Chapter 4

Programming Regulative Norms

In the previous chapter we introduced the notion of organizational artifact as a useful tool that resides in the environment of the agents. In the class of systems that can be characterized by a degree of openness in which agents that are not a priori known dynamically start interacting with an organizational artifact (cf. (Davidsson, 2001)), also little can be assumed about the behavior these agents will exhibit. When unknown agents start their interactions dynamically and little can be assumed about the features of these interactions, there is a strong need for mechanisms to regulate their behavior in order to guarantee the global objectives of the artifact (and the system as a whole) to be achieved or maintained. Agents may, for example, (deliberately or accidentally) exhibit behavior that obstructs the system’s global goals. The use of an explicit representation of normative concepts such as obligations, prohibitions and sanctions has been widely promoted as a suitable tool for such regulation. Indeed, much literature can be found on the formal specification and verification of norms (Meyer and Wieringa, 1993), (Prakken and Sergot, 1996), (Dignum et al., 2004), (Sergot and Craven, 2006), (Boella et al., 2008) and also our own work (Dastani et al., 2008) are just but a few examples), and literature on the practical issues associated to operationalizing norms for their use in computational (multi-agent) systems e.g. (Esteva et al., 2004; Garcia-Camino et al., 2005; Silva, 2008; Minsky and Ungureanu, 2000; Dastani et al., 2009) (more examples can be found in chapter 2).

Despite these developments we observe a gap between research on the construction of normative frameworks and multi-agent programming languages. We conjecture that the following two issues underly the root cause of this gap. First, all of the above mentioned implementations (except for (Dastani et al., 2009;
Silva, 2008)) are primarily targeted at “procedural” norms specifying which actions agents ought or ought not perform and disregard the issue of expressing “declarative” norms related to a description of a desired state of affairs that should be brought about in an environment the agents interact with. Norms in existing frameworks thus typically take on the form of “ought-to-do” statements pertaining to actions rather than “ought-to-be” statements pertaining to the declarative description of a state. We argue that expressing declarative norms is also important, because:

1. If we can relate to actions only, writing norms to ensure a certain state is achieved might become a rather tedious task, especially when establishing it involves a multitude of actions and the endeavor of multiple agents. It is, for example, difficult to express a norm for a conference management system that at least two reviews should be received for each paper;

2. By only stating the actions the agents should perform to reach a desired situation we risk to limit their autonomy, because we leave them no choice in deciding how to reach it. For example, by telling an agent to take the train to a conference we leave it with less options than by just telling it to be at the conference;

3. Related to the previous point, expressing declarative norms accords more with the concept of declarative goal as often used for modeling agents, viz. a description of a desirable situation. This conformity facilitates the internalization of norms by agents for reasoning with them (see for example (Meneguzzi and Luck, 2009)). In the above example, the agent probably has multiple plans to achieve the goal of being at the conference, such that this obligation can be more easily internalized and acted upon.

Second, current work on normative frameworks for regulating the behavior of agents is not endowed with an operational semantics (Plotkin, 1981) that allows for a direct implementation of an interpreter. An operational semantics allows us to evaluate a framework by formally studying the key properties it exhibits and it relates it to some of the results established by the field of deontic logic. Many agent-oriented programming languages are already investigated by means of an operational semantics and also doing this for the normative concepts contributes to the longer-term goal of studying the properties of a multi-agent system regulated by norms.

In this chapter we aim to narrow this gap by extending the basic idea of an organizational artifact to include normative concepts for regulating the agents’ interactions with the aim of preventing the artifact from ending up in sub-ideal states. More concretely, the main contributions of this chapter are:

- We describe the overall architecture of a multi-agent system regulated by normative components in section 4.1. In particular, we explain our norm
enforcement mechanism in the context of an organizational artifact as presented in previous chapter and motivate our choice of the involved normative constructs.

- We introduce the syntax and intuitive semantics by which norms can be programmed in section 4.2. Our norms take on the form of conditional obligations and prohibitions with a deadline. Their representation and semantics is inspired by norms as presented by Boella et al. (2008) who only consider obligations. Contrasting existing work on computational norms (except for (Dastani et al., 2009; Silva, 2008)) that is primarily targeted at “procedural” norms specifying which actions agents ought or ought not perform, we consider “declarative” norms related to a description of a desired state of affairs that should be brought about in some environment the agents interact with.

- To motivate agents to abide by the norms we introduce sanctioning rules which are taken from our earlier work on counts-as rules (Dastani et al., 2008, 2009; Tinnemeier et al., 2009a) (also in section 4.2). The solution we develop allows both for regimentation and enforcement of the norms (cf. (Aldewereld, 2007; Castelfranchi, 2000; Grossi, 2007)). We demonstrate how these norms and sanctioning rules can be used to regulate the agents’ behavior and to prevent an organizational artifact from ending up in unwanted situations by means of our conference management system example (section 4.2).

- We endow the syntactical constructs with a structured operational semantics (Plotkin, 1981) in section 4.3. This enables us to study the normative concepts in a rigorous manner and an operational semantics is already close to the implementation of an interpreter without committing to a particular programming language. Moreover, it reduces the gap between research on agent programming languages (that are often endowed with an operational semantics) and research on computational, normative frameworks (that are rarely studied by an operational semantics).

- We demonstrate that different strategies exist for enforcing the norms in section 4.4. This has not been done before in respect of a computational normative framework (except for (Aştefânoaei et al., 2009a)). Moreover, in this section we show some of the key properties our norms exhibit, relating them to some of the properties that have been long studied by the field of deontic logic and normative multi-agent systems. This relation is generally overlooked by existing work on computational frameworks for norm enforcement.
4.1 Organizational Artifacts with Norms

As previously explained in chapter 3, we conceive a multi-agent system as consisting of a collection of heterogeneous agents and a collection of artifacts. The agents exploit the functionality provided by organizational artifacts to achieve their goals. An organizational artifact implements the non-autonomous functionalities that are better implemented by non-agent concepts. They encapsulate a domain specific state and function, which is modeled by a set of brute facts. The agents perform actions that change the brute state to interact with the artifact and exploit its functionality. An overview of the internals of an organizational artifact is shown in figure 4.1. In this chapter we will focus on the normative dimension of the organizational artifact.

![Diagram of organizational artifact internals](image)

Figure 4.1: A (simplified) conceptual representation of the internals of an organizational artifact. Optional concepts (e.g. norms, positions) are outlined with a dashed border. Solid arrows denote the reading and modification of the concepts as explained in more detail below. Dotted arrows denote actions and messages between agent and artifact. A brief description of all the concepts can be found in chapter 3.

When unknown agents of which little can be assumed about their behavior interact with the artifact, it is essential to coordinate their behavior and to guide them in interacting with it in a meaningful way. What is considered to be desired behavior is described by the norm schemes, a set of conditional obligations and prohibitions. The norm schemes define under which conditions obligations and prohibitions should be created. For example, if a reviewer is assigned a paper to
review, then an obligation to have uploaded a review for that paper is created.
The condition of a norm scheme relates to the brute and institutional state of
the artifact and whenever its condition is satisfied the artifact instantiates the
obligation or prohibition belonging to it, hence the name *norm instance*. The
process of triggering is summarized in figure 4.2.

The norm instances that are created out of the norm schemes are thus a set
of active (unconditional) obligations and prohibitions. They specify which brute
and institutional states should (not) be achieved. Seeing to it that the norms
and their instances are about the brute and institutional state an artifact encap-
sulates and maintains, we consider it the artifact’s responsibility to detect when
obligations and prohibitions should be created and detect violations and fulfill-
ments. We refer to this process by the name *monitoring* as shown in figure 4.3.
Assigning this responsibility to third-party entities or even to agents, would break
the principle of data hiding. Moreover, delegating this responsibility to another
entity with its own thread of control introduces nasty synchronization issues. In
particular, putting the burden of monitoring on designated ‘enforcement’ agents,
requires advanced reasoning capabilities of those agents. In chapter 2 we showed
that such capabilities are beyond the reasoning capabilities agents typically have,
and additional mechanisms would be needed to verify if the monitoring agents
carry out their task satisfactorily. To conclude, assigning the responsibility of
monitoring to external entities does not buy us anything, it would be merely
shifting the problem. We do note, however, that this is differently from how it
is done in the real world where, for example, policemen are used to detect traffic
violations. The point we make here is that if the system itself can detect viola-
tions, monitoring should be done by the system. Indeed, the same trend can be
observed in real life where the tasks of policemen are often replaced by automatic
devices, e.g. speeding cameras.

Usually, a norm instance is accompanied by a deadline (cf. (Dignum et al.,
2004; Boella et al., 2008)). For example, an obligation to have reviewed a paper
should be fulfilled before the notification phase starts. Intuitively, an obliga-
tion specifies which brute state should be established before a particular moment,
whereas a prohibition specifies which brute state should be avoided until the dead-
line. Resembling the specification of deadlines as propositional formulae instead
of time (Boella et al., 2008), in our approach, deadlines are expressed as a sit-
uation that can be entailed by the brute state. This gives us more freedom in
specifying different notions of deadlines and allows us, for example, to relate to
a situation that is established by the actions of other agents without the need to
know if and when it will be established.

A special type of norm that has also been discussed in chapter 2, is the one that
specifies which obligations or prohibitions should be instantiated in the sub-ideal
situation in which some other norm instance is violated. Contrary-to-duty norms
(Prakken and Sergot, 1996) are a special case of this. For example, a reviewer
that did not abide by its obligation to have reviewed a paper obliges the program
chair to remind him/her of this task. Recall that information about which norm

91
instances are violated is stored by the institutional facts (as taken from (Searle, 1995)). To also allow for the specification of these types of norms we let the condition of norm schemes relate to institutional facts. We call a norm that is triggered by some condition of the brute state a primary norm, and a norm that rises due to the violation of another norm a secondary one. This resembles the notion of interlocking of norms as proposed in (Lopez y Lopez et al., 2006).

Usually, a norm instance is directed at someone (Lopez y Lopez et al., 2006), but because the artifact’s interactants are not known beforehand, in expressing the norm schemes we rather refer to the roles that are played by agents instead of to the agents directly. For the present work we adopt the same simple representation of roles and their enactment as we did in the previous chapter. A role is a label \( r \) that identifies it by a unique name. We model the institutional fact that an agent identified by \( a \) has enacted role \( r \) by a proposition \( rea(a,r) \) as stored by the institutional facts. In subsequent chapters we will change this representation of roles to include a richer account.

To motivate agents to abide by the norms, there are often sanctions associated to the violation or fulfillment of a norm (cf. (Vázquez-Salceda et al., 2008; Lopez y Lopez et al., 2006)). These sanctions include punishments for an agent who infringes the norms and rewards for an agent who obeys the norms. For example, an author that violates a prohibition to upload a paper of more than 15 pages is punished by an instant rejection of the paper by removing it from the database. A question that rises is “who should be responsible for applying these sanctions?”

We already argued that the organizational artifact is responsible for activating obligations and prohibitions based on their underlying norms, and detecting violations and fulfillments of these active obligations and prohibitions. Based on a similar reasoning we could conclude that also sanctioning should be the artifact’s
Figure 4.3: Monitoring compliance of an obligation amounts to checking if the desired situation (of the brute and institutional state) can be entailed before the deadline. Violated and fulfilled obligations are removed from the set of norm instances. Monitoring obedience of a prohibition boils down to checking if the undesired situation can be entailed before the deadline. Prohibitions are removed when their deadline can be entailed. Information about violation and fulfillment is added to the institutional facts.

task. Additionally, sanctioning might require to modify the brute facts as our example of removing a paper suggests. However, sanctioning does not require advanced reasoning capabilities of the agents; they only need to know that someone has complied to or has infringed the rules without needing detailed knowledge about how the norms are expressed. This pleads for a mechanism that at the very least allows the artifact to notify agents about (non)compliance and leaving the decision and act of sanctioning to them, but at the very same time allows for the artifact to apply sanctions automatically.

Note that not every norm has a sanction associated to it. One might, for example, decide not to punish the infringement of a norm, but only reward compliance (or vice versa). Based on this observation and the fact that sanctioning is a different concern than norm activation and verifying compliance, we argue that the specification of sanctions and the process of applying them should be sensibly separated out as a different component. In our approach, sanctions are specified by so-called sanctioning rules taken from (Dastani et al., 2008, 2009; Tinnemeier et al., 2009a). They take on the form of a kind of reversed counts-as rules that specify which brute facts should be accommodated to the brute state of the artifact as a consequence of a violation or fulfillment of some obligation or prohibition. To enable the artifact to inform agents and other artifacts about (non)compliance we allow these reverse counts-as rules to have the same side effects as the performance of actions, namely messages to be sent or actions to be performed upon other artifacts.
4.2 Programming The Normative Dimension

In this section we describe the syntax of the programming constructs by which the normative component of an organizational artifact can be defined. We use the conference management system example to show how norms can be programmed and to explain their intuitive semantics. Recall that to program a basic normative artifact is to specify the roles agents can play, a set of facts specifying the initial brute state and a set of effects specifying how the brute state evolves under the performance of actions. The grammar of these constructs can be found in table 3.1 of chapter 3. All these constructs were explained in detail in that chapter and will not be repeated here. We do, however, repeat some of the elementary syntactical constructs in table 4.1 that were originally listed in table 3.1 and extend them, because they are needed for defining the syntax by which norms can be programmed. To extend our notion of organizational artifact to include norms, we extend the previously defined clause ⟨artifact⟩ as defined in figure 3.6 of chapter 3 in the following manner:

\[
⟨artifact⟩ = "Name:" ⟨id⟩ [ ⟨roles⟩ ] [ ⟨enacts⟩ ] [ ⟨facts⟩ ] [ ⟨effects⟩ ] [ ⟨norms⟩ ] [ ⟨sanctions⟩ ];
\]

The grammar of norms and sanctions is shown in figure 4.5. In what follows, we explain the intuitive semantics of the syntactical constructs by our conference management system example.

Norm schemes and Norm Instances

The conference management system has some behavioral expectations about its participants. We expect, for example, reviewers to review their assigned papers.
⟨label⟩ a first-order atom with constants and variables used to uniquely identify norms
⟨b−atom⟩ a first-order atom denoting a brute fact. The special facts starting with predicate symbol viol, obey and rea (their meaning to be explained later on) are excluded from the set of brute facts.
⟨i−atom⟩ a first-order atom of the form viol(φi) or obey(φi) with φi a ⟨label⟩ identifying a norm. The first denotes a violation of norm labeled φi whereas the latter denotes the obedience to the norm.
⟨r−atom⟩ a first-order atom of the form rea(i,r) in which r denotes a role and i the agent playing it; “rea” is short for role enacting agent.
⟨send⟩ a first-order atom of the form Send(r,p,c) in which r denotes the message’s receiver, p its performative and c its content.
⟨do⟩ a first-order atom of the form Do(id,α) in which id denotes the artifact upon which action α is to be performed.

Table 4.1: Elementary syntactical constructs.

in time. Such behavioral expectations are are expressed by the norm schemes, that is, conditional obligations and prohibitions. In our framework, we do not consider permissions; if something is not explicitly forbidden, we assume it to be permitted.\(^1\) A conditional obligation is expressed as a labeled tuple of the form φi : ⟨φc,O(φx),φd⟩ with the intuitive reading that “if condition φc holds then there is an obligation to establish the brute state denoted by φx before deadline φd”. A conditional prohibition is expressed as a labeled tuple φi : ⟨φc,F(φx),φd⟩ that can be intuitively read as “if condition φc holds then it is forbidden to establish the brute state denoted by φx before deadline φd.” Note that each φ

\(^1\)Here we assume that permission is the dual of prohibition. It should be noted that there are also interpretations in which this is no longer the case, see for example (Hansen et al., 2007) for a discussion.

95
is a conjunction of brute and institutional literals. By allowing the precondition to contain rea atoms, we can associate norms to roles that are played by agents. The parameterized labels $\phi_l$ the norms are equipped with, is used to uniquely identify them and to keep track of which of them are violated. Remember that we distinguish between primary norms that are triggered because of a particular condition of the brute state and secondary norms that are triggered due to the violation of another norm. The condition of primary norms thus relates to brute facts, whereas the condition of norms at level one and beyond relates to violation and obedience facts as stored in the institutional state. We acknowledge that norm schemes can contain more components as, for example, explained in (Lopez y Lopez et al., 2006; Vázquez-Salceda et al., 2004). Nevertheless, we limit ourselves to the elementary components that are needed to explain the semantics of the enforcement mechanism. Other components such as addressees and beneficiaries and rewards can be incorporated without any technical difficulties, but including them will obfuscate the explanation of the norms’ semantics.

The norm schemes of the reviewing system, expressing the desired behavior of the agents playing the role of reviewer, author or chair are listed in code fragment 4.1. The first norm scheme expresses that uploaded papers should not exceed the page limit of 15 pages. Suppose an author, say jane, has uploaded an abstract and has been assigned id. 547. As soon as the chair puts the system in the submission phase the condition is satisfied and the norm scheme is instantiated
**Code fragment 4.2 Norms and sanctions of registration system**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Norms:</strong></td>
<td></td>
</tr>
<tr>
<td><code>author_early_reg(A):</code></td>
<td>1</td>
</tr>
<tr>
<td><code>&lt; author(A) and phase(early) and not registration(A,author,complete)</code></td>
<td>2</td>
</tr>
<tr>
<td><code>O(registration(A,author,complete))</code></td>
<td>3</td>
</tr>
<tr>
<td><code>. phase(registration)</code></td>
<td>4</td>
</tr>
<tr>
<td><strong>Sanctions:</strong></td>
<td>5</td>
</tr>
<tr>
<td><code>viol(author_early_reg(A)) =&gt; {Send(A,request,register)}</code></td>
<td>6</td>
</tr>
<tr>
<td><code>fulfill(author_early_reg(A)) =&gt; {free_banquet_ticket(A)}</code></td>
<td>7</td>
</tr>
</tbody>
</table>

into a norm instance pertaining to a prohibition $F(\text{page}\_\text{size}(547) > 15)$ stating that jane’s paper is not allowed to exceed 15 pages. This prohibition stays into effect during the whole submission phase, i.e. until the review phase starts which is expressed by the deadline `phase(review)`. A violation is detected as soon as jane uploads a paper of more than 15 pages. Note that a norm scheme may instantiate multiple norm instances; they are implicitly universally quantified in the widest scope.

To be able to make decisions about which papers to accept and which ones to reject during the collect phase, reviewers are required to have uploaded their reviews before the end of the reviewing phase. This is specified by the second norm scheme, which states that a reviewer is obliged to have uploaded its assigned reviews before the reviews are collected. This obligation becomes active as soon as a reviewer is assigned a paper and stays into effect until it is either fulfilled (the review has been uploaded) or it is violated (the review has not been uploaded before the deadline).

To increase the quality of the reviewing process, the third norm specifies that by the end of the reviewing phase there should be at least three reviews per paper. In the sub-ideal situation this norm is violated, the chair is expected to take charge of reviewing this paper as expressed by the fourth norm scheme. To explain this norm scheme, suppose that when the review phase has ended still only two reviews for paper 547 have been received. Then the obligation $O(\text{nr_reviews(547)} >= 3)$ that emanated from the second norm scheme is violated. This violation is marked by the assertion of an institutional fact `viol(minimum_reviews(547))` to the institutional state. This will trigger the fourth norm scheme, that will instantiate an obligation for the chair to have uploaded a review for paper 547 before the notification phase starts. Note how parameters of labels are used to pass information between norm schemes at different levels. It should be emphasized that the parameters of the labels are formal output parameters that become actual parameters upon instantiation of the norm scheme. By allowing the condition of a norm scheme to contain institutional facts that contain information about norm violations, we can express obligations and prohibitions that are effectuated when certain norms are broken (or abided by).
Remember that in our example participants should first register themselves to the registration system (authors of accepted papers are automatically registered by the reviewing system). Registering alone is not enough, a registration is considered completed if the conference fee has been paid for. Authors of accepted papers are requested to complete their registration in time, preferably already during the early registration. This is expressed by the first norm scheme of code fragment 4.2 specifying that authors should complete their registration during the early registration phase. Early registration is preferred, but not a hard requirement. In fact, as we shall see later on, no punishment is associated to the violation of this obligation.

Sanctioning Rules

To make agents respect the norms, sanctions are often imposed to punish infringements or reward obedience. Sanctions (taken from (Dastani et al., 2008, 2009; Tinnemeier et al., 2009a)) are expressed separately from the norms as a kind of inverted counts-as rules. They associate a sanction with a normative assessment of the artifact’s situation as modeled by the brute and institutional facts. More concretely, they are expressed as rules of the form $\phi \Rightarrow \Psi$ which can be intuitively read as: “a normative situation (as modeled by the brute and institutional facts) expressed by $\phi$ is sanctioned by accommodating brute facts $\Psi$ to the artifact’s brute state.”

The sanctioning rules of the reviewing system are listed on lines 22-27 of code fragment 4.1. The first sanctioning rule of the reviewing system states that reviewers who fail to have uploaded their reviews in time are blacklisted, i.e. a decrease in reputation. This information could be used in deciding who should be on the programme committee of a future conference. The same sanction is applied to the chairs who do not conform to the obligation to review a paper for which less than three reviews have been uploaded. This is expressed by the third sanctioning rule. Again, note how the parameters of the norm scheme in combination with the violation (and obey) facts are used to pass on information about the actual norm instance that has been violated (or abided by). The third sanctioning rule of the reviewing system is an example of regimentation, cf. (Aldewereld, 2007; Castelfranchi, 2000; Grossi, 2007; Jones and Sergot, 1993). Recall that regimentation boils down to ruling out all the actions that will lead to an intolerable state, such that a violation of the norms will never happen. The fact that this state is intolerable is denoted by the special atom $\text{false}$. The third sanctioning rule thus says that it is impossible for an author to upload a paper that exceeds the page limit. Even though an author might still try to upload a paper of more than fifteen pages, the effect of this action will not be effectuated by the organizational artifact.

Recall the norm specifying that authors of accepted papers should register for the conference, preferably during early registration. To motivate these authors to register already during early registration, we associate a reward to the fulfill-
ment of the obligation to register before the registration phase starts (i.e., the first norm of the registration system.) The reward that is imposed is a free ticket for the banquet, which is presumably handed out at the registration desk. As a consequence of violating this obligation a reminder to register is sent to the author concerned. Because this norm pertains to a suggestion expressing preferable behavior instead of required behavior, no punishment is involved. Both sanctions are expressed by the first and second sanctioning rule listed on lines 7-11 of code fragment 4.2.

Ideally, an author who fails to register before the registration system closes is punished by automatically rejecting all its formerly accepted papers. Expressing such a norm and associated sanction, however, turns out not to be trivial. Information about the registration of authors is stored in the registration artifact. This suggests that the norm which obliges authors to register should be implemented as part of the registration artifact. This registration artifact, however, does not have information about which papers belong to that author. This information is stored by the reviewing artifact. The underlying problem is that the information that is needed to enforce this rule is spread across different artifacts. In this case we can overcome this problem by (for example) letting the registration artifact inform the reviewing artifact about the registration status of authors. Then, the norm and sanctioning rule can be implemented as part of the reviewing system. However, this introduces the storage of redundant information, which may lead to nasty synchronization issues. Besides, we can also imagine comparable problems that are not so easily fixed by letting artifacts exchange information. Suppose, for example, we would break up our reviewing system in different organizational artifacts each implementing the reviewing functionalities of a conference sub-track. How to express a norm scheme specifying that a reviewer can only be on the program committee for one sub-track or a norm scheme that specifies that not more than a certain amount of papers may be accepted for the whole conference is unclear. We believe the problem to be a fundamental one; we need norms (and sanctioning rules) to overspan multiple artifacts. This issue is left for future research.

4.3 Executing the Normative Dimension

In the previous section we have defined the syntax of the language by which the normative component of organizational artifacts can be specified and have shown an example of such an implementation. In this section we explain how the normative component of artifacts is executed by endowing the syntax with an operational semantics (Plotkin, 1981).
Intended Semantics of Obligations and Prohibitions

Before we proceed with defining the formal operational semantics of our norm enforcement mechanism, we first recapitulate and motivate the intended semantics of norm schemes and their instances, i.e. obligations and prohibitions. Once we have defined the formal operational semantics, we return to the intended semantics described in this section by proving that the norm schemes and their instances indeed behave as we claim. The semantics we attribute to norm instances is not completely designed by ourselves. In fact, the semantics of obligations is based on the one of Boella et al. (2008), in which the logic of conditional obligations with deadlines is presented. This choice is motivated by the fact that the way in which their semantics is presented is already close to an operational one. The choice for the semantics of prohibitions is mainly driven by some elementary properties from the field of deontic logic and pragmational considerations.

As discussed before, a norm scheme instantiates a norm instance (obligation or prohibition) when its condition is satisfied. We write a norm instance as a tuple $(\phi_l, M\varphi_x, \varphi_d)$ with $M$ either $O$ to denote an obligation or $F$ to denote a prohibition, $\phi_l$ the parameterized label identifying the norm scheme, $\varphi_x$ the description of the state to be achieved or avoided before deadline $\varphi_d$. Once instantiated the behavior of an obligation and a prohibition differs as illustrated by figure 4.6.

Figure 4.6a shows the behavior of an instantiated obligation. As can be seen, the obligation stays into effect as long as it is not fulfilled or violated. An obligation is fulfilled when the state denoted by it can be entailed by the brute (and institutional) state before or when the deadline is reached. Fulfillment of an obligation labeled $\phi_l$ is marked by the assertion of a fact $\text{obey}(\phi_l)$ to the institutional state, whereas an institutional fact $\text{viol}(\phi_l)$ denotes a violation of $\phi_l$. It is violated when $\varphi_x$ has not been established from the state the obligation was instantiated up to the state in which the deadline can be entailed. Note that this means that when the brute state is changed such that deadline $\varphi_d$ starts to hold and desired state $\varphi_x$ starts to hold the obligation is considered to be fulfilled, instead of violated. From a conceptual point of view, one can argue that an action was undertaken to reach the desired state before the deadline held, and therefore the obligation should be considered achieved. The foremost argument, however, is of a technical nature. In this semantics we can express so called instantaneous obligations (see also (Broersen and van der Torre, 2007)). Instantaneous obligations are of the form $(\phi_l, O(\varphi_x), \top)$, i.e. $\varphi_x$ should be established now. If we would have chosen a semantics in which an obligation is fulfilled if, and only if, it is reached before the deadline, an instantaneous obligation $(\phi_l, O(\varphi_x), \top)$ would be vacuously violated! Our semantics of an obligation resembles the one presented in (Boella et al., 2008).

The behavior of a prohibition is shown in figure 4.6b. Unlike an obligation, a prohibition stays into effect until its deadline $\varphi_d$ is reached. That is, a prohibition is removed only when its deadline $\varphi_d$ holds irrespective of whether it is violated.
Figure 4.6: Some possible evolutions of an organizational artifact and the corresponding behavior of a norm instance \((\phi_l, M(\varphi_x), \varphi_d)\) instantiated out of a norm scheme \(\phi'_l : \langle \varphi'_c, M(\varphi'_x), \varphi'_d \rangle\). Each circle denotes a possible state of the artifacts that evolves from one state into another by actions performed by agents. The relevant brute state is shown inside of the states, the effectuated norm instances above of them, and the institutional facts \(\text{viol}\) and \(\text{obey}\) below. States in which it is concluded that the norm instance is respected are colored white, whereas violation states are colored black. Gray states denote the situations where the norm instance is not yet violated or abided by.
Note that this implies that one and the same prohibition can be violated more than once. It is violated when the state described by $\varphi_x$ can be entailed before or when the deadline starts to hold. It is abided by when $\varphi_x$ cannot be entailed by the brute and institutional state during the period starting from when the prohibition was effectuated up to the moment the deadline holds for the first time.

The fact that a prohibition $\phi_l$ has been respected is denoted by the assertion of a fact $\text{obey}(\phi_l)$, whereas a violation is denoted by an institutional fact $\text{viol}(\phi_l)$. Based on a similar reasoning as that for an obligation, we chose to consider an action that simultaneously establishes the prohibition’s deadline $\varphi_d$ as well as the undesired state $\varphi_x$ as a violation. The action leading to $\varphi_x$ was performed at a moment when establishing it was still forbidden. Moreover, we can now meaningfully express instantaneous prohibitions ($\phi_l, F(\varphi_x), \top$) meaning that $\varphi_x$ is forbidden only in the present state.

As observed before, a prohibition can in principle be violated more than once. Suppose, for example, that $\varphi_x$ is established in multiple states until deadline $\varphi_d$. Then in each such state a violation as denoted by institutional fact $\text{viol}(\phi_l)$ can be inferred. To express a prohibition that disappears after it has been violated is to ensure that deadline $\varphi_d = \varphi_x$. In this case a violation will be detected because an action has been performed that led to a forbidden state, and because the deadline is now established, the prohibition will be revoked. Note, however, that this implies that the prohibition will be in effect as long as it is respected. Due to the lack of disjunctions in the deadline it is in general not possible to express a prohibition that stays into effect until its deadline $\varphi_d \neq \varphi_x$ is established or until it is violated.

Although a prohibition can be violated multiple times, it can only be fulfilled once. A prohibition is abided by when during the whole time the prohibition was in effect, the forbidden state $\varphi_x$ is not established. To illustrate the intuition behind this consider, for example, we forbid someone to be in the park between sunset and sunrise. Suppose that someone walked through the park at night, but has left the park before sunrise. Would we now at sunrise – when the prohibition ceases to hold – conclude that this person has respected the prohibition? We would say no, the prohibition is only abided by if the person has not entered the park the whole night. In other words, once a prohibition is violated it cannot be abided by anymore. So, to conclude whether a prohibition is abided by or not, information about past violations becomes relevant.

**Preliminaries**

To understand the execution of the normative component, we extend the previously defined configuration of an organizational artifact to include norm schemes, norm instances and sanctioning rules also. However, before doing so, we provide some auxiliary functions regarding norm schemes and norm instances. They will facilitate defining the configuration of an organizational artifact and the transition rules explaining how an organizational artifact configuration may evolve.
We start with defining the notion of instantiating a norm scheme into a norm instance. As explained before, a norm instance is instantiated from its norm scheme when its condition is derivable from the brute and institutional state for some substitution of its formal parameters. Instantiating a norm scheme is then to apply this substitution on it resulting in a norm instance. Norm instances denoting a prohibition are annotated with extra information about past violations. This information is needed later on to enshrine obedience. For now, an annotation \( \bot \) means that the prohibition has not been violated.

**Definition 4.1 (Scheme Instantiation)** Let \( ns = \phi_l(\overline{v}_1) : \langle \phi_c(\overline{v}_2), \underline{M}(\phi_x(\overline{v}_3)), \phi_d(\overline{v}_4) \rangle \) be a norm scheme with \( \underline{M} \) either an obligation \( O \) or prohibition \( F \) and \( \overline{v}_1, \ldots, \overline{v}_4 \) the sets of variables occurring in the formulae. Then the function \( \text{inst}(ns, \theta) \) that instantiates a norm instance from norm scheme \( ns \) given a formal substitution \( \theta \) for the variables is defined as:

\[
\text{inst}(ns, \theta) = \begin{cases} 
(\phi_l(\overline{v}_1)\theta, O(\phi_x(\overline{v}_3)\theta), \phi_d(\overline{v}_4)\theta) & \text{if } \underline{M} = O \\
(\phi_l(\overline{v}_1)\theta, F(\phi_x(\overline{v}_3)\theta), \phi_d(\overline{v}_4)\theta) \circ \bot & \text{if } \underline{M} = F
\end{cases}
\]

The substitution that is applied in instantiating a norm scheme is obtained via its condition. In what follows we demand norm instances to be ground, i.e. no variables to occur in them. To ensure a norm instance to be ground, we demand each variable that occurs in the norm scheme also to occur in the condition. Having unground norm instances raises the question whether the variables occurring in them are existentially or universally quantified. This question and a possible extension of the language to include quantifiers is left for future research. To ensure norm instances to be ground, we demand each variable that occurs in the norm scheme also to occur in its condition. This is formally defined by the next definition introducing well-formedness of norm schemes.

**Definition 4.2 (Well-formedness of Norm Schemes)** Given a norm scheme \( ns \) of the form \( \phi_l(\overline{v}_1) : \langle \phi_c(\overline{v}_2), \underline{M}(\phi_x(\overline{v}_3)), \phi_d(\overline{v}_4) \rangle \) such that \( \underline{M} \) either an obligation \( O \) or prohibition \( F \) and \( \overline{v}_1, \ldots, \overline{v}_4 \) the sets of variables occurring in the formulae. We say that \( ns \) is well-formed if and only if it holds that \( \overline{v}_1 \cup \overline{v}_3 \cup \overline{v}_4 \subseteq \overline{v}_2 \).

Having defined some auxiliary functions and knowing how (well-formed) norm schemes are instantiated we are now in a position to define the configuration of an organizational artifact with a normative component. This definition extends the previous definition of an organizational artifact as defined in chapter 3. For the sake of readability we repeat all the previously defined components here.

**Definition 4.3 (Artifact Configuration)** An organizational artifact, typically denoted by \( O \), is defined as a tuple \( \langle id, \sigma_b, \sigma_i, E, \Delta, \delta, \Sigma, \epsilon_{in}, \epsilon_{out} \rangle \), in which:

- \( id \) is a name uniquely identifying the artifact;
• $\sigma_b$ is a set of ground, first-order atoms describing the brute state of the artifact;

• $\sigma_i$ is a set of ground, first-order atoms describing the artifact’s institutional state;

• $E$ is a set of effect specifications;

• $\Delta$ a set of well-formed norm schemes;

• $\delta \subseteq \{\text{inst}(ns, \theta) | ns \in \Delta \text{ and } \theta \text{ a ground substitution of the variables}\}$, i.e. a set of ground norm instances defining the active obligations and prohibitions;

• $\Sigma$ the set of sanctioning rules by which the punishments and rewards associated to norm compliance and violation are defined;

• $\epsilon_{in}$ is a list of ground first-order atoms, the events perceived by the artifact;

• $\epsilon_{out}$ is a list of ground first-order atoms either with predicate name $\text{Send}$ or $\text{Do}$, the communication messages to be sent and external actions to be performed.

Just like in previous chapter we define the notion of an initial artifact configuration. This configuration redefines the one previously defined in chapter 3. The initial organizational artifact is the one that is determined by the program code. The artifact’s program defines the artifact’s name, its initial brute state, the effect specifications, its norm schemes and their associated sanctions. Initially, no events are received, no events need to be sent, no obligations and prohibitions have been instantiated, and no norms have been complied with or violated and no roles have been enacted. This is defined by the following definition.

**Definition 4.4 (Initial Artifact Configuration)** An initial artifact configuration is an artifact $⟨id, \sigma_b, \emptyset, E, \Delta, \emptyset, \emptyset⟩$ specified by a program such that $id$ is specified by the program’s name component, $\sigma_b$ is characterized by the program’s facts component, $E$ is defined by the program’s effect component, $\Delta$ is determined by the program’s norm component and $\Sigma$ is defined by the program’s sanctions.

**Execution of Norm Schemes and Their Instances**

Having defined the organization configuration we can now define the transition rules that specify how an organizational artifact may evolve in the light of norm schemes that trigger obligations and prohibitions, and monitoring of the compliance of these instantiated norm schemes. In what follows we assume an artifact configuration $⟨id, \sigma_b, \sigma_i, E, \Delta, \delta, \Sigma, \epsilon_{in}, \epsilon_{out}\rangle$. The components that will not change during computation will be omitted, viz. the artifact’s $id$, its effects specification $E$, its norm schemes $\Delta$ and its sanctioning rules $\Sigma$. 

104
We start with the transition rule explaining how norm schemes are triggered. Call to mind that a norm scheme is triggered when its condition can be entailed by the artifact’s brute and institutional state. Triggering a norm instance is to create a norm instance (as defined by the instantiation function) which is added to the set of norm instances. Triggering is defined by the next transition rule.

**Transition Rule**  Given ground substitution $\theta$, the rule for triggering of norm schemes is defined as:

$$\text{ns} = (\phi_l : \langle \varphi_c, M(\varphi_x), \varphi_d \rangle) \quad \text{ns} \in \Delta \quad \sigma_b \cup \sigma_i \models \varphi_c \theta \quad ni = \text{inst}(\text{ns}, \theta)$$

$$(\sigma_b, \sigma_i, \delta, \epsilon_{in}, \epsilon_{out}) \rightarrow_{\text{org}} (\sigma_b, \sigma_i, \delta \cup \{ni\}, \epsilon_{in}, \epsilon_{out})$$

We start with the transition rule explaining how norm schemes are triggered. Call to mind that a norm scheme is triggered when its condition can be entailed by the artifact’s brute and institutional state. Triggering a norm instance is to create a norm instance (as defined by the instantiation function) which is added to the set of norm instances. Triggering is defined by the next transition rule.

**Transition Rule**  Given ground substitution $\theta$, the rule for triggering of norm schemes is defined as:

$$\text{ns} = (\phi_l : \langle \varphi_c, M(\varphi_x), \varphi_d \rangle) \quad \text{ns} \in \Delta \quad \sigma_b \cup \sigma_i \models \varphi_c \theta \quad ni = \text{inst}(\text{ns}, \theta)$$

$$(\sigma_b, \sigma_i, \delta, \epsilon_{in}, \epsilon_{out}) \rightarrow_{\text{org}} (\sigma_b, \sigma_i, \delta \cup \{ni\}, \epsilon_{in}, \epsilon_{out})$$

Next, we define the rules specifying the (non)compliance of an obligation. When there is an obligation $(\phi_l, \varphi_c, O(\varphi_x), \varphi_d)$ in a certain configuration and the deadline $\varphi_d$ has passed, but $\varphi_x$ has not been achieved yet then this obligation is violated. Remember that violated obligations are removed from the set of norm instances. An obligation is achieved in this configuration when the state denoted by $\varphi_x$ can be entailed by the brute and institutional state. Also in this case the obligation is removed from the set of norm instances. This is defined by the following two transition rules. The first pertains to violation, the second to fulfillment of an obligation.

**Transition Rule**  The rule for violation of an obligation is defined as follows:

$$ni = (\phi_l, O(\varphi_x), \varphi_d) \quad ni \in \delta \quad \sigma_b \cup \sigma_i \not\models \varphi_x \quad \sigma_b \models \varphi_d$$

$$(\sigma_b, \sigma_i, \delta, \epsilon_{in}, \epsilon_{out}) \rightarrow_{\text{org}} (\sigma_b, \sigma_i \cup \{\text{viol}(\phi_l)\}, \delta \setminus \{ni\}, \epsilon_{in}, \epsilon_{out})$$

**Transition Rule**  The rule for fulfillment of an obligation is defined as follows:

$$ni = (\phi_l, O(\varphi_x), \varphi_d) \quad ni \in \delta \quad \sigma_b \cup \sigma_i \models \varphi_x$$

$$(\sigma_b, \sigma_i, \delta, \epsilon_{in}, \epsilon_{out}) \rightarrow_{\text{org}} (\sigma_b, \sigma_i \cup \{\text{obey}(\phi_l)\}, \delta \setminus \{ni\}, \epsilon_{in}, \epsilon_{out})$$

Just like obligations we define the transition rules that specify when a prohibition is (not) abided by. When there is a prohibition $(\phi_l, \varphi_c, F(\varphi_x), \varphi_d)$ in a certain configuration and the deadline $\varphi_d$ has not passed, but $\varphi_x$ has been achieved, then this prohibition is violated. As opposed to an obligation, a violated prohibition remains in effect. That is, it is not removed from the set of norm instances. A prohibition is abided by when deadline $\varphi_d$ holds and $\varphi_x$ has not been established during the whole period for which the prohibition was in effect. In defining the transition rules by which we can derive a transition $\gamma_k \rightarrow \gamma_{k+1}$ we cannot refer to the trace $\gamma_0 \rightarrow^* \gamma_{k-1}$ that was used to reach $\gamma_k$. But in defining the transition rule for obedience of a prohibition we still need to know if the prohibition was violated in the past. Recall that, to have this information at our disposal in defining
the transition for obedience, we annotate a prohibition \( ni \) with information about past violations by adding flags \( \perp \) (not violated) and \( \top \) (violated). In concrete, we write \( ni \circ \top \) to denote that prohibition \( ni \) has been violated, whereas annotation \( ni \circ \perp \) means that \( ni \) has been abided by up to now. Note that obligations do not need this annotation, because for detecting the fulfillment of an obligation we only need the present state (cf. transition rule \( obey_o \)).

The following two transition rules define when a prohibition is not complied with. The first rules amounts to the case in which a prohibition is violated before its deadline holds. In this case the prohibition will stay in effect as its deadline is not yet reached, i.e. it is not removed from the set of norm instances. Besides asserting a violation fact to denote its violation, also its flag is set to \( \top \) to record that it has been violated (and remember this during the rest of its lifetime). The second rule pertains to the case in which a prohibition is violated just when the deadline holds. In this case it is removed and a violation fact is added to the institutional state.

**Transition Rules**  
The rules for violation of a prohibition are defined as follows:

\[
\begin{align*}
\text{ni} & = (\varphi_l, \varphi_c, F(\varphi_x), \varphi_d) \quad (\text{ni} \circ s) \in \delta \quad \sigma_b \cup \sigma_i \models \varphi_x \quad \sigma_b \not\models \varphi_d \\
\langle \sigma_b, \sigma_i, \delta, \epsilon_{in}, \epsilon_{out} \rangle & \xrightarrow{\text{org}} \langle \sigma_b, \sigma_i \cup \{\text{viol(}\varphi_l\}), (\delta \setminus \{\text{ni} \circ s\}) \cup \{\text{ni} \circ \top\}, \epsilon_{in}, \epsilon_{out} \rangle \\
& \quad \text{(viol1p)}
\end{align*}
\]

\[
\begin{align*}
\text{ni} & = (\varphi_l, \varphi_c, F(\varphi_x), \varphi_d) \quad (\text{ni} \circ s) \in \delta \quad \sigma_b \cup \sigma_i \models \varphi_x \quad \sigma_b \models \varphi_d \\
\langle \sigma_b, \sigma_i, \delta, \epsilon_{in}, \epsilon_{out} \rangle & \xrightarrow{\text{org}} \langle \sigma_b, \sigma_i \cup \{\text{viol(}\varphi_l\}), \delta \setminus \{\text{ni} \circ s\}, \epsilon_{in}, \epsilon_{out} \rangle \\
& \quad \text{(viol2p)}
\end{align*}
\]

The following two transition rules handle the case in which the deadline holds, but the prohibition is abided by, at least at that moment. More specifically, the first transition rule pertains to the case in which the deadline holds of a prohibition that is momentarily not violated and has not been violated in the past (as denoted by its annotation \( \perp \).) In this case the prohibition is removed from the set of norm instances and an institutional fact recording its obedience is asserted to the institutional facts. When this prohibition was not respected in the past, i.e. it has been violated before (as indicated by its annotation \( \top \)), such an obedience fact is not added to the institutional state. This is expressed by the second transition rule.

**Transition Rules**  
The rules for detecting compliance to a prohibition at the deadline are defined as follows:

\[
\begin{align*}
\text{ni} & = (\varphi_l, \varphi_c, F(\varphi_x), \varphi_d) \quad \text{ni} \circ \perp \in \delta \quad \sigma_b \cup \sigma_i \not\models \varphi_x \quad \sigma_b \models \varphi_d \\
\langle \sigma_b, \sigma_i, \delta, \epsilon_{in}, \epsilon_{out} \rangle & \xrightarrow{\text{org}} \langle \sigma_b, \sigma_i \cup \{\text{obey(}\varphi_l\}), \delta \setminus \{\text{ni} \circ \perp\}, \epsilon_{in}, \epsilon_{out} \rangle \\
& \quad \text{(obey1p)}
\end{align*}
\]

\[
\begin{align*}
\text{ni} & = (\varphi_l, \varphi_c, F(\varphi_x), \varphi_d) \quad \text{ni} \circ \top \in \delta \quad \sigma_b \cup \sigma_i \not\models \varphi_x \quad \sigma_b \models \varphi_d \\
\langle \sigma_b, \sigma_i, \delta, \epsilon_{in}, \epsilon_{out} \rangle & \xrightarrow{\text{org}} \langle \sigma_b, \sigma_i \setminus \{\text{ni} \circ \top\}, \epsilon_{in}, \epsilon_{out} \rangle \\
& \quad \text{(obey2p)}
\end{align*}
\]
As can be seen from the definition of above transition rules, the normative judgment about the organizational artifact, i.e. information about (non)conformity to the norms, is stored by the institutional facts. As we shall see in a moment, sanctioning rules are applied based on these institutional facts to punish infringements and reward obedience. The institutional facts are designed such that, by default, they do not contain any information about when a norm has been violated or abided by. The intended meaning of violation (and obedience) facts is that a norm is violated (or abided) by now. That is to say, no information is stored in the institutional facts about past (non)conformance. As we will show later on, the organizational coordination cycle is designed such that, before detecting new infringements and obedience, all institutional facts containing information about (non)conformance are removed first. For this purpose, we introduce a transition rule that takes care of cleaning up the institutional facts by removing all information concerning the normative judgment. Note that the rea facts keeping information about role enactments should not be removed.

**Transition Rule** Let \( \text{pred}(\phi) \) be a function evaluating to the predicate name of a first-order atom \( \phi \). Then the rule for cleaning up institutional facts is defined as follows:

\[
⟨\sigma_b, \sigma_i, \delta, \epsilon_{\text{in}}, \epsilon_{\text{out}}⟩ \rightarrow_{\text{org}} ⟨\sigma_b, \sigma'_i, \delta, \epsilon_{\text{in}}, \epsilon_{\text{out}}⟩ \quad (\text{clean})
\]

where \( \sigma'_i = \sigma_i \setminus \{\phi \mid \phi \in \sigma_i \text{ and } \text{pred}(\phi) \in \{\text{viol, obey}\}\} \)

When, for some reason, past occurrences of non(conformity) should be recorded, for example, to apply sanctions based on past and current violations, this can be achieved by designing the norm schemes such that information about when the norm instance was in effect is stored by the parameters of the norm scheme’s label. Of course, this implies leaving out the previous rule in the organizational coordination cycle. Alternatively, information about past (non)conformities could be stored by the brute facts that are established by the sanctioning rules.

**Executing Sanctioning Rules**

Now we know when a norm has been complied with or has been violated, we define the transition rule that explains how sanctions are applied by executing the sanctioning rules. Recall that a sanctioning rule is applicable when its antecedent is satisfied by the brute and institutional state for some substitution \( \theta \). To apply a sanctioning rule is to assert the facts of its consequent to the brute state of the organizational artifact. That is, if the special atom \texttt{false} is not part of the sanctioning rule’s consequent. Remember that similar to effect specifications, the consequent of a sanctioning rule may contain special atoms \texttt{Send} and \texttt{Do} for sending messages and performing actions, both of which are not asserted to the brute state, but added to the list of out events instead. Recall that we use a colon ‘:’ to denote the operation of appending two lists.
**Transition Rule**  The rule for applying an enforced sanctioning rule is defined as follows:

\[
(\varphi \Rightarrow \Psi) \in \Sigma \quad \sigma_b \cup \sigma_i \models \varphi \theta \quad \text{false} \notin \Psi \\
\langle \sigma_b, \sigma_i, \delta, \epsilon_{in}, \epsilon_{out} \rangle \rightarrow_{org} \langle \sigma_b', \sigma_i, \delta, \epsilon_{in}', \epsilon_{out}' \rangle
\]

where \(\sigma'_b = \sigma_b \cup \{\phi \mid \phi \in \Psi \theta \text{ and } \text{pred}(\phi) \notin \{\text{Send, Do}\}\}\)

\(\epsilon_{out}' = \epsilon_{out} : [\phi \mid \phi \in \Psi \theta \text{ and } \text{pred}(\phi) \in \{\text{Send, Do}\}]\)

The above transition rule denotes the application of a sanctioning rule pertaining to an enforcement strategy of the norms. In other words, the violation of some norm(s) is tolerated, but possibly punished. In case a sanctioning rule is applicable that pertains to a regimentation strategy of some norm(s), i.e. has special atom false in its consequent, the sanction is not materialized. In fact, not only the sanctioning rule is not effectuated, the whole configuration that causes this intolerable situation is dismissed. Note that this is not expressible by one single transition rule, because we cannot refer to the artifact configuration(s) preceding the undesirable configuration that has just been reached by the application of some transition rule(s). We can, however, do exactly that in our coordination strategy language in which we are able to talk about the outcome of a sequence of transitions. To enable us to refer to intolerable configurations reached by the application of some “regimented sanctioning rule” in defining the organizational coordination strategy, we define the following transition rule. A configuration is intolerable if some sanctioning rule with special atom false in its consequent is applicable. To indicate the impossibility of actually applying a sanctioning rule with false in its consequent, we write \(\bot\) to denote a “deadlock” configuration. As we shall see later on, the coordination strategy ensures that such a state is never reached.

**Transition Rule**  The rule for applying a regimented sanctioning rule is defined as follows:

\[
(\varphi \Rightarrow \Psi) \in \Sigma \quad \sigma_b \cup \sigma_i \models \varphi \theta \quad \text{false} \in \Psi \\
\langle \sigma_b, \sigma_i, \delta, \epsilon_{in}, \epsilon_{out} \rangle \rightarrow_{org} \bot
\]

When at some point a regimented sanctioning rule is applicable, we want to roll back to the desirable configuration preceding the intolerable one we just reached. Typically, the effect of some action has been determined (followed by triggering, monitoring and sanctioning), causing the intolerable configuration. We thus want to return to the configuration before the effect of the action was effectuated. However, because at this point we already know that handling the received action would lead to a regimented action, there is no sense in leaving this action on top of the list of received actions when rolling back. Therefore, we introduce the following, auxiliary transition rule that merely removes the head of the list of received actions. We use this transition rule in defining our coordination strategy that accounts for regimentation.
Transition Rule  Let $\alpha$ be an action (including enact and deact) performed by an agent. Then the rule for ignoring this action is defined as follows:

$$\langle \sigma_b, \sigma_i, \delta, [\alpha] : \epsilon_{in}, \epsilon_{out} \rangle \rightarrow_{org} \langle \sigma_b, \sigma_i, \epsilon_{in}, \epsilon_{out} \rangle$$

\[\text{(ignact)}\]

4.4 Organizational Coordination Strategies

Just like previous chapter we only elaborated on the meaning of the programming constructs, without committing to a particular sequence in which they are executed. Different sensible strategies exist, each possibly yielding different outcomes. In this section we discuss some possible organizational coordination strategies by means of the coordination language introduced in previous chapter. The next section, that formally studies some of the elementary properties of the normative component, further elaborates upon the implications different strategies may have.

Call to mind that the strategy language we borrowed from Astefanoaei et al. and Eker et al. (2009a; 2007) includes: construct $l@\gamma$ for applying a transition rule labeled $l$ to a configuration $\gamma$, evaluating to the set of all possible configurations of applying this transition rule; $idle@\gamma$ evaluating to $\{\gamma\}$ and $fail@\gamma$ evaluating to the empty set; sequence operator `;` for sequencing expressions; choice operator $|$ for non-deterministically choosing among two expressions; operator $!$ for repeating an expression until no longer applicable; operator $+$ to repeat an expression one or more times; operator $*$ to repeat an expression zero or more times; and choice operator of the form $E ? E' : E''$ to mean that if expression $E$ does not result in the empty set we apply expression $E'$ on its outcome, otherwise $E''$ on the initial configuration. In what follows we apply some of the strategies that were also discussed by Astefanoaei et al. (2009a) for the counts-as based language on the normative language presented here.

4.4.1 Monitoring and Triggering Strategies

We start with some strategy that focus on the triggering and monitoring process, and do not include sanctioning. We do so, because sanctions complicate the proofs of the propositions we posit next (see also (Tinnemeier et al., 2009c)). As a first strategy consider the following organizational coordination step (that presumably comes after having determined the effect of an action):

$$clean; (trig!; (viol_o|viol1_p|viol2_p)!; (obey_o|obey1_p|obey2_p)!)! \quad \text{(totalitarian)}$$

saying that we first clean information about past violations and obedience, after which we trigger all applicable norm schemes, detect all violations of obligations and prohibitions, and finally detect all obedience of obligations and prohibitions.
The process of triggering and checking for compliance and infringement is repeated, because the violation of or compliance to some norm instance may trigger a secondary norm scheme. This coordination strategy is called totalitarian, because it amounts to a strategy where all norms are enforced.

Remark that $\text{trig}!; (\text{viol}_o | \text{viol}_{1_p} | \text{viol}_{2_p})!; (\text{obey}_o | \text{obey}_{1_p} | \text{obey}_{2_p})!$ always terminates, because the sets of brute and institutional facts and the set of norm schemes are all finite and, consequently, will create a finite set of norm instances. However, it should also be remarked that the repetition of the triggering and monitoring sequence might not terminate at all. Suppose, for example, we have a norm scheme that instantiated an obligation that is instantaneously fulfilled (say), and thus removed. Because the same norm scheme then might again instantiate the same obligation this process would be repeated ad infinitum. Instead of accounting for such situations in the semantics and strategies, it is the responsibility of the programmer to define the schemes’ conditions such that this situation never happens. In (Tinnemeier et al., 2009c) we stratified the norms in different levels such that at level 0 we find norms of which the condition only refers to brute facts, and at each level $k > 0$ we find norms that only trigger because of some violation or fulfillment of a level $k - 1$ norm. Determining (non)conformity and triggering is then done per norm level such that it will always terminate.

In the totalitarian strategy every norm is enforced. That is to say, if a norm scheme is applicable, it will be applied and, consequently, all instances it may create will be instantiated. Moreover, each norm instance is monitored for violation and fulfillment. When there is a large amount of norm schemes creating many norm instances, a totalitarian strategy might not be feasible. As an alternative, a strategy might be deployed in which not all instantiable norm instances are actually created, but only some of them. An example of such a strategy is the following:

$$\text{clean}; (\text{trig}^*; (\text{viol}_o | \text{viol}_{1_p} | \text{viol}_{2_p})!; (\text{obey}_o | \text{obey}_{1_p} | \text{obey}_{2_p})!)^+ \quad \text{(liberal)}$$

that imposes no restrictions on if and how many norm schemes are triggered, i.e. not all norms are enforced. The process of triggering and checking for compliance and infringement is possibly repeated (due to the repetition operator +), because the violation of or compliance to some norm instance may trigger a secondary norm scheme. It should be emphasized that also in this strategy all active obligations and prohibitions are monitored for (non)conformance. This is an important feature in proving the propositions that follow.

Of course, many other sensible coordination strategies are possible, but a thorough investigation of all these strategies is not our aim. By showing these we merely want to point out that different strategies do exist. Our foremost aim is to show that in the coordination strategies totalitarian and liberal, the interpretation of obligations and prohibitions is indeed the one as explained in 4.6. The crux of most of the proofs that will follow is that in both strategies all norm instances are monitored for (non)compliance and that this is always done after triggering norm schemes.
Remember the coordination step ‘action’ we defined in previous chapter for processing the effect of an action and the coordination step ‘out’ for processing all the events (messages and actions) that are stored in the list of out events. In what follows we assume an organizational coordination strategy \( E \) that is either of the form action; out; totalitarian or action; out; liberal. Recall the function \( C_E^O(O) \) (defined in chapter 3) that evaluates to the execution trace that results after iteratively applying coordination strategy \( E \) \( n \) times on organizational artifact configuration \( O \).

What is more, we write \( \sigma_b[j] \) to refer to the brute state of the \( j \)th organizational configuration, i.e. \( O_j \), as part of a coordinated trace \( O_0 \cdots O_n \). We use a similar notation to refer to the other elements of \( O_j \). Although it should be clear from the context, the notation we use for organizational artifacts, viz. \( O \), should not be confused with the modality for obligations, namely an italicized \( O \).

The first three propositions are about the behavior of an obligation. The first proposition shows that an obligation persists as long as it is not achieved and it is still before the deadline. More precisely, it will be in effect if and only if it is not violated or achieved.

**Proposition 4.5** Let \( O_0 \) be an artifact configuration s.t. \( ni = (\phi_1, O(\varphi_x), \varphi_d) \in \delta[0] \) and \( C_E^O(O_0) = O_0 \cdots O_n \) with \( n > 0 \) be a coordinated trace. Then it holds that \( ni \in \delta[n]\) if and only if \( \sigma_b[j] \cup \sigma_i[j] \not\models \varphi_x \) and \( \sigma_b[j] \not\models \varphi_d \) for \( 0 < j < n \).

**Proof**

(\( \rightarrow \)) Given a derivation of length 1 we have to prove that \( ni \in \delta[1] \). The totalitarian strategy ensures that \( ni \in \delta[1] \) because: 1) \( ni \in \delta[0] \) (by assumption); 2) \( \text{viol}_o \) and \( \text{obey}_o \) are the only rules that remove obligations; 3) both rules are not applicable for \( (\phi_1, O(\varphi_x), \varphi_d) \) because \( \sigma_b[1] \cup \sigma_i[1] \not\models \varphi_x \) and \( \sigma_b[1] \not\models \varphi_d \) (by assumption). The proof for the inductive case proceeds by a similar reasoning.

(\( \leftarrow \)) Given a derivation of length 1 we have to prove that \( \sigma_b[1] \cup \sigma_i[1] \not\models \varphi_x \) and \( \sigma_b[1] \not\models \varphi_d \). In the strategy \( E \) this is indeed the case, because: 1) \( ni \in \delta[1] \) (by assumption); 2) transition rule \( \text{trig} \) is applied before \( \text{viol}_o \) and \( \text{obey}_o \) are applied until no longer applicable; and 3) \( ni \) would have been removed by \( \text{viol}_o \) or \( \text{obey}_o \) if either \( \sigma_b[1] \cup \sigma_i[1] \models \varphi_x \) or \( \sigma_b[1] \models \varphi_d \). The proof for the inductive case proceeds by a similar reasoning.

An obligation of which the state denoted by it has not been fulfilled from the moment it was created up to the deadline is marked violated. It is marked fulfilled when the state denoted by it is achieved before or at the state at which the deadline first holds. This is demonstrated by the proposition below. Note that from previous proposition it follows that an obligation is removed upon a violation or fulfillment.

**Proposition 4.6** Let \( O_0 \) be an artifact configuration s.t. \( ni = (\phi_1, O(\varphi_x), \varphi_d) \in \delta[0] \). Then for every coordinated trace \( C_E^O(O_0) = O_0 \cdots O_n \) for \( n > 0 \) s.t. \( \sigma_b[j] \cup \sigma_i[j] \not\models \varphi_x \) and \( \sigma_b[j] \not\models \varphi_d \) for \( 0 < j < n \) it holds that:
(a) if $\sigma_b[n] \cup \sigma_i[n] \not\models \varphi_x$ and $\sigma_b[n] \models \varphi_d$ then $\sigma_i[n] \models \text{viol}(\phi_l)$

(b) if $\sigma_b[n] \cup \sigma_i[n] \models \varphi_x$ then $\sigma_i[n] \models \text{obey}(\phi_l)$

Proof

(a) Given a derivation of length 1 we have to prove that $\sigma_i[1] \models \text{viol}(\phi_l)$. The coordination strategy ensures that $\sigma_i[1] \models \text{viol}(\phi_l)$, because: 1) $\sigma_i \in \delta[0]$ (by assumption); 2) $\sigma_b[1] \cup \sigma_i[1] \not\models \varphi_x$ and $\sigma_b[1] \models \varphi_d$ (by assumption); 3) transition rule $\text{viol}_o$ will be applied (because of strategy $E$ and 1 and 2); transition rule $\text{viol}_o$ ensures that $\sigma_i[1] \models \text{viol}(\phi_l)$. Assume this proposition holds for a derivation of length $k$. To prove that it also holds for a derivation of length $k+1$. That $\sigma_i \in \delta[k]$ follows from proposition 4.5. The rest of the proof is based on a similar reasoning as the base case.

(b) Given a coordinated trace of length 1 we have to prove that $\sigma_i[1] \models \text{viol}(\phi_l)$. The strategy $E$ ensures that $\sigma_i[1] \models \text{obey}(\phi_l)$, because: 1) $\sigma_i \in \delta[0]$ (by assumption); 2) $\sigma_b[1] \cup \sigma_i[1] \models \varphi_x$ (by assumption); 3) transition rule $\text{obey}_o$ will be applied (because of strategy $E$ and 1 and 2); transition rule $\text{obey}_o$ ensures that $\sigma_i[1] \models \text{obey}(\phi_l)$. Assume this proposition holds for a derivation of length $k$. To prove that it also holds for a trace of length $k+1$. That $\sigma_i \in \delta[k]$ follows from proposition 4.5. The rest of the proof is based on a similar reasoning as the base case.

Unlike an obligation, a prohibition is removed if and only if the deadline is reached, irrespective of whether the prohibition is violated or not. This is demonstrated by the proposition that follows next.

Proposition 4.7 Let $O_0$ be an artifact configuration s.t. $ni = (\phi_l, F(\varphi_x), \varphi_d)$ such that $ni \circ s \in \delta[0]$. Then for every coordinated trace $C^n_E(O_0) = O_0 \cdots O_n$ for $n > 0$ it holds that there exists an $s'$ that is either $\top$ or $\bot$ such that $ni \circ s' \in \delta[n]$ if and only if $\sigma_0[j] \not\models \varphi_d$ for $0 \leq j \leq n$.

Proof

($\rightarrow$) Given a derivation of length 1 we have to prove that $ni \circ s' \in \delta[1]$. The coordination strategy ensures that $ni \circ s' \in \delta[1]$ because: 1) $ni \in \delta[0]$ (by assumption); 2) $\text{viol}_2$, $\text{obey}_1$, and $\text{obey}_2$ are the only rules that remove prohibitions; 3) neither of these rules is applicable for $ni$ because $\sigma_b[1] \not\models \varphi_d$ (by assumption). The proof for the inductive case proceeds by a similar reasoning.

($\leftarrow$) Given a derivation of length 1 we have to prove that $\sigma_b[1] \not\models \varphi_d$. In the strategy $E$ this is indeed the case, because: 1) $ni \circ s' \in \delta[1]$ (by assumption); 2) transition rule $\text{trig}$ is applied before $\text{viol}_2$, $\text{obey}_1$, and $\text{obey}_2$ are applied until no longer applicable; and 3) $ni \circ s'$ would have been removed by either $\text{viol}_2$, $\text{obey}_1$, or $\text{obey}_2$ if $\sigma_b[1] \models \varphi_d$. The proof for the inductive case proceeds by a similar reasoning.
A prohibition is violated whenever the forbidden state denoted by it is established before or at the moment at which the deadline holds. It is abided by if its forbidden state has not been established during the whole period the prohibition was in effect. These two properties are shown by the next proposition.

**Proposition 4.8** Let $O_0$ be an artifact configuration s.t. $ni = (\phi_l, F(\varphi_x), \varphi_d)$ and $ni \circ \bot \in \delta[0]$. Then for every coordinated trace $C^*_E(O_0) = O_0 \cdots O_n$ for $n > 0$ s.t. $\sigma_b[j] \not\models \varphi_d$ for $0 \leq j < n$ it holds that:

(a) if $\sigma_b[n] \cup \sigma_i[n] \models \varphi_x$ then $\sigma_i[n] \models \text{viol}(\phi_l)$.

(b) if $\sigma_b[j] \cup \sigma_i[j] \not\models \varphi_x$ for $0 \leq j \leq n$ and $\sigma_b[n] \models \varphi_d$ then $\sigma_i[n] \models \text{obey}(\phi_l)$

**Proof**

(a) Given a trace of length 1 we have to prove that $\sigma_i[1] \models \text{viol}(\phi_l)$. The strategy $E$ ensures that $\sigma_i[1] \models \text{viol}(\phi_l)$, because: 1) $ni \circ \bot \in \delta[0]$ (by assumption); 2) $\sigma_b[1] \cup \sigma_i[1] \models \varphi_x$ (by assumption); 3) either transition rule $\text{viol}_1p$ or $\text{viol}_2p$ will be applied (because of strategy 1 and 2); both transition rules ensure that $\sigma_i[n] \models \text{viol}(\phi_l)$. Assume this proposition holds for a trace of length $k$. To prove that it also holds for a derivation of length $k + 1$. That $ni \circ s \in \delta[k]$ with $s$ either $\top$ or $\bot$ follows from proposition 4.7. The rest of the proof is based on a similar reasoning as the base case.

(b) Given a trace of length 1 we have to prove that $\sigma_i[1] \models \text{obey}(\phi_l)$. The strategy $E$ ensures that $\sigma_i[1] \models \text{obey}(\phi_l)$, because: 1) $ni \circ \bot \in \delta[0]$ (by assumption); 2) $\sigma_b[1] \models \varphi_d$ and $\sigma_b[1] \cup \sigma_i[1] \not\models \varphi_x$ (by assumption); transition rule $\text{obey}_1p$ will be applied (because of strategy and 1 and 2); transition rule $\text{obey}_1p$ ensures that $\sigma_i[1] \models \text{obey}(\phi_l)$. Assume this proposition holds for a derivation of length $k$. To prove that it also holds for a trace of length $k + 1$. Observe that $ni \circ \bot \in \delta[k]$ because of: 1) proposition 4.7 and the fact that rule $\text{viol}_1p$ will never be applied because of the assumption $\sigma_b[j] \cup \sigma_i[j] \not\models \varphi_x$ for $0 \leq j \leq n$. From here on, the rest of the proof is similar as that of the base case.

### 4.4.2 Sanctioning Strategies

Up till now we did not mention sanctions that are applied upon a violation or conformance to a norm. In our framework there are basically two ways to respond to a violation of some norm. The first is to impose punishments on the brute state, whereas the second is to rollback to the situation the artifact was in before the effect of the action causing the violation has been applied. The first strategy pertains to an enforcement of the norms, whereas the latter pertains to the regimentation of the norms (see also chapter 2).

Choosing an enforcement strategy does not exclude the possibility to use regimentation also. Typically, only some norms will be regimented, whereas most norms will be enforced. Whichever strategy is chosen, still many choices remain.
For example, whether to apply every applicable sanctioning rule or to apply only some of them. What is more, different sanctioning strategies can be applied in combination with different monitoring and triggering strategies. Consider, for example, the next coordination strategy:

\[
\text{test}(\text{action}; \text{totalitarian}; \text{reg}) \ ? \ \text{ignact} : \ \text{totalitarian}; \text{sanc!}; \text{out!} \\
\text{(totalitarian-reg)}
\]

saying that a totalitarian strategy will be applied after which all applicable sanctioning rules will be effectuated only if applying the totalitarian strategy does not enable the application of regimentation rules. If a regimented sanctioning rule is indeed applicable the result of this strategy is as if the action would not have been received at all. Observe that this implies that no punishments are carried out. Also note that in case an organizational artifact would not have any regimented sanctioning rules, this strategy would produce the same outcome as the strategy:

\[
\text{action}; \text{totalitarian}; \text{sanc!}; \text{out!} \\
\text{(totalitarian-enf)}
\]

In both of these strategies the totalitarian strategy could be easily replaced by a liberal strategy. Moreover, instead of applying all applicable sanctioning rules we could have chosen to apply only some. Again, the investigation of these and more alternative strategies is beyond the scope of present work. Though, it is interesting to note that in these strategies the propositions 4.5–4.8 do not hold. This is due to the fact that the application of a sanction might establish or undo the $\varphi_d$ and $\varphi_x$ components of a norm instance $(\phi_l, M(\varphi_x), \varphi_d)$. Suppose, for example, we have a norm instance $ni = (\phi_l, O(\varphi_x), \varphi_d)$ for some artifact configuration, and that $\varphi_x$ can be entailed by the brute state. Applying rule $\text{obey}_o$ will then remove this norm instance and mark it as achieved. Now, suppose that some sanctioning rule is applied such that $\varphi_x$ no longer holds, but $ni$ is still removed and considered achieved. For those who think that such interference should be prevented consider the following example. In the conference management system example we regimented the act of uploading a paper exceeding the page limit. Now suppose that we still want to prevent papers of more than fifteen pages from being uploaded, but we also want to punish one for trying (regimentation is thus not an option). Then we could model this by replacing the original regimented sanctioning rule by a rule (with the same antecedent) that removes the paper from the system and at the same time imposes a sanction. The facts that are imposed as sanction thus interfere with the state denoted by the prohibition.

4.5 On the Notion of Normative Conflict and Coherency

Due to the many different representations and meanings of normative concepts that have been proposed in literature, there are also many different conceptions of when a set of norms is said to be conflicting. An overview of some different
notions of normative conflict can be found in the article of Elhag et al. (2000). Standard deontic logic, for example, comprises \( \neg(Op \land O\neg p) \) and \( \neg O(p \land \neg p) \) as theorems. The first stating that one cannot be obliged to do conflicting action, and the second stating that there cannot be contradictory obligations. In standard deontic logic, normative conflicts thus do not exist. However, deriving (by law) that someone is obliged to pay his taxes and is not obliged to pay his taxes seems strange, but might happen in practice (and is why we have judges to interpret the law.) Based on the observation that such situations seem to occur in practice, some have argued that normative conflicts do exist (cf. (Hansen et al., 2007) for a discussion). That normative conflicts may occur, does not imply, however, we should do nothing to prevent them. Indeed, in (Vasconcelos et al., 2007) Vasconcelos et al. propose a mechanism for avoiding normative conflicts. They adopt a representation of norms expressing obligations, prohibitions and permission relating to a single action. They mark a situation in which an agent is simultaneously obliged and forbidden to perform an action as a normative conflict.

Suppose we adopt the same definition as that of (Vasconcelos et al., 2007), meaning that we say that two norms are in conflict when whatever action is undertaken, a violation is inevitable. Because the situations that should be achieved or avoided and the deadlines of our norms range over states that can be achieved, defining when a set of norm instances is conflicting is suddenly not trivial anymore. It seems, for example, that we should mark two norm instances \( (l_1, F(p), q) \) and \( (l_2, O(p), q) \) as conflicting, because a violation is inevitable. However, here a violation is only inevitable in case the deadline \( q \) will eventually hold. A more fundamental difficulty in defining a notion of normative conflict is caused by the fact that both deadline and the state denoted by the norm instances may be related. Consider, for example, three norm instances \( (l_1, O(p), q) \), \( (l_2, O(r), p) \) and \( (l_3, F(r), q) \). To abide by the first obligation we should establish \( p \) before \( q \), but to also abide by the second obligation we should have first achieved \( r \). This implies that we achieve \( r \) before \( q \), but this would violate the prohibition! It somehow seems that from the norm instances \( (l_1, O(p), q) \) and \( (l_2, O(r), p) \) we can derive an obligation \( (l_2, O(r), q), \) which reduces this example to the first example given above. This suggests that to detect normative conflicts we need a derivation mechanism, which is a non-trivial problem in itself.

Even when we have a mechanism at our disposal by which we can make derivations from a set of norm instances, there are still situations in which a violation is inevitable, but are nevertheless hard to detect. Consider, for example, an obligation \( (l_1, O(p), q) \) and a prohibition \( (l_2, F(\neg q), \neg p) \). The only way to avoid a violation is to satisfy both \( p \) and \( q \) at the same time. The fact that a violation can be avoided, means that these two norm instances are not in conflict. Now assume that these two norm instances are directed to a single person, \( p \) stands for “being in room 1” and \( q \) for “being in room 2” with room 1 and room 2 being physically separated rooms. Then a violation is inevitable, because it is physically impossible to be in two rooms at the same time. The problem here is that to conclude that
these two norm instances raise a conflict we need background knowledge about the brute state telling us that \( \neg(p \land q) \). However, such background knowledge is usually not available, at least not in our framework.

Another issue that is closely related to normative conflicts pertains to the coherency of norm schemes (see also (Hansen et al., 2007)). That is, the issue of whether a set of norm schemes generates situations in which a violation is inevitable. If we classify two norm instances \((l_1, F(p), q)\) and \((l_2, O(p), q)\) as conflicting, we should also regard two norm schemes \(l_1 : \langle c, F(p), q \rangle\) and \(l_2 : \langle c, O(p), q \rangle\) as incoherent. Here also the issue of background knowledge plays an important role. Suppose norm schemes \(l_1\) and \(l_2\) had different conditions, say \(c\) and \(c'\). At first sight, these norm schemes seem coherent. But they are incoherent if we know that \(c'\) is always true whenever \(c\) is true, i.e. \(c \rightarrow c'\). Moreover, norm schemes are not only incoherent when they give rise to conflicting norm instances. Consider, for example, a norm scheme \(l_3 : \langle c, F(c), q \rangle\). Triggering this norm scheme leads to a norm instance that in itself does not necessarily lead to a normative conflict. But knowing that the forbidden state is already established when the norm instance is asserted, triggering the norm scheme nevertheless inevitably causes a violation.

Above discussion shows the intricacies of defining the concept of normative conflict and normative coherence. Detecting, resolving or even preventing normative conflicts at runtime or design time is not trivial. We strongly feel that model checking techniques might prove useful in verifying and detecting situations in which violations cannot be avoided given a normative specification. This issue is left for future research.

4.6 Conclusion

In this chapter we enriched our organizational artifacts with a normative dimension. We introduced the syntax of norm schemes pertaining to conditional obligations and prohibitions that are accompanied by a deadline. These norm schemes instantiate (unconditional) obligations and prohibitions when their condition is satisfied. We underpinned the syntax with an operational semantics (Plotkin, 1981), which allowed us to formally investigate some essential properties our obligations and prohibitions exhibit. Where norms in existing frameworks typically take on the form of “ought-to-do” statements pertaining to actions that should (or should not) be performed, our norms take on the form of “ought-to-be” statements pertaining to the declarative description of a state that should be achieved (or avoided). To motivate agents to abide by the norms we introduced a sanctioning mechanism to reward obedience and punish infringements.