On Obligations*

Manuel Hilty, David Basin, and Alexander Pretschner

Information Security, ETH Zürich, Switzerland
{hiltym, basin, pretscha}@inf.ethz.ch

Abstract. Access control is concerned with granting access to sensitive data based on conditions that relate to the past or present, so-called provisions. Expressing requirements from the domain of data protection necessitates extending this notion with conditions that relate to the future. Obligations, in this sense, are concerned with commitments of the involved parties. At the moment of granting access, adherence to these commitments cannot be guaranteed. An example is the requirement "do not re-distribute data", where the actions of the involved parties may not even be observable. We provide a formal framework that allows us to precisely specify data protection policies. A syntactic classification of formulas gives rise to natural and intuitive formal definitions of provisions and obligations. Based on this classification, we present different mechanisms for checking adherence to agreed upon commitments.

1 Introduction

With the ever increasing use of modern communication technologies in the commercial and public sectors, the adequate handling of personal data is a growing concern. Such data is often distributed across many public and commercial databases, and processed by many applications. This opens the door to illegitimate access and misuse of data. To prevent misuse, many countries have passed data protection laws and privacy regulations, but these are often not adequately reflected in the distributed information systems that store and process sensitive data. When building such systems, one needs firstly to be able to precisely define and specify the underlying requirements, and secondly a means of ensuring that personal data is handled in accordance with applicable laws and regulations. To date, neither problem is solved in a fully satisfactory manner.

By controlling who may access which data, traditional access control mechanisms solve one part of the problem. However, they are unable to make decisions based on *how* the data will be used once accessed, and this is an essential aspect of many data protection regulations. For instance, it should be possible to grant read access to personal data under the stipulation that the data be used exclusively for certain predefined purposes. An example is a database maintained by one government agency that must not disclose citizen data to other administrations unless the data is used solely for statistical purposes. Retention, protection,

* This work was partially supported by the SBF Project 03.0468-1, "GUIDE: Creating a European Identity Management Architecture for eGovernment".

S. De Capitani di Vimercati et al. (Eds.): ESORICS 2005, LNCS 3679, pp. 98–117, 2005.

or redistribution of data are further examples of relevant issues in the context of data protection. This paper discusses the handling of *data protection requirements*: statements that, in addition to specifying who may access which data, also impose constraints on how the data may be used. Our formal framework makes it possible to precisely define and specify a large subset of data protection requirements, and, upon implementation, also to control adherence to commitments related to data protection. This makes the framework attractive as a stepping stone to future, practical, data protection specification languages. Naturally, it also gives rise to the possibility of formally analyzing data protection requirements, e.g. to determine their consistency.

Our system model consists of agents that own or request access to data. Access is granted or denied by a reference monitor based on conditions that are expressed in (formalized) data protection requirements. In the form of *provisions*, these conditions can relate to the past or present, e.g. "explicit owner consent must be presented when access to data is granted". In the form of *obligations*, the conditions can also relate to the future, e.g. "data must be deleted within a month", or "the further distribution of the data must be logged". We consider obligations from two perspectives: time and observability, i.e. the possibility of checking adherence to a commitment. Our formalization of obligations leads to a natural classification along these two dimensions.

1. Time: Obligations are either concerned with fixed time intervals, or are defined as eternally valid conditional statements—invariants. We will argue that unbounded eventuality—roughly, liveness—is not relevant in the context of data protection.

2. Observability: We address the problem exemplified by the obligation to delete data some time after it has been obtained from the reference monitor. From the monitor's perspective, this is usually impossible to enforce and because it is not directly observable, difficult to check. We show how to transform non-observable events into observable ones, which then can be checked by the reference monitor. Upon violation of an obligation, the reference monitor can take necessary actions.

Existing generalizations of traditional access control models cannot capture the entirety of the above data protection issues. Some omit certain classes of data protection requirements and thus leave room for generalization; others do not provide a precise semantics for their frameworks (Section 5).

To summarize, the *problem* that we tackle is the lack of precise definitions of data protection requirements (that necessarily precede a practically useful specification language for data protection policies) and, in particular, the lack of a precise understanding of how adherence to obligations can be checked. Our *solution* consists of a conceptual and a methodological part. Conceptually, to precisely define and understand the cornerstones of data protection requirements, we show how to specify them in a logic that adds the dimension of distribution to classical temporal logics. This allows for the intuitive distinction between provisions and obligations on the one hand, and between observability and non-

observability on the other. Methodologically, we show how to specify mechanisms that rely on the transformation of non-observable parts of an obligation into observable ones. Overall, we see our *contribution* as (a) the extension of access control models in the context of data protection with precisely defined concepts, (b) a classification of data protection requirements along the dimensions of time and observability, and (c) the description of generic mechanisms to cope with non-observable parts of obligations.

The remainder of this paper is organized as follows. We take an informal look at data protection and set the scene at the beginning of Section 2. Then we introduce Distributed Temporal Logic (DTL) [5,6], which we use to define our formal framework and to classify data protection requirements. In Section 3 we show how non-observable requirements can be transformed into requirements that can be checked by a central reference monitor. We use Section 4 to illustrate the ideas of this paper with an example. In Sections 5 and 6 we describe related work and our conclusions. Formal details on DTL can be found in the appendices.

2 Data Protection Requirements

In this section, we examine data protection requirements. First we introduce the kinds of problems we are concerned with from an informal point of view. Then we present our formalism and show how it can be used to formalize data protection policies. We also present a classification of data protection requirements based on their temporal structure and the kinds of observations that different principals in a distributed system can make.

2.1 An Informal View on Data Protection

Privacy is concerned with anonymity and data protection. Our work focuses on data protection, i.e. controlling the access to and the usage of sensitive personal data. By personal data, we denote any data that is, or can be, associated with a person. We consider systems consisting of two kinds of agents: a server reference monitor r and a set of subjects Sub. Dat is a set of personal data objects. The reference monitor[1] controls access to the data items in Dat. We consider scenarios where a subject $s_i \in Sub$ attempts to access a data object $d_k \in Dat$. In this case, we also refer to s_i as the *requester*.

We examine policies that define how to control the flow of sensitive data in a distributed environment (this involves access control decisions made by r as well as controlling the further distribution of data between the subjects), and how the data must be used by the subjects. Sources for the different kinds of requirements were existing privacy policy description languages like P3P [19], scientific papers such as [11], and data protection regulations from different countries.

Data protection requirements include (1) access control requirements, which should be as flexible as possible, i.e. able to express different access control

[1] We will use the term reference monitor rather liberally to describe control programs that can not only monitor and prohibit actions, but can also trigger corrective actions such as the application of penalties.

paradigms; (2) actions that must be performed prior to the access, e.g. presenting a certificate or gathering the consent of the data owner; (3) actions that must be performed within a certain time period or on a recurring basis, e.g. informing the data owner about the access or about any usage of the data; (4) restrictions on the further distribution of the data; (5) restrictions on the possible purposes for which the data may be used; (6) limitations on the retention time; (7) mandatory uses of protection mechanisms; and (8) duties of keeping the data up-to-date.

2.2 Gentle Introduction to DTL

We use Distributed Temporal Logic (DTL) to formally express privacy requirements. In this section, we give a brief, high-level overview of DTL, relegating the formal presentation to Appendix A.

DTL [5,6] can be seen as a generalization of Linear Temporal Logic (LTL) [13] and our explanation here will focus on the differences. DTL contains temporal operators for reasoning about the past and the future. In addition, it provides a distributed view of systems. Whereas LTL formulas express a global view of a distributed system in terms of temporal properties of its traces, DTL formulas formalize system properties from the local view of the system's agents. Statements φ refer to an agent a's local data space; this is expressed as $@_a[\varphi]$. $@.[\cdot]$ formulas cannot be nested. Intuitively, φ includes *state propositions* such as "a possesses a certificate", and *actions* such as "a deletes a file". φ may also express temporal behavior relative to the agent's past or future. Action symbols and state propositions are domain dependent and must be defined for each concrete scenario. Examples are given later in this section and in Section 4.

Agents a and b can communicate messages time-synchronously and reliably via *snd/rcv* pairs, which are parts of their respective set of local actions: the action $snd(b, m)$ denotes the sending of message m to agent b and $rcv(b, m)$ denotes the simultaneous reception of message m from agent b.

DTL future-time operators are the unary operators X (*next*) and G (*always*), and the binary U operator (*weak until*). DTL past-time operators are the unary operators Y (*previous*), P (*sometime in the past*), and H (*always in the past*), and the binary operator S (*since*). X^n is shorthand for n repeated applications of X. As further syntactic sugar, we define $\mathsf{F}^{\leq n}\varphi \equiv \bigvee_{i=0}^{n} \mathsf{X}^i\, \varphi$ for any $n \in \mathbb{N}$. We will employ a similar shorthand with G (replacing $\vee$ by $\wedge$ in the above). Analogously, by substituting X with Y in the definition $\mathsf{F}^{\leq n}\varphi$ (respectively $\mathsf{G}^{\leq n}\,\varphi$), we get the definition of $\mathsf{P}^{\leq n}\,\varphi$ ($\mathsf{H}^{\leq n}\,\varphi$).

Grossly simplifying matters, the semantics of DTL—formally defined in Appendix A—is similar to LTL and it takes into account that (a) propositions can be true only for specific agents and (b) there is communication (synchronization) between agents.

2.3 Formal Representation of Data Protection Policies

We now show how the kinds of systems introduced in Section 2.1 can be described using DTL, and how privacy policies can be formally expressed. To simplify

notation, we assume discrete time, e.g. one time step per day for the examples given later.

We define two subsets of DTL formulas: *provisional formulas* and *obligational formulas*. Provisional formulas are formulas $@_a[\varphi]$ for an agent a that contain no future-time temporal operators. Obligational formulas are formulas of the same form that do contain future-time temporal operators, but no past-time temporal operators. Further, obligational formulas must not contain temporal operators that are under the scope of a negation, respectively on the left-hand side of an implication.[2]

We also divide DTL formulas into observable formulas and non-observable formulas. These are defined from the reference monitor r's point of view. Intuitively, observable formulas are only concerned with r's local state or actions that r can observe. Actions that r can observe are either local actions of r or communication involving r. More formally, *observable formulas* are of the form $@_r[\varphi]$ or $@_{s_i}[snd(r, msg)]$, for a subject s_i and a message msg. *Non-observable formulas* are all other formulas.

The policies we consider are conjunctions of *decision rules*, which are formulas of the form

$$@_r[authorize(s_i, d_k)] \Rightarrow (p_1 \wedge ... \wedge p_m \wedge o_1 \wedge ... \wedge o_n),$$

where $p_1, ..., p_m$ are observable provisional formulas, $o_1, ..., o_n$ are obligational formulas, and $m, n \geq 0$. The action $authorize(s_i, d_k)$ stands for authorizing subject s_i to access data d_k. Such decision rules may exist for each possible subject/object pair. Of course, access policies are often expressed in a simpler and more compact way by introducing hierarchies or roles, thereby eliminating the necessity of having an entry for each subject/object pair. Without loss of generality, we assume that decision rules of the form presented above may be derived from a more compact representation.

Note that the right-hand side of the implication is only a necessary precondition for the access to be authorized. One reason is that we do not reflect the fact that an access is only authorized after it has been requested by the subject. The acceptance of obligations (explained in Section 2.5) is not reflected in our representation of a decision rule either. Thus, we formulate an implication rather than an equivalence.

2.4 Provisions

A provisional formula that appears in a decision rule is called a *provision*. Intuitively, provisions cover traditional access control requirements. This is because the information stored in r's local state (cf. Section 2.3) can contain attributes about the other subjects and about the data objects, which enables us to express rules in different access control paradigms. Many practical examples can

[2] By restricting this, we cannot express liveness properties in our formalism. This is intentional and it is not a restriction in our context since liveness does not appear to be of practical relevance for formulating privacy requirements. In practice, one generally sets a time limit for carrying out an action.

therefore be found in the access control area, e.g. "s_i has the role *manager*". This could be formalized as $@_r[role(s_i, manager)]$, where $role(s_i, manager)$ is a local state proposition that holds whenever s_i has the role *manager*.

Provisions may also concern actions that must have occurred before the authorization. This is referred to as provisional authorization in the literature [8]. For brevity, we focus only on the results of such provisional authorizations (e.g. that the owner's consent must be present before allowing access), and not on how this result is achieved (e.g. how and when the owner's consent is gathered). Therefore, we assume that all aspects of r's past that are relevant for the current access decision are reflected in r's current local state, and we restrict ourselves to expressing provisions without using past-time temporal operators. This can be done without loss of generality. For example, instead of stating that a certificate must be submitted *before* access is authorized, we demand that the certificate be present *at the time* of the access decision.

As the reference monitor needs to be able to evaluate provisions before authorizing access, we restrict the allowed provisional formulas to observable ones. In other words, we allow only provisions that r is able to check. Addressing non-observable provisions is an area for future work.

2.5 Obligations

Obligations impose conditions on the future (i.e. the time after an access is authorized) that an agent is bound to fulfill. An obligational formula becomes an obligation when a binding is established. Such a binding may be created in many ways, for example by the subject explicitly committing himself to an obligation, or by a law that applies to all agents of a system. For simplicity's sake, we assume that the reference monitor ensures that for all obligational formulas in a decision rule, the bindings are established before the access is authorized and, consequently, we do not explicitly state this in the decision rules. In the remainder of this paper, we hence drop the conceptually important distinction between obligations and obligational formulas, and we will use the term obligation for all obligational formulas in the context of a data protection policy.

We now explain what it means for an obligation to be *violated*. We give an intuitive description here, and refer to Appendix B for a formal definition of this notion. Consider a decision rule of the form $@_r[authorize(s_i, d_k)] \Rightarrow (... \wedge o_l \wedge ...)$, where o_l is an obligation. We assume that each action $authorize(s_i, d_k)$ occurs at most once,[3] which allows us to uniquely relate an obligation to a specific access decision. The obligation o_l is *violated* at a time point t_k if the action $@_r[authorize(s_i, d_k)]$ occurred at a previous time point t_j, and the formula $@_r[authorize(s_i, d_k)] \wedge \neg o_l$ holds at t_j for all possible futures after t_k.

Consider the example $@_r[authorize(s, d)] \Rightarrow @_s[\mathsf{F}^{\leq 3} send(r, m)]$ for a subject s, data item d, and message m. This decision rule states that s must send m to r within three time steps after the authorization. Let t_0 be the time point where $authorize(s, d)$ occurs, and t_l be l time steps later. If $send(r, m)$ does not

[3] This is without loss of generality. From a theoretical perspective, one may safely assume arbitrarily many copies of each rule, each copy uniquely indexed.

occur in the time span between t_0 and t_3, then the obligation $@_s[\mathsf{F}^{\leq 3} send(r,m)]$ is violated at t_4 and at every subsequent time step. If $send(r,m)$ does not occur at times t_0, t_1, or t_2, then the obligation is not violated at any of these time points: $send(r,m)$ could still occur at t_3, and thereby satisfy the decision rule.

We have classified data protection requirements in terms of provisions and obligations. As provisions have been thoroughly analyzed in the literature (see Section 5 for references), we henceforth focus on the part of the landscape that is not yet as well-understood: obligations.

2.6 Classification of Obligations

We classify obligations along two dimensions: temporal structure and distribution. This categorization provides a useful mapping from requirements to enforcement mechanisms, as we will see in Section 3. In the temporal dimension, we have the categories of *bounded future* and *invariance* properties.

We call an until statement $\varphi \,\mathsf{U}\, \psi$ *temporally bounded* if there is an upper time limit for the occurrence of ψ, i.e., there is a constant $n \in \mathbb{N}$ such that $\varphi \,\mathsf{U}\, \psi$ can be rewritten as $\bigvee_{j=1}^{n} \left(\mathsf{X}^j \psi \wedge \bigwedge_{i=0}^{j-1} \mathsf{X}^i \varphi\right)$ without changing the semantics of the formula that contains the until statement. We require that temporally bounded until statements be transformed into the equivalent statement that contains only X operators. With this transformation, we can syntactically distinguish between bounded future (in particular, no U) and invariance properties.[4]

Bounded Future	Only X as the temporal operator
Invariance Properties	At least one G or U, and any number of X operators

In the distribution dimension, we distinguish between observable and non-observable formulas as defined in Section 2.3. This classification results in four categories of obligations named $\mathcal{C}_I$ through $\mathcal{C}_{IV}$.

Time/Distribution	Observable	Non-Observable
Bounded Future	$\mathcal{C}_I$	$\mathcal{C}_{II}$
Invariance Properties	$\mathcal{C}_{III}$	$\mathcal{C}_{IV}$

For each category of obligations, we describe their intuitive nature, present practical examples, and show how some of these examples can be formalized. We continue the convention of calling the reference monitor r and the requester s_i.

Category I—Bounded future, observable: Intuitively, $\mathcal{C}_I$-obligations make statements about properties and events that are observable for r and cover a limited time frame. A classical example comes from the domain of e-commerce: the obligation to pay a fee within a fixed number of days. Data protection examples for $\mathcal{C}_I$-obligations include "Data item d_k may not be accessed for x days" or "r must notify the data owner about the access within x days". The latter could be formalized as $@_r[\mathsf{F}^{\leq x} snd(Owner, released(d_k, s_i))]$, where $Owner$ is the data owner, and $released(d_k, s_i)$ is a message indicating that d_k was given to s_i.

[4] The transformation of temporally bounded until statements into sequences of X is decidable because we are concerned with propositional statements only.

Category II—Bounded Future, Non-observable: This category is similar to $\mathcal{C}_I$, with the difference that we are dealing with non-observable formulas. Privacy-relevant examples include "d_k must be deleted within x days" or "must not be redistributed in the next x days". The former obligation could be formalized as $@_{s_i}[\mathsf{F}^{\leq x}(del(d_k))]$, where the action $del(d_k)$ corresponds to deleting d_k.

Category III—Invariance Properties, Observable: A practical example for a $\mathcal{C}_{III}$-obligation is "Re-access the data at least every x days", which can be formalized as $@_{s_i}[\mathsf{G}(\mathsf{F}^{\leq x} snd(r, request(d_k)))]$, where $request(d_k)$ is a message used to request d_k. This obligation is relevant to data protection since the freshness of data is sometimes demanded by existing data protection regulations.

Category IV—Invariance Properties, Non-observable: $\mathcal{C}_{IV}$-obligations occur often in practice. Examples include "Only for statistical analysis", "Do not distribute further", "Each usage of the data must be reported immediately", or "Must be protected with protection level L until it is declassified by the owner". We formalize the latter obligation as $@_{s_i}[protect(d_k, L)\,\mathsf{U}\, rcv(Owner, declassifiy(d_k))]$. Here $protect(d_k, L)$ reflects that d_k is protected according to protection level L, $Owner$ is the data owner, and $declassifiy(d_k)$ is a message that indicates the declassification of d_k.

3 Coping with the Non-observable

In this section we address the question of how a central reference monitor can ensure that data protection requirements are adhered to. In some sense, this is an impossible task, because it is difficult to imagine how a reference monitor can *enforce* something that it cannot even observe. We use a generalized notion of enforcement here. It covers not only the strict sense of enforcement defined in [14] (the prevention of unwanted executions of a system through system monitoring and denying actions that would violate the policy), but also execution monitoring combined with compensating actions (e.g. penalties) in case the execution violates the policy. More specifically, such a penalty can be applied once an obligation is violated (as defined in Section 2.5 and in the Appendix).

Therefore, a crucial point for achieving enforceability is to be able to monitor the relevant parts of the system. We show how a reference monitor can use non-technical mechanisms, such as audits or legal means, to support the task of making non-observable actions observable, and how the use of such mechanisms can be reflected in the policy of the reference monitor.

As in the previous section, our focus is on obligations. Enforcing provisions is something that has been studied in many variations (see Section 5) and is well understood. We first show how observable obligations can be enforced, and then we show how we can use those enforcement mechanisms also for enforcing non-observable obligations.

3.1 Enforcing Observable Obligations

For observable obligations, suitable enforcement mechanisms have already been described by other authors, e.g. [3]. We give a short overview here, as these

mechanisms will play a role in enforcing the non-observable obligations as well. Among the observable obligations in $\mathcal{C}_I$ and $\mathcal{C}_{III}$, we have three cases, which require different means of enforcement.

First, obligations that are requirements on r's local actions or r's local state are called *r-obligations*. An example is that r must notify the owner of a data item after it has been accessed, or that r must gather evidence about the usage of a data item after some predefined time. Such obligations are of the form $@_r[\varphi]$, and they are enforced by specifying and implementing r such that it always respects them. In other words, we consider r to be a "trusted system". Thus no additional mechanisms are needed.

Second, those obligations that are enforceable by preventing unwanted executions (EM enforceable in the sense of [14]) form a strict subset of the $\mathcal{C}_I$ and $\mathcal{C}_{III}$-obligations. This is, for example, the case for an obligation that prohibits a subject from accessing a certain data object in the future. Here, r can just deny the respective access requests in order to enforce the obligation.

Third, an approach for enforcing all other $\mathcal{C}_I$-obligations is presented in [3]. A reference monitor can check if the obligation is violated at the end of the bounded time period. If this is the case, the reference monitor can penalize the subject, e.g by reducing its service level, lowering some trust or credibility rating of the subject, or taking legal action. This approach can also be applied to $\mathcal{C}_{III}$-obligations. In this case, the obligation must be continuously monitored, not only for a predefined amount of time.

3.2 Enforcing Non-observable Obligations

How can we enforce non-observable obligations? We show how, in some cases, it is possible to reuse existing mechanisms for provisions and observable obligations. Namely, we present three strategies for transforming a non-observable obligation into a set of provisions and observable obligations that specify a similar goal. It is likely that the transformed policy will not specify exactly the goals the original policy expresses. This is the cost of observability. To what extent the original goals are weakened by such a transformation depends on the mechanisms that are used and is explored in the remainder of this section as well as in the next section. The first two strategies we present below aim at making the non-observable parts of an obligation observable, thereby enabling r to monitor their fulfillment. The third strategy limits possible executions to prevent violations of the obligations.

(1) Reserving the right to pull evidence: In this strategy, the reference monitor reserves the right to obtain evidence in the future, for example through an audit. This right could be executed after some deadline (in $\mathcal{C}_{II}$) or whenever r suspects that the obligation has been violated (in $\mathcal{C}_{IV}$). Conceptually, the aim of such an audit is to make certain local events and parts of the local state of the subject visible to the reference monitor. To obtain the above mentioned right, r demands a legal binding (e.g. in the form of a digitally signed statement from the requester) that allows it to trigger an investigation in the future. Demanding a legal binding is now a requirement that can be evaluated in the present, and

therefore a provision. As we illustrate in the next section, this provision is accompanied by a set of r-obligations that define when and under which conditions r must perform an investigation, and how it should react to the result.

(2) Imposing the duty to push evidence: Another strategy is that the requester must commit himself to delivering evidence. The evidence could be delivered by presenting the results of an audit performed by a trusted third party. Delivery of such evidence can occur just once (as a prerequisite for gaining access, which can be expressed by a provision), or on a recurring basis (which constitutes a $\mathcal{C}_{III}$-obligation). As in strategy (1), this requirement needs to be accompanied by a set of r-obligations.

(3) Limiting possible executions: The third possibility for the reference monitor is to limit of possible executions of the agents to those that do not violate the obligation. In order to achieve this, r may put the data into an environment that imposes restrictions on how the data can be used. Basically, this is the idea of digital rights management (DRM) [16,17,18]. It remains to be seen how suitable DRM is for enforcing data protection policies, since any data which is output outside the DRM environment can be re-recorded. At the very least, DRM can act as a support mechanism to the two strategies listed above and thereby increase the likelihood that the obligations are fulfilled, or at least prevent unintended violations resulting from carelessness.

These three strategies are not necessarily applicable to all non-observable obligations. In the case of further distribution, for example, implementing one of the first two strategies might be difficult as r might not be able to audit every subject that could possibly have received the data. The three strategies can at least be applied to some non-observable obligations, however, and the question of which enforcement mechanisms can enforce a given set of requirements is the subject of future work. In the following example, we examine the application of the pull-evidence strategy (1) to enforce $\mathcal{C}_{II}$-obligations.

4 Example

We now show how a simple data protection policy containing provisions, observable obligations, and non-observable obligations can be formalized and enforced. We sketch the semantics of the different requirements and use the pull-evidence strategy to transform the non-observable obligations into observable ones. We also discuss the results of this transformation, and how they relate to the original policy.

Our system consists of two government agencies A and B, a citizen *Owner*, and a trusted auditor T. A acts as the reference monitor and maintains a database of citizen information, where data items d_1 and d_2, belonging to *Owner*, are stored. These data items are sensitive, e.g. medical records, criminal records, or tax statements. B is the only subject in the sense introduced in Section 2, whereas T and *Owner* are additional agents introduced for different purposes.

Recall that our policies contain decision rules for each subject/object pair. To keep things simple, we just present the entries for the pairs (B, d_1) and (B, d_2).

A's policy requires the following for authorizing B to access d_1: **(R1)** B must have role *statistician*; **(R2)** A must notify *Owner* immediately after the access; and **(R3)** B must delete d_1 after at most 90 days. For the pair (B, d_2), the policy requires that **(R4)** B must have trust level *high*; **(R5)** B must notify *Owner* immediately after the access; and **(R6)** B must ensure that its copy of d_2 is never older than 30 days.

R1 and R4 are provisions because the roles and trust levels of the different agents can be reflected in state propositions of A. R2 is a $\mathcal{C}_I$-obligation for A (recall the notion of r-obligations introduced in Section 2). R3 is a $\mathcal{C}_{II}$-obligation because it is non-observable and belongs to the bounded future category. R5 looks similar to R2, but is also a $\mathcal{C}_{II}$-obligation because the action of B notifying *Owner* is not observable to A in our definition. To formalize R6, we must define more precisely what it means for B to ensure that the data is kept fresh. In order to get a new copy of the data, B must request it again, A must grant the request and send the data to B, and B must update its local record of the data with the new version received. B cannot be responsible for the request being granted and the data being sent. Therefore we only demand the following two things in our policy: that B regularly (at least every 30 days) requests d_2, and that B updates its local record every time it receives the new version. This corresponds to a $\mathcal{C}_{III}$-obligation for the regular requests and a $\mathcal{C}_{IV}$-obligation for the updating. Therefore, we will formalize this as two separate requirements in our policy.

4.1 Formalization

We need to define appropriate actions and state propositions for our agents, as well as the messages that are exchanged. We also introduce the actions and state propositions that will be used when transforming the policy later. In these definitions, we use the convention that s_i is an arbitrary subject, d_k is a data item, *obl* is the textual representation of an obligation, and $penalty \in Penalties$, where *Penalties* is a set of applicable penalties in case a requirement is violated. This could be the lowering of a trust or credibility rating, or a legal action.

For the agency A, acting as the reference monitor, we define $authorize(s_i, d_k)$, the action of allowing s_i to access d_k (this includes delivering the data item, i.e. sending d_k to s_i), and $penalize(s_i, penalty)$, the action of penalizing s_i by applying *penalty*. The state propositions include: $role(s_i, statistician)$, for reflecting that s_i is a *statistician*, $trustlvl(s_i, high)$, for reflecting that s_i has trust level *high*, and *investigate*, which reflects the value of a function that outputs *true* or *false* (we will use this later for implementing the pull-evidence approach).

For the agency B, we define $del(d_k)$, the action of deleting d_k (i.e. no copies of d_k's value are retained), $update(d_k)$, the action of updating the previously stored value of d_k, and no state propositions.

The *snd* and *rcv* actions are defined for all agents as explained in Section 2.2. In addition, the following messages may be exchanged, where *procedure* is an element of a set of procedures like *push* or *pull*:

- $request(d_k)$ — message that a subject uses to request d_k from A;
- $released(s_i, d_k)$ — message that A uses to indicate the release of d_k to s_i;

- $accessed(d_k)$ — message that B uses to indicate that it accessed d_k;
- $doc(d_k)$ — document containing data item d_k;
- $accept(obl, procedure, penalty)$ — message that a subject sends to accept that A may use the indicated procedure for checking the obligation, and to accept the penalty in case the obligation is violated;
- $reqAudit(obl)$ — message that A uses to request an audit regarding obl; and
- $viol(obl)$ — message sent as the result of an audit, indicating that the obligation represented by obl is violated.

Now we are in a position to write down the two policy entries for (B, d_1) and (B, d_2). We again assume discrete time with one day per time step. We formalize the policy entries regarding B accessing d_1 and d_2 as

$$@_A[authorize(B, d_1)] \Rightarrow (@_A[role(B, statistician)] \wedge @_A[\mathsf{X}\, snd(Owner, released(d_1, B))] \wedge @_B[\mathsf{F}^{\leq 90} del(d_1)])$$

and

$$@_A[authorize(B, d_2)] \Rightarrow (@_A[trustlvl(B, high)] \wedge @_B[\mathsf{X}\, snd(Owner, accessed(d_2))] \wedge @_B[\mathsf{G}\, \mathsf{F}^{\leq 30} snd(A, request(d_2))] \wedge @_B[\mathsf{G}(rcv(A, d_2) \Rightarrow \mathsf{X}\, update(d_2))]).$$

4.2 Transformation

We have three non-observable obligations in this policy as discussed above: one $\mathcal{C}_{II}$-obligation deriving from R3, one $\mathcal{C}_{II}$-obligation deriving from R5, and one $\mathcal{C}_{IV}$-obligation that defines a part of R6. We now transform the obligation $@_B[\mathsf{F}^{\leq 90} del(d_1)]$ into a set of provisions and observable obligations. We do this in two steps. In the first step, we follow our notion of enforcement introduced in Section 3 and require that if B does not delete d_1 within 90 days, then it will be penalized within a certain time period, say an additional 30 days. This goal can be formalized as

(G1) $@_B[\mathsf{G}^{\leq 90}(\neg del(d_1))] \Rightarrow @_A[\mathsf{X}^{90}\, \mathsf{F}^{\leq 30} penalize(B, pen)]$,

where $pen \in Penalties$.

In the second step, we define how this goal should be achieved using the pull-evidence approach. Roughly, the strategy is as follows. A must obtain B's permission to perform audits that check whether the obligation is violated (recall that this is in addition to accepting the obligation itself, which is done implicitly as stated in Section 2). 90 days after the access, A can ask the auditor to check whether d_1 was really deleted by B. For this to work, we must address the following issues.

First, we must specify under which conditions A requests such an audit. The reason for employing the pull-evidence strategy instead of the push-evidence strategy is that such an audit is not necessary in all cases. This reduces the auditing overhead. Generally, there are two options for triggering audits. One option is that audits are performed randomly as control samples, and the other option is to perform an audit only when suspicion arises. For both options, the state proposition *investigate* indicates whether an audit should be performed.

Second, we assume that the auditor is able to determine if the data was actually deleted, given that B allows the audit to be performed. B must agree to this audit beforehand. Furthermore, we assume that upon request from A, the auditor T always performs an audit and returns the result within a given time period, say one day. This assumption can be formalized as follows (where o_1 is the textual representation of the obligation $@_B[\mathsf{F}^{\leq 90} del(d_1)]$):

(A1) $(@_A[snd(T, reqAudit(o_1))] \wedge \neg @_B[\mathsf{P}^{\leq 90}\, del(d_1)]) \Rightarrow @_T[\mathsf{X}(snd(A, viol(o_1)))]$

Note that this assumption also expresses that the auditor is able to decide a violation at run-time: we know that the audit request will (if ever) be sent 90 days after the access decision, and therefore the obligation is violated at that point if the data has not been deleted within the last 90 days.

Applying the pull-evidence strategy yields an new set of requirements:

(P1) $@_B[snd(A, accept(o_1, pull, pen))]$
(P2) $@_A[\mathsf{X}^{90}\,(investigate \iff snd(T, reqAudit(o_1)))]$
(P3) $@_A[\mathsf{X}^{91}\,(rcv(T, viol(o_1)) \Rightarrow \mathsf{X}\, penalize(B, pen))]$

Requirement P1, which is a provision, expresses the fact that B must accept the procedures and penalties used for enforcing the original obligation. P2 is in $\mathcal{C}_I$and states that, after 90 days, if an audit should be performed then A sends an audit request to T. Requirement P3 is in $\mathcal{C}_{III}$ and defines when a penalty must be applied.

4.3 Comments

G1 is a consequence of A1 and P1–P3 under the semantics of DTL, provided that *investigate* is always true when the obligation is violated. This means that if the audit is always performed, our goal can be achieved. If audits are only performed at random, then this is not always the case and the success of the enforcement strategy depends on whether the possibility of being penalized acts as a deterrent or not. This depends on the agents of a particular system, and is not within the scope of this paper. But the example shows how the application of such a strategy works, and the kinds of assumptions it requires. These assumptions derive from the fact that the result of such a transformation is usually only an approximation of the original policy, as mentioned in Section 3.

We have also shown that to give a formal semantics to data protection requirements, we need to define the actions and state propositions for a concrete system. In this example, we have kept these definitions at a relatively abstract level, but for a practical application this needs to be done in more detail.

The representation of the policy we introduced in Section 2 has its focus on the semantics of the requirements. It does not define when the reference monitor has to authorize an access. There are two reasons for this. First, we only have an implication in a decision rule, not an equivalence (cf. Section 2.3). Second, if obligations are involved, the reference monitor will not be able to decide whether to authorize the access or not, because it does not yet know if the obligations will

be violated. This can only be determined from a global point of view where we know the full, potentially infinite executions. In a concrete realization of such a reference monitor, one would authorize the access exactly when all the provisions are satisfied and the obligations are accepted by the subject.

Finally, note that our formalism is rich enough to allow formulas that do not make sense from a practical point of view. For example, it is possible to express the requirement $@_A[\mathsf{X}\, rcv(B, msg)]$, which is equivalent to $@_B[\mathsf{X}\, snd(A, msg]$. However, it does not make sense from a practical point of view to make A responsible for receiving a message that may never be sent. Addressing this problem is the subject of future work.

5 Related Work

Many extensions and generalizations of access control have been proposed. Jajodia et al. [9] present a framework for combining multiple access control policies within a single system. Temporal criteria may also be used for access decisions whereby access to data is only allowed at certain time points or intervals [2,15] or based on temporal attributes of data, such as the creation date [7]. In our model, these are all provisions, given that the current time is reflected in the reference monitor's state. Policies of the same expressiveness are considered in [10], which introduces the concept of policy automata. Policy automata combine defeasible logic with state machines and represent complex policies as combinations of simpler policies. A variant of access control called provisional authorization is discussed in [8]. Provisional authorization stipulates that access be only authorized if the requester or the system takes certain actions prior to authorizing the request. In our model, this corresponds to provisions that contain action symbols.

The UCON model [12,21] extends access control by introducing decision continuity and attribute mutability. In terms of our model, UCON covers both provisions (including temporal criteria and provisional authorization) and $\mathcal{C}_I$-obligations (provided they do not contain negations). The latter are also discussed in [3], where the authors present a logical framework for monitoring these obligations, and for taking compensating actions when obligations are violated. EPAL [1] is a description language for privacy policies (data protection policies in our terminology) that can express all categories presented in this paper, but does not specify a semantics for obligations. An XML-based syntax for describing privacy statements for web sites is defined in the P3P standard [19].

A collection of obligations encountered in practice is given in [11]. The authors have a more operational view of obligations, and differentiate between short-term obligations, long-term obligations and ongoing obligations. Formal definitions of the corresponding predicates are not given, but practical examples are provided for each category.

The terms *obligation* and *provision* are often but not consistently used in the literature to describe different types of data protection requirements. Some authors (e.g. [12]) use obligations to refer to actions that principals must perform, or must have performed *before* an access is authorized. For other authors (e.g. [3]), obligations concern actions that must be performed in the future, *after* the

access is authorized. We adopt and generalize the latter definition: obligations are agreed-upon conditions that not only concern actions that are to be performed in the future, but also more general propositions about the future, for example the mandatory presence of certain protection mechanisms. From [3], we also adopt and generalize the notion of provisions. In our model, provisions are conditions that must hold before an access is authorized, whereas in [3], they are specific actions that must be taken before an access is authorized.

6 Conclusions and Future Work

In this paper, we showed how different aspects of data protection can be handled by an extension of access control models. This extended model allows us to precisely specify a broad range of data protection requirements—provisions and obligations—that take into account observable and non-observable elements. We showed how existing approaches to enforcing non-observable obligations fit into our conceptual and formal framework.

We are aware that our syntactic classification imposes a restriction on the presentation of requirements. For example, a provisional formula may have a semantic equivalent that does not fit the syntactic criterion for a provisional formula. This restriction does not seem too strong when the criterion is used a-priori rather than a-posteriori, e.g. by describing policies in a dedicated policy editor that simply forbids certain constructs. Whether or not our syntactic separation into provisions and obligations leaves out some important semantic constructs remains to be investigated.

One promising direction for future work is to use the framework as a basis for a practically useful policy language that caters for both provisions and obligations. It is likely that "policy patterns" in such a language can address many recurring data protection needs. Since our framework language (DTL) has a formal semantics, it is amenable to formal reasoning about policies and their composition (interesting aspects include consistency and subsumption). Another direction to explore, by case studies, is the practical applicability of the mechanisms presented in this paper. This is important in identifying the boundary between those requirements that can be technically enforced and those that require non-technical mechanisms, such as legal ones. It will also shed light on how much non-observable requirements must be weakened when transforming them into observable ones. Our long-term goal here is to design a server-side reference monitor controlling access to sensitive personal data. Finally, the use of DRM mechanisms for handling obligations, as mentioned in Section 3, also requires further investigation. In particular, we currently do not know how much can be achieved by using client-side reference monitors. A related question concerns honest subjects, where we need only to prevent careless, but not malicious, behavior. In this case, the mechanisms are likely to be weaker than in the case of potentially malicious subjects.

Acknowledgments. P. Hankes Drielsma, F. Klaedtke, P. E. Sevinç, C. Sprenger and L. Viganò provided useful comments on earlier versions of the paper.

References

1. M. Backes, B. Pfitzmann, and M. Schunter. A toolkit for managing enterprise privacy policies. In *Proc. ESORICS'03*, Springer LNCS 2808, pages 162–180. 2003.
2. E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. An access control model supporting periodicity constraints and temporal reasoning. *ACM Transactions on Database Systems*, 23(3):231–285, 1998.
3. C. Bettini, S. Jajodia, X. S. Wang, and D. Wijesekera. Provisions and obligations in policy rule management. *J. Network and System Mgmt.*, 11(3):351–372, 2003.
4. C. Caleiro, L. Viganò, and D. Basin. Metareasoning about security protocols using distributed temporal logic. In *Proc. IJCAR'04 Workshop on Automated Reasoning for Security Protocol Analysis (ARSPA'04)*. ENTCS 125(1), 2005.
5. H.-D. Ehrich and C. Caleiro. Specifying communication in distributed information systems. *Acta Informatica*, 36:591–616, 2000.
6. H.-D. Ehrich, C. Caleiro, A. Sernadas, and G. Denker. Logics for specifying concurrent information systems. In *Logic for Databases and Information Systems*, pages 167–198. Kluwer Academic Publishers, 1998.
7. A. Gal and V. Atluri. An authorization model for temporal data. In *Proc. 7th ACM Conference on Computer Communications Security*, pages 144–153. ACM Press, 2000.
8. S. Jajodia, M. Kudo, and V. Subrahmanian. Provisional authorizations. In A. Gosh, editor, *E-Commerce Security and Privacy*, pages 133–159. Kluwer, 2001.
9. S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems*, 26(2):214–260, 2001.
10. M. McDougall, R. Alur, and C. A. Gunter. A model-based approach to integrating security policies for embedded devices. In *Proc. 4th ACM international conference on Embedded software*, pages 211–219. ACM Press, 2004.
11. M. C. Mont. Dealing with privacy obligations in enterprises. Technical report, HP Laboratories Bristol, Jun 2004.
12. J. Park and R. Sandhu. The UCON ABC Usage Control Model. *ACM Transactions on Information and Systems Security*, 7:128–174, 2004.
13. A. Pnueli. The temporal semantics of concurrent programs. In *Proc. Intl. Symp. on Semantics of Concurrent Computation*, pages 1–20. Springer-Verlag, 1979.
14. F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.
15. F. Siewe, A. Cau, and H. Zedan. A compositional framework for access control policies enforcement. In *Proc. 2003 ACM workshop on Formal methods in security engineering*, pages 32–42. ACM Press, 2003.
16. S. W. Smith. *Trusted Computing*. Springer-Verlag, 2005.
17. P. van Oorschot. Revisiting software protection. *Information Security, LNCS*, 2851:1–13, 2003.
18. P. van Oorschot. Software protection and application security: understanding the battleground. In *State of the art and evolution of computer security and industrial cryptography*, 2003.
19. W3C. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification, April 2002. Available at `http://www.w3.org/TR/P3P/`.
20. G. Winskel. Event structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, Springer LNCS 255, pages 325–392. 1987.
21. X. Zhang, J. Park, F. Parisi-Presicce, and R. Sandhu. A logical specification for usage control. In *Proc. 9th ACM symp. on Access control models and technologies*, pages 1–10. ACM Press, 2004.

A Distributed Temporal Logic

In this appendix, which is adopted from [4], we explain the basics of DTL.

The syntax of DTL is defined over a *distributed signature*

$$\Sigma = \langle Id, \{Act_i\}_{i \in Id}, \{Prop_i\}_{i \in Id} \rangle$$

of a system, where Id is a finite set of *agent identifiers* and, for each $i \in Id$, Act_i is a set of *local action symbols* and $Prop_i$ is a set of *local state propositions*. The *global language* $\mathcal{L}$ is defined by the grammar $\mathcal{L} ::= @_i[\mathcal{L}_i] \mid \bot \mid \mathcal{L} \Rightarrow \mathcal{L}$, for $i \in Id$, where the *local languages* $\mathcal{L}_i$ are defined by

$$\mathcal{L}_i ::= Act_i \mid Prop_i \mid \bot \mid \mathcal{L}_i \Rightarrow \mathcal{L}_i \mid \mathcal{L}_i \,\mathsf{U}\, \mathcal{L}_i \mid \mathcal{L}_i \,\mathsf{S}\, \mathcal{L}_i \mid j{:}\mathcal{L}_j$$

with $j \in Id$. Locally for an agent, U and S are respectively the *weak until*[5] and *since* temporal operators. Actions correspond to true statements about an agent when they have just occurred, whereas state propositions characterize the current local states of the agents. Note that the global formula $@_i[\varphi]$ means that φ holds at the current local state of agent i. A local formula $j{:}\varphi$ appearing inside a formula in $\mathcal{L}_i$ is called a *communication formula* and it means that agent i has just communicated with agent j for whom φ held. The interpretation structures of $\mathcal{L}$ are suitably labelled distributed life-cycles, built upon a simplified form of Winskel's *event structures* [20]. A *local life-cycle* of an agent $i \in Id$ is a pair $\lambda_i = \langle Ev_i, \rightarrow_i \rangle$, where Ev_i is the set of *local events* and $\rightarrow_i \subseteq Ev_i \times Ev_i$ is the *local successor relation*, such that the transitive closure $\rightarrow_i^*$ defines a well-founded total order on Ev_i, called *local causality*. A *distributed life-cycle* is a family $\lambda = \{\lambda_i\}_{i \in Id}$ of local life-cycles such that the transitive closure $\rightarrow^*$ of $\rightarrow = \bigcup_{i \in Id} \rightarrow_i$ defines a partial order on the set $Ev = \bigcup_{i \in Id} Ev_i$ of all events, called *global causality*. This last condition is essential since events can be shared by several agents at communication points.

We can check the progress of an agent by collecting all the local events that have occurred up to a certain point. This yields the notion of the *local configuration* of an agent i: a finite set $\xi_i \subseteq Ev_i$ closed under local causality, i.e. if $e \rightarrow_i^* e'$ and $e' \in \xi_i$ then also $e \in \xi_i$. The set Ξ_i of all local configurations of an agent i is clearly totally ordered by inclusion and has $\emptyset$ as the minimal element. In general, each non-empty local configuration ξ_i is reached, by the occurrence of an event that we call $last(\xi_i)$, from the local configuration $\xi_i \setminus \{last(\xi_i)\}$. We can also define the notion of a *global configuration*: a finite set $\xi \subseteq Ev$ closed for global causality, i.e. if $e \rightarrow^* e'$ and $e' \in \xi$ then also $e \in \xi$. The set Ξ of all global configurations constitutes a lattice, under inclusion, and has $\emptyset$ as the minimal element. Clearly, every global configuration ξ includes the local configuration $\xi|_i = \xi \cap Ev_i$ of each agent i. Given $e \in Ev$, note that $e{\downarrow} = \{e' \in Ev \mid e' \rightarrow^* e\}$ is always a global configuration.

An *interpretation structure* $\mu = \langle \lambda, \alpha, \pi \rangle$ consists of a distributed life-cycle λ plus families $\alpha = \{\alpha_i\}_{i \in Id}$ and $\pi = \{\pi_i\}_{i \in Id}$ of local labelling functions. For

[5] In contrast to the strong until operator, the weak until operator does not require γ to eventually happen in the formula $\varphi \,\mathsf{U}\, \gamma$.

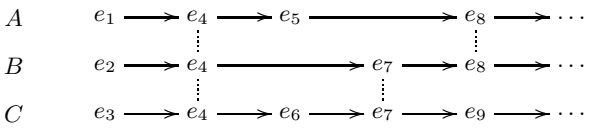

Fig. 1. A distributed life-cycle for agents A, B and C

$$\pi_A(\emptyset) \xrightarrow{\alpha_A(e_1)} \pi_A(\{e_1\}) \xrightarrow{\alpha_A(e_4)} \pi_A(\{e_1, e_4\}) \xrightarrow{\alpha_A(e_5)} \pi_A(\{e_1, e_4, e_5\}) \xrightarrow{\alpha_A(e_8)} \cdots$$

Fig. 2. The progress of agent A

each $i \in Id$, $\alpha_i : Ev_i \to Act_i$ associates a local action to each local event, and $\pi_i : \Xi_i \to \wp(Prop_i)$ associates a set of local state propositions to each local configuration.

Fig. 1 illustrates the notion of a distributed life-cycle, where each row comprises the local life-cycle of one agent. In particular, $Ev_A = \{e_1, e_4, e_5, e_8, \dots\}$ and $\to_A$ corresponds to the arrows in A's row. We can think of the occurrence of the event e_1 as leading agent A from its initial configuration $\emptyset$ to the configuration $\{e_1\}$, and then of the occurrence of the event e_4 as leading to configuration $\{e_1, e_4\}$, and so on; the state-transition sequence of agent A is displayed in Fig. 2. Shared events at communication points are highlighted by the dotted vertical lines. Note that the numbers annotating the events are there only for convenience since no global total order on events is in general imposed. Fig. 3 shows the corresponding lattice of global configurations.

We can then define the *global satisfaction relation* at a global configuration ξ of μ, where Ξ_i is the set of all local configurations of agent i in μ, as

- $\mu, \xi \Vdash @_i[\varphi]$ if $\mu, \xi|_i \Vdash_i \varphi$;
- $\mu, \xi \nVdash \bot$;
- $\mu, \xi \Vdash \gamma \Rightarrow \delta$ if $\mu, \xi \nVdash \gamma$ or $\mu, \xi \Vdash \delta$,

where the *local satisfaction relations* at local configurations are defined by

- $\mu, \xi_i \Vdash_i act$ if $\xi_i \neq \emptyset$ and $\alpha_i(last(\xi_i)) = act$;
- $\mu, \xi_i \Vdash_i p$ if $p \in \pi_i(\xi_i)$;
- $\mu, \xi_i \nVdash_i \bot$;
- $\mu, \xi_i \Vdash_i \varphi \Rightarrow \psi$ if $\mu, \xi_i \nVdash_i \varphi$ or $\mu, \xi_i \Vdash_i \psi$;
- $\mu, \xi_i \Vdash_i \varphi \,\mathsf{U}\, \psi$ if the following holds: if there exists $\xi_i'' \in \Xi_i$ with $\xi_i \subsetneq \xi_i''$ such that $\mu, \xi_i'' \Vdash_i \psi$, then $\mu, \xi_i' \Vdash_i \varphi$ for every $\xi_i' \in \Xi_i$ with $\xi_i \subsetneq \xi_i' \subsetneq \xi_i''$; otherwise, $\mu, \xi_i' \Vdash_i \varphi$ for every $\xi_i' \in \Xi_i$ with $\xi_i \subsetneq \xi_i'$;
- $\mu, \xi_i \Vdash_i \varphi \,\mathsf{S}\, \psi$ if there exists $\xi_i'' \in \Xi_i$ with $\xi_i'' \subsetneq \xi_i$ such that $\mu, \xi_i'' \Vdash_i \psi$, and $\mu, \xi_i' \Vdash_i \varphi$ for every $\xi_i' \in \Xi_i$ with $\xi_i'' \subsetneq \xi_i' \subsetneq \xi_i$;
- $\mu, \xi_i \Vdash_i j{:}\varphi$ if $\xi_i \neq \emptyset$, $last(\xi_i) \in Ev_j$ and $\mu, (last(\xi_i)\downarrow)|_j \Vdash_j \varphi$.

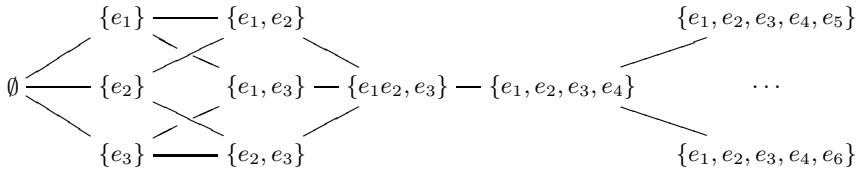

Fig. 3. The lattice of global configurations

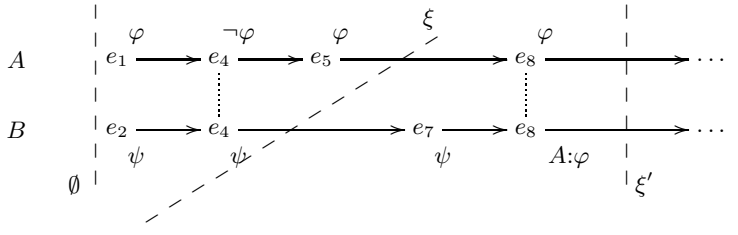

Fig. 4. Satisfaction of formulas

We say that μ is a model of $\Gamma \subseteq \mathcal{L}$ if $\mu, \xi \Vdash \gamma$ for every global configuration ξ of μ and every $\gamma \in \Gamma$. Fig. 4 illustrates the satisfaction relation with respect to communication formulas of our running example. Clearly $\mu, \emptyset \Vdash @_B[\psi \,\mathsf{U}\, A{:}\varphi]$, because $\mu, \xi' \Vdash @_B[A{:}\varphi]$. Note however that $\mu, \xi \not\Vdash @_B[A{:}\varphi]$, although $\mu, \xi \Vdash @_A[\varphi]$.

Other usual logical operators are defined as abbreviations, e.g. $\neg$, $\top$, $\vee$, and $\wedge$. We also define the following temporal operators:

$\mathsf{X}\,\varphi \equiv \bot \,\mathsf{U}\, \varphi$ (weak next)
$\mathsf{Y}\,\varphi \equiv \bot \,\mathsf{S}\, \varphi$ (previous)
$\mathsf{P}\,\varphi \equiv \top \,\mathsf{S}\, \varphi$ (sometime in the past)

$\mathsf{G}\,\varphi \equiv \varphi \,\mathsf{U}\, \bot$ (always in the future)
$\mathsf{H}\,\varphi \equiv \neg\, \mathsf{P}\, \neg\, \varphi$ (always in the past)

Let *Msg* be a (not necessarily finite) set of messages. For each agent $a \in Id$, the set of actions Act_a includes $snd(b, m)$ (send message m to agent b) and $rcv(b, m)$ (receive message m from agent b), where $b \in Id$ is another agent, and $m \in Msg$. Now we introduce the following axiom.

$$\forall\, a, b \in Id, m \in Msg : @_a[snd(b, m) \iff b{:}rcv(a, m)]$$

This axiom defines a reliable and synchronous communication channel for each pair of agents. In this paper, we do not make other use of the $j{:}\varphi$ operator, i.e. the *local languages* $\mathcal{L}_i$ are only defined by

$$\mathcal{L}_i ::= Act_i \mid Prop_i \mid \bot \mid \mathcal{L}_i \Rightarrow \mathcal{L}_i \mid \mathcal{L}_i \,\mathsf{U}\, \mathcal{L}_i \mid \mathcal{L}_i \,\mathsf{S}\, \mathcal{L}_i,$$

in the body of this paper. The reason is that we do not need this operator for our policy language, but only to define the semantics *snd* and *rcv*. Moreover, omitting this simplifies the syntactical classification between observable formulas and non-observable formulas that we introduce in Section 2.

B Violation of Obligations

Let $\mu = \langle \lambda, \alpha, \pi \rangle$ be an interpretation structure with $\lambda = \{\langle Ev_i, \rightarrow_i \rangle\}_{i \in Id}$ and $\xi = \bigcup_{i \in Id} Ev_i$ a global configuration of μ. We consider a formula of the form $@_j[a] \Rightarrow \varphi$, where a is a local action symbol that is only defined for agent j and has exactly one pre-image under α_j, and $\varphi \in \mathcal{L}_i$ for some $i \in Id$. The subformula φ is *violated* with respect to μ iff there exists $\xi' \in \Xi$ with $\xi' \subseteq \xi$ such that $\mu, \xi' \Vdash a \wedge \neg\varphi$.

The reason why a must only be related to one single event is that otherwise, there could be $\xi'', \xi''' \in \Xi$ with $\xi'' \subseteq \xi; \xi''' \subseteq \xi; \mu, \xi'' \Vdash a \wedge \neg\varphi$ and $\mu, \xi''' \nVdash a \wedge \neg\varphi$. In our context of obligations and authorization actions, this would mean that the actions that follow two different authorizations could interfere.

Note that this formal definition of violation cannot be used by a reference monitor *at runtime*. This is because our characterization is with respect to a *fixed* interpretation structure of the logical formulas that make up our models of a system. Because this interpretation is fixed, and, in this sense, encompasses everything that has happened, it does not leave any room for decisions of the agents. Without formalizing it here, we hence assume an "operational" definition that allows the auditor to decide violation at runtime.