Faculdade de Engenharia da Universidade do Porto

Universidade do Porto

Faculdade de Engenharia

FEUP

Physical Access Control System

Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Engenharia de Software

Bruno Moreira

Márcio Fontes

November, 2015

Abstract

On this report we present a formal, tool-supported approach to the design and maintenance of access control policies expressed in the eXtensible Access Control Markup Language (XACML). Our aim is to develop an application using the model-oriented specification language from Vienna Development Method (VDM++), capable of perform actions based on targets, subjects and subjacent policies, and therefore apply the specified policy combination algorithms to determine its outcome status (e.g., denial, permit, etc.).

Content

1. Introduction

1.1 Project Description

This project aims to develop a physical access control system using XACML [1] language, implemented in VDM++, in order to perform authorization, identification, authentication, access approval and keep records of all succeeded or failed access requests.

1.2 Objectives

The physical access control system should fulfill the objectives given by Table 1. These objectives are the ones which are enumerated on the assessment and, therefore, no further detail is supplied.

Table 1 - Objectives

ID	OBJECTIVE DESCRIPTION
O1	May be used in all sorts of physical facilities, such as hotels, schools, banks, military facilities, etc.
O2	Should be able to control the access to buildings, sectors (inside a building), rooms, parking lots, floors (in elevators), and other facilities.
O3	Each authorized user is given a contactless card to present at appropriate access points, communicating with NFC (near field communication) or other means.
O4	Access cards may be temporary, with a defined date-time of expiration (e.g., for hotel guests).
O5	Each access card has a unique identifier and access cards may be reused.
O6	Both users and facilities may be organized into groups (e.g., students, teachers, classrooms, computer laboratories, etc.) to facilitate the definition of access rules.
O7	A user or facility may belong to multiple groups.
O8	Access policies are defined by means of access rules.
O9	Each access rule specifies a user or group of users, a facility or group of facilities, and possibly a temporal constraint (a specific date-time interval, a recurrent time interval, etc.).
O10	Rules may be defined as exceptions to other rules (e.g., to deny access for some period of time).

[1] XACML – eXtensible Access Control Markup Language

O11	The system should be able to decide on access requests.
O12	The system should keep a log of all succeeded or failed access requests.

1.3 Requirements

This project was implemented based on the requirements described by Table 2. The list of requirements was formulated taking into consideration the project's delivery date and its corresponding scope. Furthermore, this list was made short to avoid enumerating a vast number of user stories, due to the project's complexity.

Table 2 - Requirements

ID	DESCRIPTION
R1	Provide a method for combining individual **rules** and **policies** into a single **policy set** that applies to a particular decision **request.**
R2	Provide a method for rapidly identifying the **policy** that applies to a given **action**, based upon the values of **attributes** of the **subjects**, **resource** and **action.**
R3	Provide a method for basing an **authorization decision** on the contents of an information **resource.**
R4	Provide a method for flexible definition of the procedure by which **rules** and **policies** are combined.
R5	Provide a method for specifying a set of **actions** that must be performed in conjunction with **policy** enforcement.

1.4 Optional Requirements

The optional requirements are described by Table 3. We consider optional requirements as features which would be implemented if there was enough time after fulfilling the high-priority requirements.

Table 3 - Optional Requirements

ID	DESCRIPTION
OR1	Provide a method for dealing with **subjects** acting in different capacities;
OR2	Provide a method for dealing with multi-valued **attributes**;

OR3	Provide a method for handling a distributed set of policy components, while abstracting the method for locating, retrieving and authenticating the policy components.
OR4	Provide an abstraction layer that insulates the policy-writer from the details of the application environment.

2. UML Modeling

On this section it's presented the use cases and conceptual model for this project, as well as additional notes and constraints concerning the diagrams.

2.1 Use Case Diagram

2.2 Class Diagram

Conceptual modelling is the abstraction of a simulation model from the part of the real world it is representing - "the real system" (Robinson, 2008). After collecting the necessary requirements, we achieved the following conceptual model, represented by (Figure 1):

3. VDM++ Modeling

3.1 Classes

This VDM++ application is consisted by the classes described in Table 4 in order to fulfill its purposes. These classes are represented on the UML Class Diagram in the previous section.

Table 4 - Classes

CLASS	DESCRIPTION
ACCESS	This class is meant to save the content about a certain target, action and the corresponding effect, which can be `<Permit>`, `<Deny>`, `<Indeterminate>` or `<notApplicable>`.
ACTION	This class is meant to save the content about the type of action, which can be `<Assign>`, `<View>` or `<Receive>`.
CARD	This class is meant to save the content about the identification card, and its corresponding expiration date if it exists.
DATE	This class is meant to describe a date (year-month-day).
FACILITY	This class is meant to save the content about a facility, i.e, the name, its corresponding type (`<Hotel>`, `<School>` or `<Bank>`) and the log of accesses to the building.
PAP[2]	This class is meant to have the application's set of policies and make them available to the PDP.
PDP[3]	This class is meant to evaluate the application policy and render an authorization decision, applying the corresponding combining algorithms.
PEP[4]	This class is meant to perform the access control, by making decision requests and enforcing authorization decisions.
POLICY	This class is meant to save the content about a set of rules, the rule-combining algorithm to be applied (which can be `<permitOverrides>` or `<denyOverrides>`), and the corresponding target.
REQUEST	This class is meant to save a request status which can be `<Active>`, `<Pending>` or `<Finished>`.
RESOURCE	This class is meant to save the content about a data, service or system component.
RULE	This class is meant to save the content about a target, an effect, facility group and user group, and eventually a temporal constraint.

[2] PAP – Policy Administration Point
[3] PDP – Policy Decision Point
[4] PEP – Policy Enforcement Point

SUBJECT	This class is meant to save the content about a person trying to access a building resource.
TARGET	This class is meant to save the content about a set of subjects, set of resources and set of actions to be taken.

3.2 Data Types

In order to develop this VDM++ application and to completed the described classes in the previous section, we used the data types given by Table 5.

Table 5 - Data Types

DATA TYPE	VALUE
COMBALG	`<denyOverrides>` or `<permitOverrides>`
EFFECT	`<Permit>` or `<Deny>` or `<Indeterminate>` or `<notApplicable>`
IDENTIFIER	Natural number
STATUS	`<Active>` or `<Pending>` or `<Finished>`
STRING	Sequence of chars
TYPE	`<Assign>` or `<View>` or `<Receive>`

3.3 Domains

4. Model Validation

4.1 Test Classes

4.2 Test Results

4.3 Requirements Traceability

5. Model Verification

5.1 Domain Verification

5.2 Invariant Verification

6. Code Generation

7. Conclusions

7.1 Results Achieved

7.2 Difficulties

We honestly think that there should be more emphasis on explaining how using VDM++ may benefit the way programmers develop applications, using imperative languages like Java. There's been a certain difficulty at the beginning to actually know what to do and where to start, and we lost tons of time on that dilemma. Furthermore, the massive amount of information about XACML was quite misleading at the beginning since we had no idea if we should implement XACML in its total completeness, as there wouldn't be enough time to develop a system with that kind of scope. This led to delays on the development and therefore the application's quality was far from we expected.

7.3 Improvements

After developing this (quite) simple physical access control system using XACML, there are some features which we would like to implement, and therefore take use of all VDM++ potential capabilities. The first enhancement would be to translate the user request into a XACML type-request in order to follow the control flow in XACML. The second enhancement would be to read a XML file containing the policies, already in XACML, and populate the set of policies. The last enhancement would be to export a file with all the requests, taken actions and combining algorithms used.

7.4 Effort

The distribution of effort (%) by each group member is given as follows:

- Bruno Moreira –
- Márcio Fontes –

References

Bryans, J. W., & Fitzgerald, J. S. Formal Engineering of XACML Access Control Policies in VDM++. Newcastle University, School of Computer Science. Newcastle: Newcastle University.

OASIS. (2013 de january de 2013). eXtensible Access Control Markup Language (XACML) Version 3.0. Acessed on December 2nd, 2015, available at OASIS Docs: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

Robinson, S. (2008). Conceptual Modelling for Simulation Part I: Definition and Requirements. Journal of the Operational Research Society.