Data & Knowledge Engineering 71 (2012) 127–147

Contents lists available at SciVerse ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Formal enforcement and management of obligation policies

Yehia Elrakaiby [a,*], Frédéric Cuppens [b], Nora Cuppens-Boulahia [b]

[a] University of Luxembourg, 6 rue Coudenhove-Kalegri, L-1359, Luxembourg
[b] Télécom Bretagne, 2 rue de la Chataigneriae, 35512 Cesson Sévigné, France

ARTICLE INFO

Article history:
Received 24 August 2010
Received in revised form 5 September 2011
Accepted 6 September 2011
Available online 16 September 2011

Keywords:
Security
Obligations
Obligation policies
Obligation monitoring
Policy management
Policy languages
Conceptual modeling

ABSTRACT

Obligations are generally *actions* that users are required to take and are essential for the expression of a large number of requirements. For instance, obligation actions may represent prerequisites to gain some privilege (pre obligations), to satisfy some ongoing or post requirement for resource usage (ongoing and post obligations), or to adhere to some privacy or availability policy. Obligations may also define *states of affairs* which should be maintained. An example of such obligations is the obligation "doctors should remain alert while in the operating room". In this paper, we introduce a formal framework for the management and enforcement of obligation policies. The framework is formalized using concepts from action specification languages and the Event Condition Action paradigm of active databases. Therefore, our framework allows reasoning about change in the state of obligations and, at the same time, provides declarative formal semantics for their enforcement. In this framework, we support many types of obligations and show how to manage obligation activation, fulfillment and violation.

1. Introduction

Traditionally, security policies have mainly focused on the specification and management of access control requirements expressed in the form of permissions and prohibitions. However, the application of security policies in different domains has shown the relevance of obligations to the expression of many requirements such as usage control and data protection [1,2], privacy [3,4], and availability [5].

Until recently, obligation requirements were hard-coded in applications. This leads to applications which are inflexible because it becomes difficult to update the application behavior when requirements change. For this reason, several obligation policy languages and frameworks have been introduced [6,2,7–11]. The use of a policy-based approach allows the expression of requirements using a high-level language, which simplifies policy interpretation and representation. It also enables administrators to dynamically update the system behavior whenever there is a change in the requirements by modifying rules specified in the policy. Moreover, the formalization of the policy language allows the formal analysis of policy and the proof of its properties. This reduces the risk of policy misspecification.

Obligation policy languages can be broadly classified into two categories. On one hand, there are *policy enforcement languages* such as [9]. These languages generally simplify policy specification and interpretation. However, they lack the formal semantics needed to allow formal policy analysis and the proof of policy properties. On the other hand, there are *policy analysis languages* such as [12,13]. These languages allow formal policy analysis and the expression of a large variety of obligations. However, these languages do not provide the operational semantics needed to dynamically enforce and manage obligations in a policy-managed system. In this paper, we introduce a policy language which aims at combining the advantages of both policy enforcement and analysis languages. We make a more thorough and detailed overview of existing policy languages in Section 12.

* Corresponding author at: Interdisciplinary Centre for Security Reliability and Trust (SnT), University of Luxembourg, 6 rue Coudenhove-Kalegri, L-1359, Luxembourg. Tel.: +352 4666445872; fax: +352 4666445669.
E-mail address: yehia.elrakaiby@uni.lu (Y. Elrakaiby).

0169-023X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2011.09.001

There are two main types of obligations, namely system and user obligations. System obligations denote actions which should be immediately taken when some conditions become true. An example of a system obligation is "when an attack is detected on some server, the source address of the attacker should be blocked". System obligations are generally enforced by mechanisms implemented in the system. On the other hand, user obligations are actions that subjects are required to take in the future. Since subjects cannot be forced to take actions, user obligations are *unenforceable* and should be monitored for violation/fulfillment. A user obligation also has a deadline that specifies when it is violated. In particular, an obligation is violated when its deadline is detected if the obligation has not been fulfilled. In the following, we mainly focus on user-obligations and the term obligation is used to refer to user obligations. One may however consider that a system obligation is a form of a user obligation where the obligation subject is the system and where the obligation deadline is detected if the obligation is not immediately enforced after its activation.

In order to formalize an obligation model, we analyze the basic elements of an obligation. An obligation is associated with an event after which the obligation becomes effective. This event may be a temporal event such as the event "at the first of every month" or an action-based event such as the event "when a file is downloaded". Obligations should also be associated with violation events (like deadlines). Without violation events, obligations are void [14], i.e. without any force. Events characterizing obligation violation may be action-based or temporal events similarly to obligation activation. They may also correspond to relative temporal deadlines. Relative temporal deadlines enable the specification that an obligation is violated if it is not fulfilled within some determined period of time after its activation. For instance, an obligation may specify that doctors should examine new patients assigned to them within 6 h.

When obligations are violated, it is often necessary to activate sanction/reaction policies. Sanction/reaction policies specify new prohibitions/obligations which should be activated to compensate the violated obligations. For instance, when the previous obligation is violated, a sanction policy may require that the doctor of the violated obligation submits a report to his/her department head. A reaction policy may, on the other hand, specify an obligation which requires that another doctor examines the unexamined patient. Finally, some obligations may persist after they are violated. For instance, an obligation to pay his/her taxes after some deadline may remain required even if it is violated. On the other hand, other obligations such as the obligation to submit paper reviews before the conference notification deadline may become unnecessary after their violation.

Obligations may also require that some state of affairs be maintained as opposed to taking some action. For instance, consider the obligation "doctors should remain alert while in the operating room". This obligation may imply, for instance, that no signs of drowsiness/fatigue should appear on doctors in the operating room. We distinguish these obligations from regular obligations requiring action by calling the latter fulfillment obligations and the former continuous obligations. We consider continuous obligations different from negative obligations (which are logically equivalent to prohibitions) since it does not seem intuitive to view the requirement above as a prohibition to show signs of fatigue. In other words, there does not seem to exist a physical action which a doctor may be prohibited to maintain the desired state of affairs. At the same time, one motivation behind including continuous obligations in the policy is to enable administrators to specify the necessary sanction/reaction policies which should apply when these requirements are violated. For instance, one may imagine a sanction in the form of a prohibition for the doctor to perform operations and a reaction policy requiring another doctor to replace the sanctioned doctor.

As illustrated in the previous examples, an obligation cannot be fulfilled/violated unless it was activated in a previous state. This ordering between the obligation activation and its violation/fulfillment implies that obligation models should consider different states for obligations. Therefore, we introduce in this paper a state-based obligation model. The model identifies the states which obligations may assume and defines when state transitions occur. The model is formalized using concepts from action specification languages [15]. Consequently, the model enables the reasoning about the evolution of obligations when change in the system state is detected. Our formalization language also integrates the Event Condition Action (ECA) rules [16] largely studied in the domain of active databases. ECA rules model reactive application behavior. Therefore, they enable the clarification of policy-enforcement and the study of several important related issues such as decidability and termination.

In this paper, we study and formalize the management of contextual obligation policies. We summarize the main contributions of this paper as follows:

– The introduction of an obligation language that is at the same time expressive to enable the specification of large number of practical real life requirements and simple to allow non-specialists to understand it and use it. Violation events in the language are not limited to simple temporal deadlines and event-based deadlines are supported.
– The association of the language with an obligation state-based model. The model clarifies the semantics of obligations by identifying the different obligation states and state transitions. The model also recognizes that some obligations remain required (persist) after their violation. Therefore, it enables the specification of many real-life obligation requirements.
– The formalization of our obligation model using the concepts of action specification languages and the Event Condition Action paradigm of active databases. Therefore, our framework enables the formal reasoning about the evolution of the state of obligations and provides operational semantics for their enforcement.
– The support of sanction and recompense policies necessary to compensate violated obligations and/or to encourage subjects to fulfill their obligations.
– The introduction and formalization of continuous obligations. Continuous obligations are generally unsupported in other obligation frameworks. Our obligation language also distinguishes between persistent and non-persistent obligations. These two points arguably add to the expressiveness of our obligation language in comparison with other obligation languages.

This paper is organized as follows. Section 2 presents some obligation examples which we discuss and analyze. Section 3 presents state description. Section 4 presents our context and obligation policy languages. In Section 5, we introduce our obligation