agents) in the organization.

After a short introduction of deontic logic we will describe the extended rule language. A detailed discussion on the execution semantics of the operators can be found in section 4.

3.1 Deontic Logic

Deontic logic is a branch of philosophical logic concerning reasoning about norms. As such it is relevant for the foundation of rule systems such as law, ethics, and organizational policies.

Deontic logic has a long history, but the first suggestion of importance was the system of Von Wright in 1951 [36]. A traditional Kripke style possible world semantics using obligatory O as the basic necessity operator gives the following relations between the basic deontic operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Permission</th>
<th>Obligation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permitted</td>
<td>Pp</td>
<td>O¬p</td>
</tr>
<tr>
<td>Forbidden</td>
<td>¬Pp</td>
<td>Op</td>
</tr>
<tr>
<td>Obligatory</td>
<td>¬P ¬p</td>
<td>Op</td>
</tr>
</tbody>
</table>

Even if most deontic systems only deal with obligations, permissions and prohibitions, Jones and Pörn have introduced a scheme differentiating between the modal concepts of ought and must [13]. The construction of this system (DL) was guided by the hypothesis that the main reason for the inadequacy of standard deontic logic derives from the fact that its O-operator can be used only for describing deontically ideal versions of a given world, whereas sub-ideal versions go beyond the power of description.

A more comprehensive overview of the area can be found in [28].

3.2 Language outline

As a result of case-studies [17], we have earlier proposed a new rule-structure[17, 18] having the following form:

WHEN trigger IF condition, IT IS deontic FOR agent consequence

The rule WHEN τ IF φ IT IS D FOR α ψ can be written

\[ D_α (ψ / ¬ • τ ∧ τ ∧ φ) \]

where ψ, τ, and φ are statements in first order temporal logic. An action is written ψα whereas a state is written ψτ.

All parts except ψ are optional. The format of τ, φ and ψ are equal to ERL.

The major extensions are

- D, being one of the deontic operators obligatory (O) , recommended (R), permitted (P), objectionable (OBJ) , prohibited (PRO). O is default.
- α, being the one subject to the rule, i.e. the one being responsible for achieving or avoiding ψ. α will typically be a role in the organization, played by a person or an organizational unit — or by the computer. The default agent is the computerized information system.

The introduction of α makes it possible to connect the rules more properly to the parts of the organization that they apply to. Most of the following discussion though will be on deontic operators, and we will only discuss the agent extension in cases where this is directly relevant as a supplement to the deontic operators.

Traditionally, deontic logic has only offered the triplet O, P, PRO. R and OBJ are not expressible in terms of these operators, but in terms of each other having OBJ ψ ↔ R ¬ψ. How they relate to the other operators will be discussed in section 4.

ERL can only express what is prohibited or obligatory. Many rules concern permissions including authorization. The more vague nature of some higher level rules can be expressed with the help of recommended and objectionable. To assist in the sense-making process, we wish to be able to express all business rules applicable in the domain, not only the ones that are
to be put into the finished information system. The need for expressing vague rules will therefore be more prominent than if we were only expressing the rules to be automated.

3.3 Using the extensions

There are several uses of the deontic and agent extensions to ERL:

- **establishing priorities among requirements/rules**: Both during analysis and execution it is useful to establish priorities among various rules or requirements. In analysis because the resources of a project are seldom sufficient for satisfying all requirements. During execution to be sure to promote the most important actions in cases of limited processing capabilities. The deontic operators directly indicate such priorities.

- **separating responsibilities between various agents**: A good requirements specification should contain not only the responsibilities of the computerized information system, but also of the human actors around the system. The agent construct in the suggested rule language will tell whether a certain action is to be taken care of by the computer or by a human agent. When it is established what agent is responsible for something, this could also be utilized during execution, for instance by the system providing a reminding service to the agent at an appropriate time before the deadline for the action.

- **distinguish requirements from facts**: As pointed out by Jones [12], there is often a need to distinguish what is obligatory per se from what is actually done to detect violations of rules. These situations often lead to so called secondary obligations, where deontic operators may be appropriate. In TEMPORA secondary obligations can be addressed using other concepts, namely the temporal operators and timers in the process interaction diagrams and by linking the action rules to processes, as discussed in [18].

- **exception handling**: With the traditional, rigid rule languages, exceptions will usually result in contradictions. With our new operators, however, it is possible to express something which should generally be the case (or not the case) as recommended (or objectionable). Then, exceptional actions which transgress the recommendation can still be specified, without creating an inconsistency.

- **authorization schemes**: To model authorization rules, P combined with an instantiation of α can be used.

4 Semantical interpretation

In this section an outline of the run-time semantics of the proposed rule-language will be presented, in the context of the TEMPORA framework focusing on how the deontic operators is to be interpreted.

For action rules, we have the following interpretation with (¬ • τ ∧ τ ∧ φ) being implicitly true:

- O ψα ⇒ Do ψα
- R, ψα ⇒ Do ψα if time
- Pαψα ⇒ Do ψα if α is authorized
- OBJ ψα ⇒ Don’t do ψα if not forced to
- PRO ψα ⇒ Don’t do ψα

The rule with prohibition will be interpreted as a constraint, and will only make impact if we have another rule with ψα in the same tick.

The semantics of P may not be obvious. The meaning is that it requires the system to support a service. If no one wants to perform the action, the system does nothing. If Pαψα, and α wants to perform ψα, it is carried through. if ¬ Pαψα then nothing is done, but the security violation is logged. This shows that one has to specify social agents for P to be meaningful. Note that this is not in contrast with the traditional interpretation of P in section 3.1, but is an extension.

The most interesting issue here, however, is the semantics of R. Since we on the one hand want a small tick size, and on the other hand cannot know how long time the actions that we want to perform at a given tick will take, it is proposed to operate with two time scales, one real and one virtual [31]. When we are not able to perform all the actions in the time set for it, we will be living on “borrowed time”. When the load on the system decreases, we will then be able to catch up and the two time axes will coincide. Hence, in periods when there is too much to do for the computerized information system, the use of various deontic operators indicate what actions are the most important to perform. Action rules that are recommended are only executed if we are not living on borrowed time. In this situation we have two choices:

1. We can suspend the recommended action, executing it later when the load decreases. For instance, we could have a situation where we receive a telephone call from a customer ordering

2We try to perform all transactions induced by the events happening in a tick in the next tick.
some goods. We might then send him the goods at once, or postpone this until we have time, but we always want to do it.

2. In other cases we like to either perform the action at a specific time in connection with the execution of other actions, or not at all. For instance we could imagine a shopping clerk which is recommended always to write a receipt to a customer who purchases goods for more than $100, but who will not do this if there are other customers in queue and the customer does not explicitly request a receipt.

In our framework, we use the second semantics. The first situation can be expressed intuitively by using the temporal sometimes operator (◊) which is already supported by ERL. Hence Rψa ≠ ◊ψa. Recommended actions are performed if we have time, but otherwise forgotten.

For integrity constraints we have the following interpretation:

- O ψs ⇒ ψs must always hold
- R ψs ⇒ ψs should hold
- P ψs ⇒ Not applicable
- OBJ ψs ⇒ ψs should not hold
- PRO ψs ⇒ ψs must never hold

As all constraints will apply to the system, P is not used in constraints.

An example of the use of recommended states is a reorder strategy where we have that it is recommended (not obligatory) that the stock-level of all articles are above a given threshold. We do not like to go below this threshold for the stock of each article in any case though, but we neither want to reorder the goods automatically when going under the threshold since this may be uneconomical. Having the constraint as a recommendation will only notify the system manager about the status, and the manager can then choose if s/he wants to activate a process that reorders the article manually.

Derivation rules are currently only obligations.

As illustrated, the use of the deontic operators are restricted to act as meta-operators used under the construction of the set of rules that applies at each tick, and will not be used in the condition of a rule. On the other hand, we see from the rule-format that we might get temporal operators within the scope of deontic operators in the cases where the operators apply to the action of an action-rule. How to treat the different cases is outlined below:

If there are no temporal operators on the action, this is performed immediately and the deontic operators applies as outlined above. When a temporal operator\(^3\) specifies that the action is to be performed some time in the future, either at a specific time, within a time-period or at an unspecific time, we will use the normal mechanisms for choosing when the action should be performed. If O(TEMP ψa), we do as usual when dealing with TEMP. If R(TEMP ψa), the situation will be different for the cases of TEMP:

- If TEMP specifies a point in time, we will check if we are living on borrowed time at the end of the previous tick, and only perform ψa if not.
- If ψa is to be performed within an interval, we will perform it as soon as there is enough time within the interval. If we have not had time to do it within the interval, we will discard the action altogether.
- R◊ψa →◊ψa since it is not essential that ψa is performed immediately as it usually is when we deal with recommendations.

If Pψa, a check on the authorization of α will be performed when it is decided to perform ψa.

An execution cycle incorporating the new deontic operators is presented below. It is an extension of the execution-cycle given by McBrien in [23]. Note that this is somewhat different from the current implementation of the TEMPORA prototype [34], which is not handling non-determinism. The principles for how to include this exist. A complete execution-cycle including the further steps necessary when also using deontic relations between rules can be found in another article.

1. A primary set of rules is constructed.

2. The head of every rule in the rule set is evaluated, and for all successful evaluations, the rule and its potentially accompanying transaction is put into the rule-set for this tick. In addition, previously rolled-back and suspended rules that now apply will be put into this set.

3. The resulting rules are checked for the following:

   - A recommendation will be discarded if we are already living on borrowed time.
   - A rule that is prefixed by ◊ or another temporal expression with similar indeterminism will be suspended if we are living on borrowed time.

\(^3\)Denoted TEMP below
• A rule that is recommended and that is supposed to be performed within a given time interval, is suspended if the time interval is not reached or if the time interval is not finished and we are living on borrowed time. The rule is discarded if the time-interval has ended and we are still living on borrowed time.
• If a rule is initiated by an agent not authorized to perform the action, the rule will be discarded.

The tails of the resulting rules will then be collected into the action set of the tick.

4. The action set is checked to see if it is not self-contradictory, contradictions being resolved if possible.

5. The transactions are executed, causing external events to occur, changing databases or scheduling new actions for future times.

6. We move to the next tick on the event time axis, making available to the TQL any external input which arrived during the evaluation of the previous tick and create a new rule set.

4.1 Practicality

Since we support temporal operators and deontic operators some might be doubtful as to the suitability of the approach for real-life engineering activities. However, some points in our defense are:

• The temporal language (from which our only extensions are the deontic operators and the agent construct) is clearly executable, as has been demonstrated in the TEMPORA project.
• The agent construct does not influence the actual execution since it only affects the rules being chosen for the rule set, removing unauthorized actions.
• The use of the deontic operators is heavily restricted. Deontic operators do not occur in the head of the rule. P is not used in constraints and only O is used for derivation rules.

Hence, the only thing which is needed in addition to a standard Horn Clause inference engine is a temporal database with a querying facility (which is already provided by the TEMPORA project, as well as by other research projects). The changes needed is in the selection of the rule set at each tick which can be performed on a level above the actual inference engine.

5 Related Work

Space limitation has made it objectionable for us to present a thorough overview of the field. Some of the important works in the area is briefly mentioned below.

The interest for applying deontic logic in computer science started in the area of legal applications. An early example of this is the LEGOL-project [32] led by Stamper for developing conceptual modeling methods for information systems development. Another early example of the use of deontic logic for the computer representation of legal reasoning is research that followed the TAXMAN project [26]. The language described here is part of LDD, a more expressive language containing in addition to traditional deontic constructs, constructs for specifying actions, sorts and subsorts, and events and time.

Allan and Saxon [1] uses the system of concepts defined by Hohfeld to define a formal language in which to analyze legal texts precisely. These concepts include various nuances of the standard deontic concepts of permission, obligation and prohibition as well as others concepts like right, duty, no-right, privilege, power, liability, disability and immunity. An approach based on the work of Hohfeld, has been developed by Kanger and Lindahl [21]. Using deontic logic and the logic of action they construct a formal theory of normative positions which provides a means of exhaustively mapping out the space of all the logically possible normative positions concerning, for instance, any given pair of agents and any given act-type.

Jones and Sergot discussed the role of deontic logic in general to the representation of law in [14].

Another application area is security and authorization. Minsky and Lockman [29] propose to include the deontic concepts of obligations and permissions to authorization systems, stressing the need for secondary obligations. Glasgow, MacEwen and Panangaden [7] use deontic logic to analyze security policies for databases. They combine the deontic concepts with epistemic logic, which is also done by Bieber and Cuppens [3].

Integrity constraints for databases is another area where deontic operators are being taken into use. Wieringa et al. [37] differentiate between necessary constraints, which cannot be violated in the natural world, and deontic constraints, which may be violated. They use the basic operators of permission, prohibition and obligation. Khosla and Maibaum[16] use a similar approach for the specification of fault-tolerant systems, using an extension of modal action logic.

Lee has used a framework including waivers in ad-
dition to the three standard operators of deontic logic for analyzing administrative organizations as deontic system [20]. He combines the deontic operators with a Petri-net specification, the logic of change developed by Von Wright and the logic of absolute time due to Rescher and Urquhart. Deontic operators are added and a reduction of standard deontic logic to alethic logic is utilized.

Dignum et al. [5] base their textual conceptual modeling language on the linguistic theory of functional grammar and suggest a language combining traditional modal logic (expressing necessity), dynamic logic, temporal logic and deontic logic with first-order predicate logic.

Finally, ERAE [6] uses obligations to reason about the formal specifications. The description of composite systems are supported by a formal real-time temporal language augmented with deontic capabilities and agents.

For a further overview of existing work in the field, see [15, 27].

6 Conclusion and Further Work

We will in this section briefly sum up the contribution of our work compared with related work using deontic operators in computer science, before giving a short overview of further work to be done.

6.1 Contribution

Comparing our framework with existing approaches using deontic operators, we notice the following:

To our knowledge, none of the existing approaches deal with intra-rule recommendations and objections explicitly, even if these concepts have been discussed in deontic logic. Note that the difference between obligation and recommended is not the same as the one of necessary and deontic constraint, our obligations are still regarded to be deontic.

Some approaches that use other operators than obligations, permissions and prohibition do exist. For instance, Lee introduces waivers, which is defined as $W\psi \leftrightarrow \neg O\psi$. More interesting is the work based on Hohfield, which is orthogonal to our extension.

In our approach, we mix several modal operators, i.e. both temporal and deontic operators. This is also done in many other existing approaches, such as LLD and the work of Lee and Dignum et al. Other examples is the mixing of epistemic and deontic logic mentioned above, and the linguistically motivated language of Dignum et al.

In addition to linking rules to each other, we are linking the rules to other, visual conceptual models. This is also to a certain degree done by Lee, linking the rules to Petri-nets.

Existing approaches for DB-integrity constraints have only permitted and forbidden states. Also we might reduce our system to such using weakening rules, for instance to perform consistency-checking during analysis [17].

Using permissions, agents, and the other modeling languages, a support for authorization comparative to the ones reported above is achieved.

6.2 Further work

There are several tasks to be performed to be able to harvest the benefits of our approach fully:

- Incorporate the semantics of the deontic operators and agents in the TEMPORA run-time system.
- Look more closely upon pruning-strategies and efficient implementation of the execution strategy.
- Investigate how to deal with situations where a constraint is in fact violated.

References


