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

December, 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

Table of Tables

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

We were unable to import the generated .uml file to Modelio and therefore we cannot paste here the use case diagram. Please check the .uml file for more details.

2.2 Class Diagram

For the same reason, we were unable to import the generated .uml file to Modelio and therefore we cannot paste here the use case diagram.

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.

Note: the classes implementations are presented on the Annexes to avoid confusion.

3.2 Data Types

In order to develop this VDM++ application and to complement 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>`

4. Model Validation

4.1 Test Classes

In order to validate the application's robustness and corresponding features, some tests were developed as described in Table 6.

Table 6 - Test Classes

CLASS	DESCRIPTION
POLICIES TEST	Asserts if the decisions of a certain policy produce the expected results, based on the specified combination algorithms.
RULES TEST	Asserts if the rules' content is consistent, the effects are well recognized, as well as the corresponding actions.
CARDS TEST	Asserts if the cards' content is consistent and the identifiers are being auto incremented (using the static member).
FACILITIES TEST	Asserts if the facilities' content is consistent and if the accesses are being added to the log.

4.2 Test Results

The results from executing the previous tests are given by Table 7.

Table 7 - Test Results

CLASS	OPERATION	COVERAGE
POLICIES TEST	TestPolicy()	100%
RULES TEST	TestID()	100%
RULES TEST	TestEffect()	100%
CARDS TEST	TestID()	100%
CARDS TEST	TestExpirationDate()	100%
FACILITIES TEST	TestEmptyLog()	100%

FACILITIES TEST	TestAddAccess()	100%
FACILITIES TEST	TestRemoveAccess()	100%

6. Code Generation

After implementing the application in VDM++, it was possible to generate the Java code. To generate the Java code, just right click on the project on Overture and then choose Code Generation -> Generate Java. The generated `.java` files are the ones located in the java folder.

Although it was possible to generate the Java code, we were unable to "connect the dots" and to ensure the application runs smoothly, i.e, it can be executed using the Main function and perform the expected results.

The only possibility to test the generated `.java` files is to create the necessary objects by hand, rather than just executing the Main function, which would wait for some input (request) and then produce a certain output (response).

The generated classes (except test classes) are the ones described by Table 8.

Table 8 - Generated Classes

JAVA CLASS	DESCRIPTION
ACCESS	This class is meant to save the content about a certain target, action and the corresponding effect (the effect quotes are located in the `quotes` folder).
ACTION	This class is meant to save the content about the type of action (the type quotes are located in the `quotes` folder)
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 (the type quotes are located in the `quotes` folder) and the log of accesses to the building.
PAP	This class is meant to have the application's set of policies and make them available to the **PDP**.
PDP	This class is meant to evaluate the application policy and render an authorization decision, applying the corresponding combining algorithms.
PEP	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, and the corresponding target.
REQUEST	This class is meant to save a request status (the status quotes are located in the `quotes` folder).

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.
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.

Beyond this, the test classes were also generated, as described on Table 9.

Table 9 - Generated Test Classes

CLASS	DESCRIPTION
POLICIES TEST	Asserts if the decisions of a certain policy produce the expected results, based on the specified combination algorithms.
RULES TEST	Asserts if the rules' content is consistent, the effects are well recognized, as well as the corresponding actions.
CARDS TEST	Asserts if the cards' content is consistent and the identifiers are being auto incremented (using the static member).
FACILITIES TEST	Asserts if the facilities' content is consistent and if the accesses are being added to the log.

7. Conclusions

7.1 Results Achieved

The results achieved are a bit disappointing since we were expecting to fulfill all the necessary requirements and to develop an application with the adequate usage of VDM++ types. Furthermore, we were unable to generate the Java code correctly and therefore we couldn't test our application as it should be tested. However, despite these flops we were able to implement a structure capable of simulating an access control system in certain conditions.

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 – 40%
- Márcio Fontes – 60%

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. (January, 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.

Annexes

Access Class

```
class Access
	types
		public Effect = <Permit> | <Deny> | <Indeterminate> | <notApplicable>;
	instance variables
		private action : Action;
		private target : Target;
		private effect: Effect;
	operations

		public Access: Target * Action * Effect ==> Access
		Access(t, a, e) ==
			(action := a;
			 target := t;
			 effect := e;
			 return self)
		post action = a and
		    target = t and
		    effect = e;

		public GetAction: () ==> Action
		GetAction () ==
			(return action);

		public SetAction: Action ==> ()
		SetAction(a) ==
			(action := a)
			post (action = a);

		public GetTarget: () ==> Target
		GetTarget () ==
			(return target);

		public SetTarget: Target ==> ()
		SetTarget(t) ==
			(target := t)
			post (target = t);

		public GetEffect: () ==> Effect
		GetEffect () ==
			(return effect);

		public SetEffect: Effect ==> ()
		SetEffect (e) ==
			(effect := e)
			post (effect = e);

end Access
```

Action Class

```
class Action
	types
		public Type = <Assign> | <View> | <Receive>;
	instance variables
		private type : Type;
	operations

		public Action: Type ==> Action
		Action(t) ==
			(type := t; return self)
		post type = t;

		public GetType: () ==> Type
		GetType() ==
			(return type);

		public SetType: Type ==> ()
		SetType(t) ==
			(type := t)
			post (type = t);

end Action
```

Environment Class

```
class Environment
	instance variables
		public permit_date : Date;
	operations

		public Environment: Date ==> Environment
		Environment(pd) ==
			(permit_date := pd;
			 return self)
		post permit_date = pd;

		public GetPermitDate: () ==> Date
		GetPermitDate() ==
			(return permit_date);

		public SetPermitDate: Date ==> ()
		SetPermitDate(pd) ==
			(permit_date := pd)
			post (permit_date = pd);

end Environment
```

Card Class

```
class Card
	types
		public Identifier = nat;
	instance variables
		public static last_identifier : Identifier := 1;
		private identifier : Identifier;
		private exp_date : Date;

	operations

		public Card: Date ==> Card
		Card (ed) ==
			(identifier := last_identifier;
			 exp_date := ed;
			 last_identifier := last_identifier + 1;
			 return self)
		post (identifier = last_identifier~ and
				exp_date = ed and
				last_identifier <> last_identifier~);

		public GetID: () ==> Identifier
		GetID() ==
			(return identifier);

		public static GetLastID: () ==> Identifier
		GetLastID() ==
			(return last_identifier);

		public GetExpDate: () ==> Date
		GetExpDate() ==
			(return exp_date);

		public SetExpDate: Date ==> ()
		SetExpDate(d) ==
			(exp_date := d)
		post (exp_date = d);

		public RemoveExpDate: () ==> ()
		RemoveExpDate() ==
			(dcl date : Date := new Date(1, 1, 1900);
			 exp_date := date)
		pre (exp_date <> new Date(1, 1, 1900));

end Card
```

Facility Class

```
class Facility
	types
		public String = seq of char;
		public Log = seq of Access;
		public Effect = <Permit> | <Deny> | <Indeterminate> | <notApplicable>;
		public Type = <Hotel> | <School> | <Bank>
	instance variables
		private name : String;
		private log : Log := [];
		private type : Type;
	operations

			public Facility: String * Type ==> Facility
			Facility (n, t) ==
				(name := n;
				 type := t;
				 return self)
			post (name = n and
			      type = t);

			public GetName: () ==> String
			GetName () ==
				return name;

			public SetName: String ==> ()
			SetName(n) ==
				(name := n)
			post (name = n);

			public GetType: () ==> Type
			GetType() ==
				(return type);

			public SetType: Type ==> ()
			SetType(t) ==
				(type := t)
			post (type = t);

			public GetLog: () ==> Log
			GetLog () ==
				return log;

			public SetLog: Log ==> ()
			SetLog (l) ==
				(log := l)
			post (log = l);
```

```
        public AddAccess: Access ==> ()
        AddAccess(a) ==
                (log := log ^ [a]);

        public RemoveAccess: Access ==> ()
        RemoveAccess(a) ==
                (dcl new_log : Log := [];
                 for l in log do (
                                if(l <> a)
                                then new_log := new_log ^ [l];
                 );
                 log := new_log;
                )
                pre (len log > 0)
                post (len log < len log~);

        public GetAccessesByType: Effect ==> seq of Access
        GetAccessesByType (effect) ==
                (dcl accesses : seq of Access := [];
                 for l in log do (
                                if (l.GetEffect() = effect)
                                then accesses := accesses ^ [l];
                 );
                 return accesses;
                )
        pre len log <> 0;

end Facility
```

Facility Class

```
class Date

	instance variables
		private day : nat;
		private month : nat;
		private year : nat;

		inv day <= 31 and
		     month <= 12 and
		     if month in set {4, 6, 9, 11}
		     then day <= 30
		     else (month = 2) => (day <= 29);

	operations

		public Date: nat * nat * nat ==> Date
		Date(d, m, y) ==
			(day := d;
			 month := m;
			 year := y;
			 return self)
			post (day = d and
						month = m and
						year = y);

		public GetDay: () ==> nat
		GetDay() ==
			(return day);

		public SetDay: nat ==> ()
		SetDay(d) ==
			(day := d);

		public GetMonth: () ==> nat
		GetMonth() ==
			(return month);

		public SetMonth: nat ==> ()
		SetMonth(m) ==
			(month := m);

		public GetYear: () ==> nat
		GetYear() ==
			(return year);

		public SetYear: nat ==> ()
		SetYear(y) ==
			(year := y);
```

```
		public equals: Date ==> bool
		equals(d) ==
			(return (day = d.day and
						month = d.month and
						year = d.year)
			);

end Date
```

PAP Class

```
class PAP -- Policy Access Point

	instance variables
		private policies : set of Policy;
	operations

		public PAP: set of Policy ==> PAP
		PAP(p) ==
			(policies := p;
			 return self)
			 pre (card policies > 0)
			 post (policies = p);

		public GetPolicies: () ==> set of Policy
		GetPolicies() ==
			(return policies);

		public SetPolicies: set of Policy ==> ()
		SetPolicies(p) ==
			(policies := p)
			pre (card policies > 0)
			post (policies = p);

end PAP
```

PDP Class

```
-- Policy Decision Point
-- Evaluates access requests against authorization policies
-- before issuing access decisions

class PDP
	types
		public CombAlg = <denyOverrides> | <permitOverrides>
	instance variables
		private policies : set of Policy;
		private policyCombAlg: CombAlg;
	operations

		public PDP: set of Policy * CombAlg ==> PDP
		PDP(p, pca) ==
			(policies := p;
			 policyCombAlg := pca;
			 return self)
			pre (card policies > 0)
			post (policies = p and
						policyCombAlg = pca);

		public GetPolicies: () ==> set of Policy
		GetPolicies() ==
			(return policies);

		public GetPolicyCombAlg: () ==> CombAlg
		GetPolicyCombAlg() ==
			(return policyCombAlg);

		public AddPolicy: Policy ==> ()
		AddPolicy(p) ==
			policies := policies union {p}
		post (p in set policies);

		public RemovePolicy: Policy ==> ()
		RemovePolicy(p) ==
			(policies := policies \ {p};)
			pre (card policies > 0 and
					p in set policies)
			post (not p in set policies);

end PDP
```

Policy Class

```
class Policy
	types
		public CombAlg = <denyOverrides> | <permitOverrides>;
		public Effect = <Permit> | <Deny> | <Indeterminate> | <notApplicable>;
	instance variables
		private target : Target;
		private rules : set of Rule;
		private ruleCombAlg: CombAlg;

		inv forall r in set rules & r.GetTarget() = target;
		--[r | r in set rules & r.GetTarget() = target];

	operations

			public Policy: Target * set of Rule * CombAlg ==> Policy
			Policy(t, r, rca) ==
				(target := t;
				 rules := r;
				 ruleCombAlg := rca;
				 return self)
				pre (card r > 0)
			 post (target = t and
						rules = r and
						ruleCombAlg = rca);

			public GetTarget: () ==> Target
			GetTarget() ==
				(return target);

			public SetTarget: Target ==> ()
			SetTarget(t) ==
				(target := t)
				post (target = t);

			public GetRules: () ==> set of Rule
			GetRules() ==
				(return rules);

			public SetRules: set of Rule ==> ()
			SetRules(r) ==
				(rules := r)
				pre (card r > 0)
				post (rules = r);

			public GetRuleCombAlg: () ==> CombAlg
			GetRuleCombAlg() ==
				(return ruleCombAlg);
```

```
public SetRuleCombAlg: CombAlg ==> ()
SetRuleCombAlg(rca) ==
        (ruleCombAlg := rca)
        post (ruleCombAlg = rca);

public GetRulesByEffect: Effect ==> set of Rule
GetRulesByEffect(effect) ==
        (dcl permits : set of Rule := {};
         for all r in set rules do (
                        if(r.GetEffect() = effect)
                        then permits := permits union {r};
         );
         return permits;
        )
        pre (card rules > 0);

        -- Still need to check the pre and post conditions

public GetDecision: () ==> Effect
GetDecision() ==
        (dcl decision : Effect := <Deny>;
         for all r in set rules do (
                if(ruleCombAlg = <denyOverrides>)
                then (if(r.GetEffect() = <Deny>)
                                then (decision := <Deny>;
                                                return decision))
                elseif(ruleCombAlg = <permitOverrides>)
                then (if(r.GetEffect() = <Permit>)
                                then (decision := <Permit>;
                                                return decision))
         );
         decision := <Indeterminate>;
         return decision;
        )
        pre (card rules > 0 and
                        ruleCombAlg <> nil);

end Policy
```

Request Class

```
class Request
	types
		public String = seq of char;
		public Status = <Active> | <Pending> | <Finished>;
	instance variables
		private line : String;
		private status: Status;
	operations

		public Request: String ==> Request
		Request(l) ==
			(line := l;
			 status := <Pending>;
			 return self)
			post (line = l and
						status = <Pending>);

		public GetStatus: () ==> Status
		GetStatus () ==
			(return status);

		public SetStatus: Status ==> ()
		SetStatus(s) ==
			(status := s)
			post (status = s);

end Request
```

Resource Class

```
class Resource
	types
		public String = seq of char
	instance variables
		private name : String;
	operations

		public Resource: String ==> Resource
		Resource(n) ==
			(name := n;
			 return self)
			 pre (len n > 0)
			 post (name = n);

		public GetName: () ==> String
		GetName() ==
			(return name);

		public SetName: String ==> ()
		SetName(n) ==
			(name := n)
			pre (len n > 0)
			post (name = n);

end Resource
```

Subject Class

```
class Subject
	types
		public String = seq of char
	instance variables
		private name : String;
		private subject_card : Card;
	operations

		public Subject: String * Card ==> Subject
		Subject (n, sc) ==
			(name := n;
			 subject_card := sc;
			 return self)
		pre (len n > 0)
		post (name = n and
					subject_card = sc);

		public GetName: () ==> String
		GetName() ==
			(return name);

		public SetName: String ==> ()
		SetName(n) ==
			(name := n)
		pre (len n > 0)
		post (name = n);

		public GetCard: () ==> Card
		GetCard() ==
			(return subject_card);

		public SetCard: Card ==> ()
		SetCard(sc) ==
			(subject_card := sc)
		post (subject_card = sc);

end Subject
```

Rule Class

```
class Rule
	types
		public Identifier = nat;
		public Effect = <Permit> | <Deny> | <Indeterminate> | <notApplicable>;
		public String = seq of char;
	instance variables
		private identifier: Identifier;
		private description: String;
		private target : Target;
		private effect : Effect := <Deny>;
		private facilityGroup : String;
		private userGroup : String;
		private temporalConstraint : Date := new Date(1, 1, 1900);
		public static last_identifier: Identifier := 1;

	operations

		public Rule: String * Target * Effect * String * String ==> Rule
		Rule(d, t, e, fg, ug) ==
			(identifier := last_identifier;
			 description := d;
			 target := t;
			 effect := e;
			 facilityGroup := fg;
			 userGroup := ug;
			 last_identifier := last_identifier + 1;
			 return self)
		post (description = d and
					target = t and
					effect = e and
					facilityGroup = fg and
					userGroup = ug and
					last_identifier <> last_identifier~);

		public Rule: String * Target * Effect * String * String * Date ==> Rule
		Rule(d, t, e, fg, ug, tc) ==
			(identifier := last_identifier;
			 description := d;
			 target := t;
			 effect := e;
			 facilityGroup := fg;
			 userGroup := ug;
			 temporalConstraint := tc;
			 last_identifier := last_identifier + 1;
			 return self)
		pre (tc <> new Date(1, 1, 1900))
		post (description = d and
					target = t and
					facilityGroup = fg and
					userGroup = ug and
					temporalConstraint = tc and
					last_identifier = last_identifier~);
```

```
public GetIdentifier: () ==> Identifier
GetIdentifier() ==
        (return identifier);

public static GetLastID: () ==> Identifier
GetLastID() ==
        (return last_identifier);

public GetDescription: () ==> String
GetDescription() ==
        (return description);

public SetDescription: String ==> ()
SetDescription(d) ==
        (description := d)
        post (description = d);

public GetTarget: () ==> Target
GetTarget() ==
        (return target);

public SetTarget: Target ==> ()
SetTarget(t) ==
        (target := t)
        post (target = t);

public GetEffect: () ==> Effect
GetEffect() ==
        (return effect);

public SetEffect: Effect ==> ()
SetEffect(e) ==
        (effect := e)
        post (effect = e);

public GetFacilityGroup: () ==> String
GetFacilityGroup() ==
        (return facilityGroup);

public SetFacilityGroup: String ==> ()
SetFacilityGroup(fg) ==
        (facilityGroup := fg)
        post (facilityGroup = fg);

public GetUserGroup: () ==> String
GetUserGroup() ==
        (return userGroup);

public SetUserGroup: String ==> ()
SetUserGroup(ug) ==
        (userGroup := ug)
        post (userGroup = ug);
```

```
            public GetTemporalConstraint: () ==> Date
            GetTemporalConstraint() ==
                (return temporalConstraint);

            public SetTemporalConstraint: Date ==> ()
            SetTemporalConstraint(tc) ==
                (temporalConstraint := tc)
                pre (not tc.equals(new Date (1, 1, 1900)))
                post (temporalConstraint = tc);

end Rule
```

SYS Class

```
class SYS
    types
        public String = seq of char;
    instance variables
        private name : String;
        private facility: Facility;
        private requests : seq of Request := [];
    operations

        public SYS: String * Facility ==> SYS
        SYS(n, f) ==
            (name := n;
             facility := f)
            post (name = n and
                        facility = f);

        public GetName: () ==> String
        GetName() ==
            (return name);

        public GetFacility: () ==> Facility
        GetFacility() ==
            (return facility);

        public AddRequest: Request ==> ()
        AddRequest(r) ==
            (requests := requests ^ [r]);

        -- The requests must be forwarded to the PEP

end SYS
```

Target Class

```
class Target

	instance variables
		private subjects : set of Subject;
		private resources : set of Resource;
		private actions : set of Action;
		private environment: Environment;
	operations

		public Target: set of Subject * set of Resource * set of Action ==> Target
		Target(s, r, a) ==
			(subjects := s;
			 resources := r;
			 actions := a;
			 return self)
			pre (card s > 0 and
					card r > 0 and
					card a > 0)
			post (subjects = s and
						resources = r and
						actions = a);

		public Target: set of Subject * set of Resource * set of Action * Environment ==> Target
		Target(s, r, a, e) ==
			(subjects := s;
			 resources := r;
			 actions := a;
			 environment := e;
			 return self)
			pre (card s > 0 and
					card r > 0 and
					card a > 0)
			post (subjects = s and
						resources = r and
						actions = a and
						environment = e);

		public GetSubjects: () ==> set of Subject
		GetSubjects() ==
			(return subjects);

		public SetSubjects: set of Subject ==> ()
		SetSubjects(s) ==
			(subjects := s)
			pre (card s > 0)
			post (subjects = s);

		public GetResources: () ==> set of Resource
		GetResources() ==
			(return resources);
```

```
        public SetResources: set of Resource ==> ()
        SetResources(r) ==
                (resources := r)
                pre (card r > 0)
                post (resources = r);

        public GetActions: () ==> set of Action
        GetActions() ==
                (return actions);

        public SetActions: set of Action ==> ()
        SetActions (a) ==
                (actions := a)
                pre (card a > 0)
                post (actions = a);

        public GetEnvironment: () ==> Environment
        GetEnvironment() ==
                (return environment);

        public SetEnvironment: Environment ==> ()
        SetEnvironment(e) ==
                (environment := e)
                post (environment = e);

end Target
```