2014 14th International Conference on Quality Software

Model-Based Testing of Obligations

Iram Rubab[1], Shaukat Ali[2], Lionel Briand[1], Yves LeTraon[1]
[1]Snt Centre for Security,Reliability and Trust, University of Luxembourg, Luxembourg
[2]Certus Software V&V Centre, Simula Research Laboratory, Lysaker, Norway
shaukat@simula.no
{iram.rubab, lionel.briand, yves.letraon} @uni.lu

***Abstract*— Obligations are mandatory actions that users must perform, addressing access control requirements. To ensure that such obligations are implemented correctly, an automated and systematic testing approach is often recommended. One such approach is Model-Based Testing (MBT) that allows defining cost-effective testing strategies to support rigorous testing via automation. In this paper, we present MBT for obligations by extending the Unified Modeling Language (UML) via a profile called the *Obligations Profile*. Based on the profile, we define a modeling methodology utilizing the concepts of Obligations Class Diagrams (OCDs) and Obligations State Machines (OSMs), which are standard UML Class Diagrams and UML State Machines with stereotypes from the *Obligations Profile*. Our methodology, using OCDs and OSMs, is automatically enforced by the validation of constraints defined in the profile. To assess the completeness and applicability of the profile and methodology, we modeled 47 obligations from four different systems. The results of our case study show that we successfully modeled all the obligations and used 75% of the stereotypes that we defined in the profile.**

In addition, using OCDs and OSMs, we automatically generate executable test cases using a standard state machine structural coverage criterion and common test data generation strategies. The effectiveness of generated test cases is assessed using mutation analysis on two systems, using mutation operators specifically designed for obligation faults. Test case execution killed 75% of the mutants and a careful analysis further suggests that more sophisticated testing strategies must be defined to further improve testing effectiveness.

Keywords—Obligations. Access control policy. UML profile. UML class diagram. UML state machines. Model based testing

I. Introduction

Access control rules are security policies to enable/disable the access of users of a system to its resources and are pivotal for preserving security properties such as confidentiality, integrity, and privacy. Access control requirements (implemented as rules) are addressed with obligatory actions that must be performed by users. Such obligatory actions are commonly referred to as obligations that can be performed before, during, or after the enforcement of access rules.

To ensure the correct implementation of obligations, one possible approach is to resort to Model-Based Testing (MBT), an approach that is gaining a lot of attention in academia and industry [1, 2]. MBT relies on systematic behavior modeling of the System Under Test (SUT), at an adequate level of abstraction, such as to facilitate the automatic generation of executable test cases. Such an approach is also potentially applicable for testing obligations, which is the goal of this paper.

To provide MBT for obligations based on standard modeling notations, we developed a UML profile for modeling obligations called *Obligations Profile*. On top of the profile, we defined a modeling methodology to use the profile for modeling Obligation Class Diagrams (OCDs) and Obligation State Machines (OSMs). Both are standard UML Class Diagrams and UML State Machines making use of stereotypes from the *Obligations Profile*. An OCD captures state variables, operations, and signals required to model the behavior of obligations, whereas an OSM models the state-based behavior of obligations. The methodology guides the modeling of OCDs and OSMs by including well-formedness constraints that can be enforced by a tool during modeling, expressed in the Object Constraint Language (OCL) [3] and defined on the stereotypes of the profile.

The OCDs and OSMs modeled using the *Obligations Profile* are further used to generate executable test cases. As an initial step towards test case generation from the profile, we implemented the *All Transitions coverage criterion* [4], along with random test data (RD) generation and Equivalence Class Partitioning (ECP) using boundary values [1], to obtain executable test cases.

To assess the *Obligations Profile* and effectiveness of test cases generated from OCDs and OSMs, we performed an empirical evaluation involving two activities. First, we modeled 47 obligations of four different systems to assess the completeness and applicability of the profile. The results showed that we successfully modeled all 47 obligations using the profile and noticed that 25% of the stereotypes had not been used while modeling the systems. The results suggest that our profile is sufficiently complete to model the required types of obligations and that 25% of the stereotypes may turn out to be unnecessary and we may remove them from the profile in the future. In the second activity, we generated executable test cases for two systems for which implementations are available. The effectiveness of the generated test cases was then assessed using mutation analysis, where the implementations of the systems were mutated using mutation operators that we specifically defined to introduce obligation faults [5]. The results showed that, for both target systems, when using ECP for test case generation, 75% of mutants were killed across the two systems. Though encouraging, this further suggests that we need to define more focused testing strategies to further improve the fault detection effectiveness.

The remainder of this paper is organized as follows: Section II presents the running example of the Medical System (MS) that we will use throughout the paper to explain various concepts, Section III provides the *Obligations Profile*, and Section IV describes our modeling methodology. In Section V, we discuss how we generate test cases from OCDs and OSMs and Section VI presents the empirical evaluation. The related work is discussed in Section VII and Section VIII concludes the paper.

II. Running Example

In this section, we describe a Medical System (MS) and its obligations that will be used as a running example for the *Obligations Profile.*

MS is an information system, which maintains information about doctors and patients in a hospital. The information about doctors includes their personal information such as name and address, and professional information such as duty, schedule, and patient's information, such as patient's medical history. Hospital records, that include general information about hospitals, are also part of MS. MS has three types of users: (1) administrators that manage hospital records, (2) doctors who manage their own information as well as the records of their patients, and (3) patients who are allowed to see their medical history. MS has an access control policy that includes access rights and obligations, which restrict the access of users to the resources of the system.

III. Obligations Profile

In this section, we define a UML profile for modeling obligations called the *Obligations Profile*. The profile is defined by lightweight extensions of UML.

In general, a UML profile enables the extension of UML for different domains and platforms, while avoiding any contradictions with UML semantics. Two main approaches for profile creation are discussed in the literature [6]. The first approach directly implements a profile by defining key concepts of a target domain, such as what was done to define SysML [7]. The second approach introduces an intermediate step by first creating the conceptual model of the domain concepts before creating a profile for the identified concepts. This latter approach has been used for defining profiles such as the UML profile for Modeling and Analysis of Real-time Embedded Systems (MARTE) [8] and the UML Testing Profile (UTP) [9] .

We used the second approach to define the *Obligations Profile* since it is more systematic as it separates the profile creation process into two stages. In the first stage, we develop a conceptual model of the domain concepts and their relationships. In the second stage, we identify the mapping between the main concepts and UML modeling elements and define corresponding stereotypes. We also define constraints on stereotypes and the relationships between stereotypes.

A. *Conceptual Model for Obligations*

In Fig 1, we present a conceptual model for obligations as a UML Class Diagram. Our conceptual model of obligations is inspired by the work of Park and Sandhu [10] and the work of Elrakaiby et al. [5]. Park and Sandhu [10] categorizes obligations related to usage control such as pre, on, and post obligations. Elrakaiby et al. [5] defines types of obligations based on their state-based behavior such as fulfillment obligations, continuous obligations, and persistent obligations. The concepts of conceptual model are defined below:

B. *Access Control Policies and Obligations*

An Access Control Policy (ACP) consists of a set of access rules and obligations that are conditions restricting the access of users on system resources.

1) Definition 1: An ACP consists of a set of access rules and a set of obligations, i.e., *ACP = {{Access Rule}, {Obligation}}*, where *Access Rule* = $\{r_1, r_2, .., r_{nr}\}$, *nr* is the number of access rules related to *ACP*, and r_i is either a permission or a prohibition. An *Access Rule* of type permission enables access to system resources, whereas a prohibition rule denies such access.

Obligations are commonly defined as actions that users are required to take and are necessary for the expression of a variety of requirements. For example, obligation actions may be linked to: (1) access requests of users, such as a requirement to be fulfilled before accessing a resource (*Pre* obligations), (2) essential requirements with an on-going access (*On* obligations), (3) post requirements after accessing a resource (*Post* obligations), or (4) satisfying requirements such as authorization.

2) Definition 2: An obligation Ob is a set Ob = {*Type, User, Object, Action, Context*} where Type = {*On, Pre, Post, Fulfillment, Continuous, Negative, Persistent*}.

We categorize obligations into two main types as shown in Fig 1: *Activation* and *Enforcement* obligations. *Activation* obligations are linked with access rules. The *Activation* type is further divided into *Pre, On* and *Post* obligations. The actions that have to be implemented before allowing an access request are called *Pre* obligations. For example, in MS, a doctor submits the report of a patient only when the doctor has access to the patient. An *On* obligation is to be fulfilled while accessing a resource. For example, the doctor should submit their patient's report everyday as long as the patient has been admitted to the hospital.

The *Post* obligation is to be satisfied after the access to a resource is completed. For example, the record of a patient has to be deleted two years after the last examination took place.

Enforcement is a concept that is used to categorize obligations into *Fulfillment, Persistent, Negative and Continuous*. A *Fulfillment* obligation is violated if it is not satisfied within specific time duration. For example, the obligation of a doctor to examine a patient is active until the end of the day. If it is not fulfilled in time, it becomes violated and is no more active. A *Persistent* obligation remains active even after the deadline is over and *a Continuous* obligation remains active during some activity. For example, a doctor should examine at least one patient per his/her shift. *Negative* obligations are equivalent to prohibitions that deny the access of user to some resources in the system. An example of *Negative* obligation is when a doctor is forbidden to examine patients with a disease for which the doctor is not qualified.

An obligation has at least one *User*. For example, one of the obligations of MS, named *Emergency Call*, states that an on-duty doctor should respond to emergency calls within 60 seconds after receiving the call. In *Emergency Call*, the doctor is the *User*, who is responsible to satisfy this obligation. *User* is defined as a set of potential actual users for an obligation. A user can be a *System* or a *Human.*

An obligation has at least one *Object*, which is the resource on which the obligation is to be performed.

An *Action* in an obligation is an activity that the user is obliged to perform. In software terms, it is one or more system operations that user must perform within a given context. We categorize actions in three types: *Administrativ*e, *Non-Administrative* and *Sanction.* An *Administrative* action involves management decisions, such as restricting the access rights of a user, while a *Non-Administrative* action is not related to managerial tasks and could be as simple as submitting a report. In *Emergency Call* (MS), the action is to respond to emergency and it is a *Non-Administrative* action. Each obligation should have at least one *Action. Sanction* is an optional *Action* that is applicable only in case of the violation of an obligation. A *Sanction* is either some kind of restriction or another obligation to be satisfied as a penalty, such as restricting access to a particular service in case of late payment. In MS, an example of *Sanction* is when the doctor will submit a violation of duty report if he/she fails to submit the examination report of the patient in due time.

An obligation has a *Context* that activates or deactivates it. In the former case, it is called an *Initiation* context and in the latter, a *Violation* context. A *Context* in obligations is either a constraint such as a timing constraint or is an event from the environment. Most of the obligations have time units, time delays, or time intervals depending on the context. For example, in *LFS Alert* in Fig 4*,* the *Initiation* context is 'when the life support system fails' and the *Violation* context is 'within 30 seconds'.

An obligation may have a *Condition* attached to it. For example, in an obligation such as 'A customer of a company must receive a notice every six months about the privacy policy of the company, as long as the customer relationship is not terminated', the obligation 'A customer in a company must receive a notice every six month about the privacy policy of the company' is satisfied only on condition 'as long as the customer relationship is not terminated.

C. *Obligations Profile*

In this section, we present the *Obligations Profile*. The profile diagram is presented in Fig 2 and Fig 3. In these figures, we defined stereotypes for *Obligations Profile* and their relationships with UML meta-classes. The left-hand side of Fig 2 is a representation of the *‹‹Obligation››* stereotype that extends from UML State Machine and UML Class meta-classes. The stereotypes for various types of obligations are also defined such as *‹‹Pre››*, *‹‹Post››*, *‹‹On››*, *‹‹Fulfillment››*, *‹‹Persistent››*, *‹‹Negative››* and *‹‹Continuous››*. The right-hand side of Fig 2 shows the *‹‹Context››* stereotype. A *‹‹Context››* can be applied

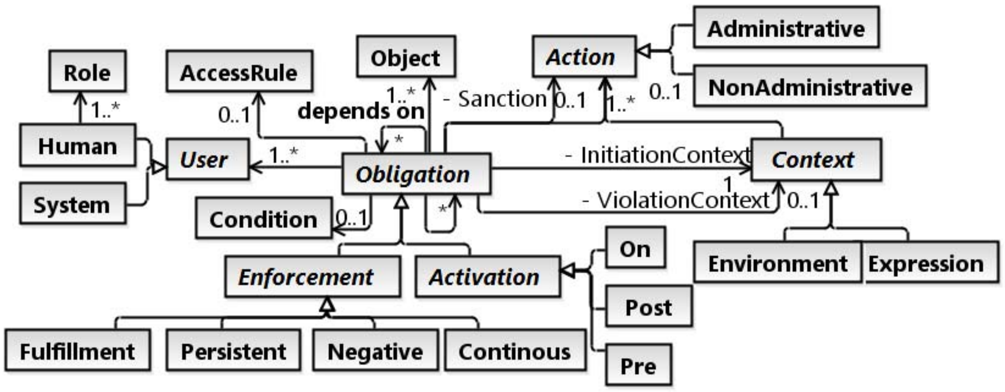

Fig 1 Conceptual Model for Obligations

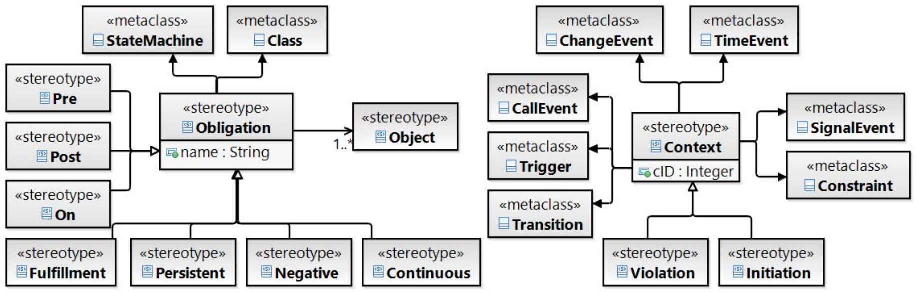

Fig 2. *‹‹Obligation››* and *‹‹Context››* stereotypes with their respective meta-classes

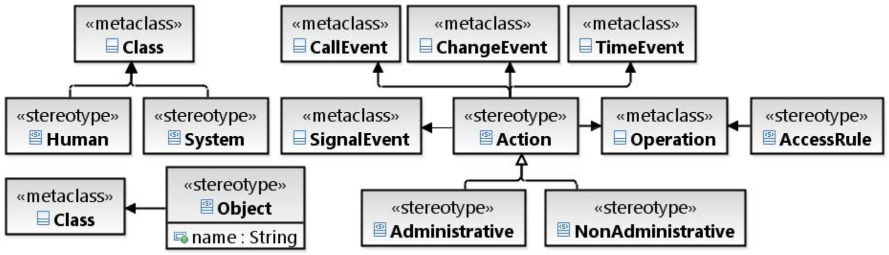

Fig 3.The stereotypes of *‹‹User››, ‹‹Object››* and *‹‹Action››* with their respective meta-classes

on the following UML meta-classes: *Transition, Trigger, CallEvent, ChangeEvent, TimeEvent, SignalEvent,* and *Constraint.* The types of *‹‹Context ››* are *‹‹ Initiation››* and *‹‹Violation››* contexts and they are also shown in the same diagram.

In Fig 3, the stereotypes for the user of obligation, i.e., *‹‹Human››* and *‹‹System››* extends the UML meta-class: *Class*. The stereotype *‹‹Object››* is also presented in Fig 3 extending the UML meta-class *Class*. The right-hand side of Fig 3 shows the *‹‹Action››* stereotype that extends from UML meta-classes of *Operation*, *CallEvent*, *ChangeEvent*, *TimeEvent, SignalEvent*, and *Constraint*. A *‹‹Action››* is either *‹‹Administrative››* or *‹‹NonAdministrative››* action.

In the next step, we present OCL constraints that are part of the *Obligations Profile*, in order to enforce mandatory modeling steps. The constraints force modelers to comply with the *Obligations Profile* through automatic enforcement of the constraints by the modeling tool. In our case we used IBM Rational Software Architect [11]. We have defined ten different constraints on the stereotypes in the *Obligations Profile*, but due to space constraints, we show only some of the constraints applied on stereotype *‹‹Obligation››*. The constraint on *‹‹Obligation››* shown in Table 1 states that exactly one class should be stereotyped as *‹‹Obligation››* in one OCD. The constraint on OSM as shown in Table 2 restrict a user to define exactly one *InActive* and exactly one *Active* state while modeling the OSM.

IV. Obligation Modeling Methodology

In this Section, we present the methodology to model obligations using the *Obligations Profile*. We precisely define OSM and OCD and provide their constraints.

A. *Obligation Class Diagram*

An OCD is a UML Class Diagram with stereotypes defined in the *Obligations Profile*. A detailed definition of OCD is provided in Table 3.

B. *Obligation State Machine*

An OSM is a UML State Machine with stereotypes defined in the *Obligations Profile*. A detailed definition of OSM is provided in Table 4.

C. *LFS Alert (Example)*

An obligation from MS is used to illustrate the modeling of OCD and OSM. The *LFS Alert* states that when a patient's life support system fails, an alert should be immediately sent to the patient's doctor and to the hospital's biomedical equipment technicians within 30 seconds after the failure. The OCD and OSM of *LFS Alert* are shown in Fig 4. In this diagram, we use stereotypes of *‹‹Obligation››* and *‹‹System››* in OCD and *‹‹Initiation››* and *‹‹Violation››* for OSM. Initially, the obligation is *InActive*, but on receiving signal *LFSAlert,* the state changes to *Active.* If an alarm is generated within 30 seconds, then the obligation is in state *Fulfilled* otherwise the state is *Violated.*

Table 1. OCL Constraint on Obligation Class

```
Context Obligation inv:
self.base_Class->any(c|c.name='Class').oclAsType(uml::Class) ->size()=1
```

Table 2. OCLConstraint on Obligation State Machine

```
Context Obligation inv:
self.base_Class ->any (b|b.name='StateMachine'). oclAsType
(uml::StateMachine).region.state
->select (s|s.name= 'Active')->size() =1 and
self.base_Class ->any
(b|b.name='StateMachine').oclAsType(uml::StateMachine).region.state
->select (s|s.name= 'InActive')->size() =1
```

V. Model-based Testing Of Obligations

A. *Obligations Profile for Testing*

In this section, we show the mapping of testing concepts in OCDs and OSMs that are essential to generate executable test cases such as test path, test case, and test data [1, 2]. The mapping is shown in Table 5 with examples based on the running example.

B. *MBT Tool for Test Case Generation*

The architecture of our prototype tool implemented in Java, is shown in Fig 5. It generates executable test cases in Java from OCD and OSM. The purpose of the tool is to facilitate our evaluation to assess the effectiveness of the generated test cases. Other existing MBT tools such as Smartesting [11] and TRUST [4] can be extended to support test case generation in the future. Our prototype tool has two main components: *Test Path Generator* (TPG) and *Test Case Generator* (TCG). *TPG* takes *OCD* and *OSM* as input and generates a set of test paths based on a structural coverage criterion (*All Transitions* coverage in current implementation). Each test path is in Java without actual test data and each test path contains Java assertions corresponding to OCL state invariants that are transformed using Dresden OCL [12]. Each test path is further transformed into a set of executable test scripts in Java based on test data using *Test Case Generator*. Test data is obtained automatically using EsOCL [13], which takes input constraints in OCL and provides test data.

VI. Empirical Evaluation

In this section, we present the empirical study that have been conducted to assess the obligations modeling and testing approach we proposed.

A. *Study Design*

We first present the research questions, and then describe the case studies. Last, we define the metrics used to assess our approach from different perspectives.

1) Research Questions: The two following research questions have been studied to assess our approach.

a) RQ1: Profile Completeness. Is the *Obligations Profile* complete for modeling various types of obligations?

b) RQ2: *Test Cases Effectiveness.* What is the effectiveness of test cases generated from models using the *Obligations Profile* in terms of its capacity to reveal obligation faults?

TABLE 3. OBLIGATION CLASS DIAGRAM

An obligation class diagram is a UML 2.0 Class Diagram stereotyped as ‹‹Obligation›› and it includes the following UML 2.0 Class Diagram elements.

a) **C:** A set of classes of the following types.
b) Exactly one class that is stereotyped as *‹‹Obligation››*. This class has all the attributes, operations, and signal receptions that describe the obligation.
c) An optional class that is stereotyped as *‹‹Human››* or *‹‹System››* to define the user of the obligation.
d) An optional class that is stereotyped as *‹‹Object››* to define the object of the obligation.
e) **R**: A set of relationships that include all those relationships that show communication between different classes in OCD.
f) OCDs may have signals, enumerations, and data types if required. The signals are modeled to represent an input from environment. Enumerations and Data Types are used to model new data types when the primitive types are not sufficient.

TABLE 4. OBLIGATION STATE MACHINE

An obligation state machine is a UML 2.0 State Machine stereotyped as *‹‹Obligation››* or any of its types from *Obligations Profile.* OSMs consist of following UML 2.0 State Machine elements.

1. I: An initial state.
2. F: A set of one or more final states.
3. MS: A set of mandatory states of the following types.
a) A state *InActive* in MS that is the state of the obligation before its activation. The state *InActive* has a state invariant that is defined over one or more attributes from *‹‹Obligation››* Class in OCD. The state invariant is an OCL constraint that provides information on the status of the state. An OSM has state invariant with each of its state except with UML Pseudostate including initial state and the final state.
A state ‘Active’ in MS that represents the state of obligation when it is activated. The state ‘Active’ has a state invariant that is defined on an attribute from *‹‹Obligation››* Class in OCD.
b) It is mandatory to add either *Fulfilled* or *Violated* state or both *’Fulfilled*’ and *Violated* states in an OSM. Each of the state has state invariants.
4. **NS:** A set of non-mandatory states includes states of the following types.
g) A state named *Fulfilled* or *Violated* can be part of the set NS only if it does not exist in MS. It is possible to have none of the states from *Fulfilled* or *Violated* in the set NS when they are already part of MS.
h) The set NS may have one or more optional states depending on the details of the obligation and choice of the modeler. For example, in order to verify access request in a *Pre* obligation, we added a state named ‘Valid’, similarly more states can be added when there are many constraints.
5. **T:** A set of transitions in OSM connecting states in MS or NS with each other. Following are the types of transitions.
a) A transition from an initial state to *InActive* state, this transition is without any event, guard, or effect.
b) One or more transitions from any state (except the initial state) to the final state.
i) A transition t in set T between states *InActive* and Active in MS. The transition may contain a trigger such as change event, call event, time event, or signal event depending on the obligation. It can be due to an operation call, signal, or a timed attribute from *‹‹Obligation››* class. The trigger is stereotyped as *‹‹Initiation››*. This transition may have a guard and an effect.
j) A transition t in T is added from any state in MS (including *Active* state) to any other state in MS or NS. The transition may have trigger i.e. a change event, call event, time event or signal event, it may have a guard and effect. The effect is generally stereotyped as *‹‹Action››*.

2) Case Study: We selected four different systems implementing various types of obligations. These systems include Crisis Management System (CMS) [14], Home Care System (HCS) [15], Virtual Meeting System (VMS) [16] and Medical System (MS). MS is already presented in Section II as a running example.

a) Home Care System (HCS): This system is developed as part of the CoPAInS (Conviviality and Privacy in Ambient INtelligent Systems) project [15].

The HCS is built for different ambient intelligent scenarios, some of them are validated by HotCity, the largest WiFi network in Luxembourg in the context of CoPAInS. HCS scenarios are implemented using the Kevoree [17] platform, which is an open source environment that facilitates component-based development of distributed systems. We took two scenarios that are already implemented in HCS to test their respective obligations.

In the first scenario, HCS is deployed at a patient’s home to help in case of an emergency triggered either by sensors or by the patient. The HCS forwards the patient’s request message to a neighbor or to the ambulance. The neighbor can accept or decline the request for help. The neighbor can view the patient’s current conditions through a camera. In case, no neighbor is ready to help, the request for help is sent to an ambulance. Different components interact with each other to achieve this scenario, such as Emergency Call List (ECL), Remote Controller, Door Sensor, Video Camera, Timer, and Smart Phone Application. Based on the current available implementation of the scenario, we tested five obligations for this scenario. In the second scenario, HCS is used for remote monitoring of health parameters and uses Electrocardiogram (ECG), Pulse Oximeter, and Blood Pressure regulator for regular check-ups of patient’s heartbeat, blood oxygen level, and blood pressure respectively. In case of any abnormality, HCS sends a message to the hospital. The hospital informs doctor and responds back to HCS. We modeled and tested ten additional obligations of HCS related to this scenario. These two scenarios of HCS consist of 3638 lines of Java code.

b) Virtual Meeting Management System (VMS): VMS implements web conferencing services and is a prototype system developed in our previous work [16]. A user can organize meetings and invite other participants. The organizer plans a meeting and sets its main parameters, such as name, agenda, and participants. The organizer can appoint a moderator for the meeting, who allocates the floor to the participant. VMS has 6070 lines of Java code in 134 classes. We modeled and tested three obligations of VMS.

c) Crisis Management System (CMS): CMS [14] is designed to help in identifying and handling a crisis such as earthquakes, tsunami, floods, terrorist attacks, and accidents.

CMS provides ways for communication and coordination among different users such as coordinators, hospital staff, firemen, and technicians. CMS supports its users in allocating necessary resources to handle a crisis and provide access to relevant crisis information. Since we do not have access to the implementation of CMS, we used the system only for evaluating the *Obligations Profile.*

All four case studies (notice that MS is presented in Section II) are used to evaluate the *Obligations Profile* by modeling their respective obligations, while only HCS and VMS are used for testing since their implementation is available.

3) Case Study Measures: We defined the following measures to address each research question.

To answer RQ1, we aim to assess the completeness of the *Obligations Profile.* The completeness is assessed based on two criteria. The first criterion is an estimate of whether we can model all the obligations of the four case studies. This is measured as follows:

TABLE 5 OBLIGATIONS PROFILE FOR TESTING

Testing Concept	Testing Concepts from *Obligations Profile*	Example
Test Path	A path in an OSM starting from initial state and ending at the final state. *InitialState.Transition$_1$.State$_1$.…FinalState* where, *Transition$_x$* consists of an optional *Guard* as OCL constraint, a *Trigger*, and an optional *Effect* *State$_y$*: Name and an OCL constraint specifying state invariant	[*Initial State*].[*InActive*]. LFSFails()[*Active*].after 30 sec [*Violated*].[*Final State*] in Fig 4.
Test Data	Test data is derived from OCL constraints on Guards using EsOCL [13] based on RD and ECP with boundary values	t>30 seconds; (Time Event) in Fig 4 RD: t=52 ECP with Boundary Value Analysis Test Data: t<=0 {-5, -1, 0, 1}; t>0 and t<30 {-1, 0, 1, 5, 15, 29, 30, 31}; t>=30 {29, 30, 31, 40}

Test Coverage	Structural Coverage: *All Transition* coverage on an OSM
	Data Coverage: One RD value per guard and ECP with boundary values from guards for inputs to triggers on transitions.
		All Transitions of an OSM shown in Fig 4:
		TestPath1= [*Initial State*].[*InActive*]. LFSFails()[*Active*].after 30 sec [*Violated*].[*Final State*]
		TestPath2=[*InitialState*].[*InActive*]. LFSFails()[*Active*].generateAlert()[*Fulfilled*].[*Final State*]

Expected Result	OCL constraints as state invariants of states that are transformed in to Java assertions using Dresden OCL [12]	State Invariant of *InActive* State in Fig 4, Alert:self.status=OStatus::inactive
Pass/Fail Criteria	It is true/false as a result of the evaluation of an assertion in Java	State Invariant of *InActive* State in Fig 4 Alert:self.status=OStatus::inactive evaluates to true/false during testing

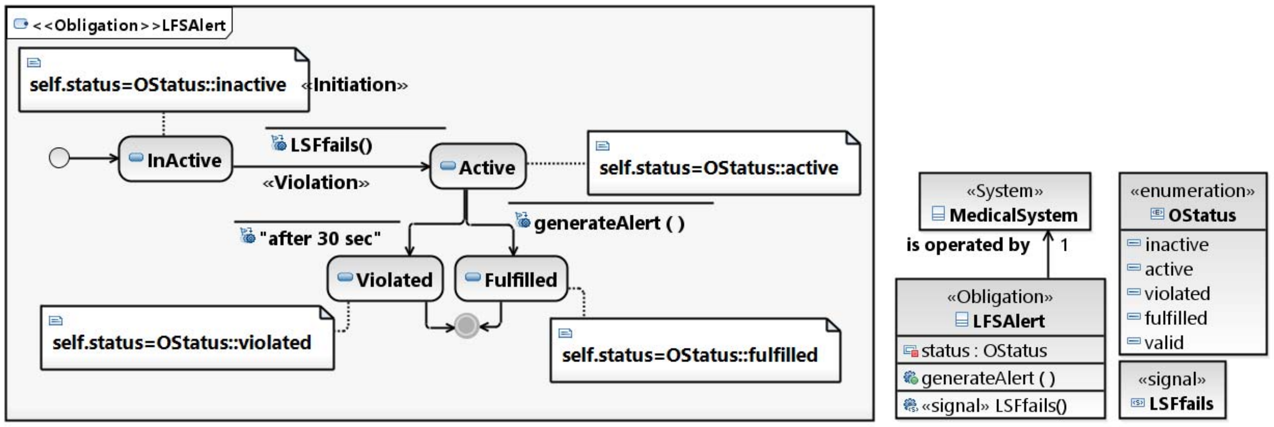

Fig 4 LFS Alert Obligation Class Diagram and Obligation State Machine

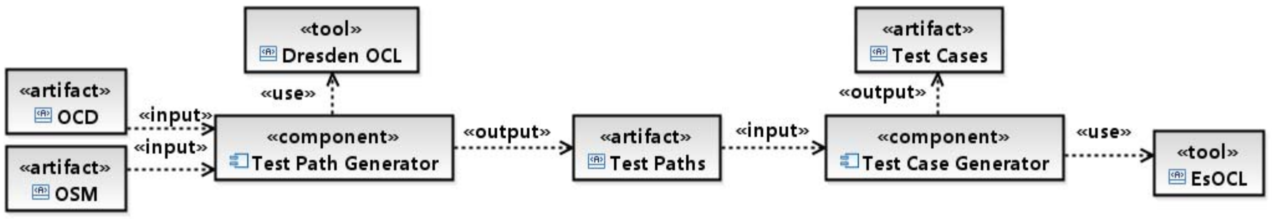

Fig 5 A high-level architecture of test case generation tool

$$Completeness_{obj} = \frac{number\ of\ modeled\ obligations}{total\ number\ of\ obligations}$$

Second, we measure whether each stereotype defined in the *Obligations Profile* is used at least once for modeling the obligations in the four case studies. The purpose of such measurement is to assess whether a particular stereotype is potentially unnecessary. This is measured as follows:

$$Completeness_{ster} = \frac{number\ of\ types\ of\ stereotypes\ modeled\ in\ a\ case\ study}{number\ of\ types\ of\ stereotypes\ defined\ in\ the\ Obligations\ Profile}$$

In the current version of the profile, we have defined 19 types of stereotypes. If in a case study, we used only five types of stereotypes, then $Completeness_{ster}$ is 5/19. Completeness of each of the four case studies is measured independently using the two measures and the overall completeness for each measure is obtained by taking the average values over the case studies.

To answer RQ2, we assess the effectiveness of generated test cases using mutation analysis. In our previous work [16], we defined various mutation operators for obligations. For experiments, we chose Context Management and Obligations Management types of mutation operators because they are used to mutate the behavior of the obligations. These mutation operators include: Context Extension (CE), Context Reduction (CR), Context Negation (CN), and Context Swap (C-Swap). The CE operator increases the scope of an obligation, CR reduces the scope of an obligation, and with CN the context is negated. The C-Swap operator is applied by swapping different contexts in an obligation. The number of mutants along with mutation operators is given in Table 6.

To answer RQ2, we assess the effectiveness of the generated test cases based on the resulting mutation score [18].

$$Mutation\ Score = \frac{Number\ of\ Mutants\ killed}{(Total\ Number\ of\ Mutants - Number\ of\ Equivalent\ Mutants)}$$

A mutant is said to be *killed* if at least one test case detects the fault seeded in the mutant program. A mutant is said to be equivalent if the seeded mutant does not change the behavior of the program (the mutant cannot be detected). We measure the mutation score for individual mutation operators in addition to measuring the overall mutation score, which is the average of all the individual mutation scores.

B. Execution of the Experiment

1) Modeling of the Case Studies: We modeled all the obligations of the four case studies using the *Obligations Profile*. As a result, one OCD and one OSM were developed per obligation, yielding a total of 47 OCDs and 47 OSMs. On average each class diagram has three classes, each state machine has four states and five transitions. The models are enriched with OCL constraints for two purposes: 1) Automated test data generation; 2) Automated test oracles.

2) Test Case Generation: We developed an MBT test case generation tool as discussed in Section V that takes OCD and OSM as inputs and generates executable test cases in Java. First, test paths are generated by applying the *All Transition coverage* [4] criterion in which each transition in an OSM is raversed at least once. In the second step, each test path is realized with two test data generation strategies: RD and ECP with Boundary Values [1]. Guards on the transitions of OSM are used as a basis for test data generation. State invariants in OSM are written in OCL and serve as test oracles. State invariants written in OCL are transformed into assertions in Java code using Dresden OCL [12].

3) Test Case Execution: Test case execution is performed for two case studies: HCS and VMS. For HCS, in order to execute test cases, the behaviors of some of the components such as ECG, Pulse Oximeter, Blood Pressure regulator, Camera, and Person are simulated in Kevore [17].

The test case execution for both case studies is carried out in the following steps: 1) Mutants are created manually by applying the mutation operators discussed earlier. The number of mutants applied is shown in Table 6. 2) Test cases with RD are executed on mutants and results are recorded; 3) Test cases with ECP and boundary values are executed on mutants and results are recorded. A mutant is killed if an assertion is violated during the execution of a test case. Such information is used to calculate mutation scores. Overall results of the experiments are summarized in Table 9.

C. Results

RQ1: The results of modeling nine obligations of MS, fifteen obligations of HCS, three obligations of VMS, and twenty obligations of Crisis Management System (CMS) [19] are summarized in Table 7.

The completeness of the *Obligations Profile* in terms of the number of obligations ($Completeness_{obj}$) is 100% for all four systems. The completeness of the *Obligations Profile* in terms of the number of applied stereotype types ($Completeness_{ster}$) is on average 75%. The highest number of types that was applied is 17 out of the 19 that were defined for the HCS as shown in Table 7. The description of stereotypes applied for different types of obligations is provided in Table 8.

Addressing RQ 1, our experiment suggests that our profile adequately captures all the obligations in four different systems, providing evidence that the profile is complete with respect to various concepts and stereotypes necessary for modeling obligations. It is noticeable in Table 8 that *Continuous* and *Post* obligations occur more rarely. On the contrary, the obligations of types *Persistent* and *Fulfillment* are the most common ones.

RQ2: To answer RQ2, we summarize the results in Table 9 based on the following information: number of obligations for each system, test data strategy (RD, ECP), number of test paths, number of test cases, mutation score for each mutant, and overall mutation score.

From Table 9, we can see that for HCS, with both RD and ECP, the mutation score is 100% for the CR type. For the CE

TABLE 6 NUMBER OF MUTANTS

Mutation Operator	# Instances
CR	7
CE	9
CN	6
C-Swap	22

TABLE 7 OBLIGATIONS PROFILE COMPLETENESS

System	# Ob	OC (%)	Overall OC (%)	US/TS	SC (%)	Overall SC (%)
MS	9	100	100	14/19	74	75
CMS	20	100		16/19	84	
HCS	15	100		17/19	89	
VMS	3	100		10/19	53	

* OC= Completeness$_{obj}$, SC= Completeness$_{ster}$ US=Unique Stereotypes, TS=Total Stereotypes, Ob=Obligations

TABLE 8 RESULTS OF MODELING OBLIGATIONS FROM DIFFERENT SYSTEMS*

Type	# Instances	Stereotypes Used
Fulfillment	8,9, 4,0	Obligation, Fulfillment, Initiation, Violation, Action, Context, Sanction, Human, System
Persistent	5, 1,1,0	Obligation, Persistent, Initiation, Violation, Action, Human, System
Pre	2, 0,1,1	Obligation, Pre, Initiation, Object, System
On	2, 0,1,1	Obligation, On, Initiation, Human
Negation	3, 1,1,0	Obligation, Negation, Initiation, Context, System
Post	0, 1,1,1	Post, Initiation
Continuous	0,2,0,0	Obligation, Continuous, Context

*In the # Instances column, w, x, y, z refers to number of obligations of a particular type in CMS (w), HCS (x), MS (y) and VMS (z), respectively.

and CN types, RD was not able to kill any mutants, whereas ECP killed 61% and 60% of mutants, respectively. For C-Swap, RD managed to kill 40%, whereas ECP managed to kill 75% of mutants. Overall, with RD and ECP we obtained mutation scores of 35% and 74%, respectively. For VMS, RD managed to obtain 80%, 0%, 75%, and 0% for CR, CE, CN, and C-Swap mutants, respectively. With ECP, we managed to obtain 100%, 40%, 85%, and 75% for CR, CE, CN, and C-Swap mutants, respectively. Overall with RD, we obtained a mutation score of 39% with RD and 75% with ECP.

From Table 9, we can observe an improvement in mutation scores when using ECP instead of RD, which is a more systematic test data strategy as compared to RD. One conclusion that we can draw from this observation is that roughly 75% (Table 9) of the faults were at the boundaries. The 25% remaining faults were probably located outside these boundaries. Selecting more random (or guided random) values outside the boundaries may further improve the mutation scores and additional test data strategies will be investigated in the future.

There were no equivalent mutants in our experiment and each of the applied mutants changed the behavior of the system in some way. However, as shown in Table 9, some mutation operators like CE generate hard-to-kill mutants, because they extend the condition to which an obligation applies. The C-Swap mutation operators are also not killed when a context is swapped with another context of larger range. As such, extending a timed-context would require finding a test that, instead of testing that the obligation is working correctly within the nominal boundaries, checks that the obligation is not working outside of these boundaries. Although we used boundary value analysis (ECP), it failed to kill most CE mutants since a larger set of test data outside the boundaries is necessary to kill them. From that viewpoint, CE mutants require test cases to go from being purely nominal/functional to being focused on robustness.

D. Discussion and Limitations

1) Modeling of Obligations: Though we have successfully applied most of the stereotypes from the *Obligations Profile* in all systems, our profile still has four main limitations that need to be addressed in the future. First, obligations may need to be enforced concurrently with other obligations and/or behavior. To model such concurrency, orthogonal states in UML2.0 state machines may be used, but we need to investigate this using an additional case study in the future. Second, in some situations, there may be an order in which two or more obligations must be enforced. We need to provide modeling support to capture such ordering between obligations. We will investigate if we can use UML activity diagrams to model such ordering among various obligations. Third, obligations may interact with each other's, implying that one obligation state machine interacts with other obligation states machines. We will investigate in the future how we can model such interaction between various obligations. Fourth, obligations crosscut functional system behavior, thus requiring a modeling approach based on aspect-orientation. One solution is to make use of UML2.0 Submachines together with aspect-orientation (e.g., AspectSM [20]) to model crosscutting obligations.

2) Testing of Obligations: Based on the results summarized in Table 9, we can see that ECP yields an overall mutation scores of 74% and 75% for the two case studies, respectively. We need to define more focused and sophisticated test strategies to further improve the effectiveness of obligation testing using the *Obligations Profile*. We will investigate the use of more advanced techniques such as constraint solving and search-based algorithms in our future work [13].

E. Threats to Validity

1) Modeling of Obligations: As with most empirical studies in software engineering, our main conclusion validity threat is related to the sample size on which we base our analysis. In our study, to assess the applicability and completeness of our profile, we modeled four different systems with diverse types of obligations.

Our main threat of construct validity is that we were not able to investigate all features of the *Obligations Profile* (e.g., all types of stereotypes), due to the nature of the case studies. We will model additional case studies in the future to investigate if the unused stereotypes are really needed to model obligations.

The modest size of our case studies can be seen as a threat to external validity. Notice that we used four different systems (one of them real) and modeled 47 different obligations. We believe that these four case studies provide initial evidence about the applicability of the approach that should be confirmed by additional larger case studies.

2) Testing of Obligations: One of the main construct validity threats is related to the selection of mutation operators to assess the effectiveness of the generated test cases. To alleviate it, we selected already published mutation operators [16] for obligations and mutated the case studies using those operators.Some of the mutation operators defined in [16] are not used in our study since they have no impact on the behavior of obligations and are specific to access control rules such as Entity Assignment, Rule Deletion and Hierarchal mutation

TABLE 9 TEST RESULTS*

System	# Obligations	Test Data Strategy	#Test Path	#Test Cases	MS (%)				OMS (%)
					CR	CE	CN	C-Swap	
HCS	5	RD	8	16	100	0	0	40	35
		ECP	8	37	100	61	60	75	74
VMS	3	RD	4	8	80	0	75	0	39
		ECP	4	21	100	40	85	75	75

*CR=CONTEXT REDUCTION, CE= CONTEXT EXTENSION, CN= CONTEXT NEGATION, C-SWAP= CONTEXT SWAP, MS= MUTATION SCORE, OMS= OVERALL MUTATION SCOR

operators. Moreover, we have tried to use all the mutation operators that were applicable on obligations, including context –based operators.

Another threat to the validity of testing obligations is related to the effects of experimental settings such as the types of obligations in both case studies, the number mutation operators selected, and the coverage criteria for test cases, which could have visible effects on fault detection effectiveness. The type of data coverage in test cases can impact the results in terms of mutants killed. We have improved data coverage from RD to ECP with boundary value analysis, however more sophisticated data coverage techniques could significantly impact results

With respect to conclusion validity threats, we based our results on two systems: HCS (a real system) and VMS. We have tried to cover different case studies in order to capture and test multiple types of obligations. However, the fact that only two systems could be tested has decreased the number of obligations types used for testing. To improve the confidence in the results, we intend to perform more thorough evaluation by defining more advanced testing strategies and additional case studies in the future

VII. RELATED WORK

The work on modeling obligations can be categorized into two main types. In the first type, obligations are modeled using different formalisms such as predicate logic [21], Lamport's temporal logic (TLA) [19], logic rules [5], and set theory [22]. The second type of works focuses on defining a dedicated language for obligations modeling such as xSPL [23], Obligations Specification Language (OSL) [24], Rei [25], and Ponder Specification Language [26]. Obligations are modeled for several purposes such as static analysis [5, 22, 27], platform for the enforcement of obligations policies with underlying system [21, 23], state-based modeling for verification [28], and software testing [29, 30]. There are also works on using UML to model access control infrastructure that does not include the modeling of obligations [31, 32]

Some contributions in the literature have focused on the model-based testing of obligations. They model obligations using different formalisms such as Extended Finite State Machines (EFSM) [29] and Petri Nets (PrT) [30]. In the work of Mallouli and Cavalli [29], EFSMs are used to model access rules and obligations with the objective to identify test objectives. However, they do not specify constraints on obligations, in particular timing constraints. Xu et al. [30] use PrT as test model, generate test inputs, and use them for automated test generation [33]. They consider timing constraints as well as negative obligations. Elrakaiby et al. [16] define mutant operators for mutation analysis of obligations. A number of works in the literature focus on testing access control rules [34-36] but do not include obligations.

Our work on modeling obligations, using the *Obligations Profile*, makes a number of contributions when compared to the above literature. First, it provides a solution based on UML, thus making it possible to use UML commercial and open source tools. This in turn is likely to increase adoption by practitioners. Moreover, to the best of our knowledge, no existing work provides a systematic and comprehensive modeling methodology to support automated testing.

VIII. CONCLUSION

Model-based testing (MBT) is increasingly used for automated testing as it is supported by a number of open source and commercial tools. However, MBT relies on behavior descriptions that must be at a sufficient level of detail to enable test automation. Obligations are mandatory actions that guarantee the security of the system if implemented correctly. To support MBT for obligations, a dedicated and systematic modeling approach is required. This paper addresses this challenge by proposing the *Obligations Profile*, extending the concepts of UML Class and State Machine Diagrams. We defined constraints on the *Obligations Profile* using the Object Constraint Language (OCL), which can be enforced by UML modeling tools and guide the user in following a systematic methodology for modeling obligations. We assess the completeness and applicability of the profile by modeling 47 different obligations for four different systems. The results showed that our profile is sufficiently complete to model the various types of obligations present in our case studies, though some stereotypes defined in the profile remained unused.

Since the main target of the *Obligations Profile* is automated testing, we provide the mapping of its concepts to testing concepts that are necessary for automated test case generation. We developed a prototype tool that takes in input UML class diagrams and UML state machines, on which the *Obligations Profile* is applied, and that generates executable test cases. To assess the effectiveness of the generated test cases, we performed mutation analysis and results showed that on average the generated test cases managed to kill 75% of the mutants. Though encouraging, the results suggest to define more focused test strategies to further improve fault detection effectiveness.

In the future, we plan to define more sophisticated test strategies by combining search-based software engineering and constraint solving to further improve fault detection effectiveness. Moreover, we plan to integrate the use of existing MBT tools such as TRUST [6] or Smartesting [7] to support the application of the *Obligations Profile* for test case generation.

References

1. R. Binder,Testing Object-oriented Software Testing: Models, Patterns, and Tools, 1st, Addison-Wesley Professional: 2000,

2. M. Utting, and B. Legeard,Practical model-based testing: a tools approach, Morgan Kaufmann Publishers Inc: San Francisco, 2007,

3. Object Constraint Language. 2014. Available at: http://www.omg.org/spec/OCL/

4. S. Ali, H. Hemmati, N.E. Holt., E. Arisholm, and L. Briand, Model Transformations as a Strategy to Automate Model-Based Testing-A Tool and Industrial Case Studies, 2010

5. Y. Elrakaiby, F. Cuppens, and N. Cuppens-Boulahia: 'Formal enforcement and management of obligation policies', Data & Knowledge Engineering, 71, (1), pp. 127-147, 2012

6. F.c. Lagarde, H. Espinoza, F.c. Terrier, C. André, and S. Gérard: 'Leveraging patterns on domain models to improve UML profile definition' in: 'Fundamental Approaches to Software Engineering'Springer2008, pp. 116-130

7. T. Weilkiens,Systems engineering with SysML/UML: modeling, analysis, design, Morgan Kaufmann OMG Press: 2011,

8. O.M. Group. Modeling and Analysis of Real-time Embedded Systems Available at: http://www.omg.org/spec/MARTE/1.0/

9. P. Baker,Model-driven testing: Using the UML testing profile, Springer-Verlag: 2009,

10. J. Park, and R. Sandhu: 'The UCON ABC usage control model', ACM Transactions on Information and System Security (TISSEC), 7, (1), pp. 128-174, 2004

11. Smartesting Software. Available at: http://www.smartesting.com/index.php/cms/en/home

12. Dresden OCL Software. Available at: http://www.dresden-ocl.org/index.php/DresdenOCL

13. S. Ali, M. Iqbal, A. Arcuri, and L. Briand: 'Generating Test Data from OCL Constraints with Search Techniques', IEEE Transactions on Software Engineering, 39, (10), pp. 1376 - 1402 2013

14. J. Kienzle, N. Guelfi, and S. Mustafiz: 'Crisis management systems: a case study for aspect-oriented modeling' in: 'Transactions on aspect-oriented software development VII'Springer Berlin Heidelberg2010, pp. 1-22

15. Convivialty and Privacy in Ambient Intelligence Systems - CoPAInS. Available at: http://wwwen.uni.lu/snt/research/serval/projects/copains

16. Y. Elrakaiby, T. Mouelhi, and Y. Le Traon: 'Testing obligation policy enforcement using mutation analysis'. in. *IEEE Fifth International Conference on Software Testing, Verification and Validation*, Montreal, 2012 2012 pp. 673-680

17. Kevoree. Available at: http://kevoree.org/

18. A.J. Offutt, G. Rothermel, and C. Zapf: 'An experimental evaluation of selective mutation'. in. *15th international conference on Software Engineering*, 1993 pp. 100-107

19. X. Zhang, F. Parisi-Presicce, R. Sandhu, and J. Park: 'Formal model and policy specification of usage control', ACM Transactions on Information and System Security (TISSEC), 8, (4), pp. 351-387, 2005

20. S.A. ., L. Briand., and H. Hemmati.: 'Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems', Software & Systems Modeling 11, (4), pp. 633-670 2011

21. C. Bettini, S. Jajodia, X.S. Wang, and D. Wijesekera: 'Provisions and obligations in policy management and security applications'. in. *Proceedings of the 28th international conference on Very Large Data Bases* 2002 2002 pp. 502-513

22. K. Irwin, T. Yu, and W.H. Winsborough: 'On the modeling and analysis of obligations'. in. *Proceedings of the 13th ACM conference on Computer and communications security* New York, 2006 2006 pp. 134-143

23. P. Gama, and P. Ferreira: 'Obligation policies: An enforcement platform'. in. *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks*, Lisboa, Portugal, 2005 2005 pp. 203-212

24. M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter: 'A policy language for distributed usage control' in: 'Computer Security–ESORICS 2007 Lecture Notes in Computer Science 'Springer Berlin Heidelberg2007, pp. 531-546

25. L. Kagal, T. Finin, and A. Joshi: 'A policy language for a pervasive computing environment'. in. *Proceedings. POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, 2003 2003 pp. 63-74

26. N. Damianou, N. Dulay, E. Lupu, and M. Sloman: 'The ponder policy specification language' in: 'Policies for Distributed Systems and Networks'Springer Berlin Heidelberg2001, pp. 18-38

27. H. Ferrier-Belhaouari, P. Konopacki, R. Laleau, and M. Frappier: 'A Design by Contract Approach to Verify Access Control Policies'. in. *17th International Conference on Engineering of Complex Computer Systems* Paris, 2012 2012 pp. 263-272

28. D.J. Dougherty, K. Fisler, and S. Krishnamurthi: 'Obligations and their interaction with programs' in: 'Computer Security–ESORICS 2007'Springer Berlin Heidelberg2007, pp. 375-389

29. W. Mallouli, and A. Cavalli: 'Testing security rules with decomposable activities'. in. *10th IEEE High Assurance Systems Engineering Symposium*, Plano, TX 2007 2007 pp. 149-155

30. D. Xu, M. Sanford, Z. Liu, S. Johnson, M. Emry, B. Brockmueller, and M. To: 'Testing access control and obligation policies'. in. *International Conference on Computing, Networking and Communications (ICNC)*, San Diego, 2013 2013 pp. 540-544

31. D. Basin, J. Doser, rgen, and T. Lodderstedt: 'Model driven security: From UML models to access control infrastructures', ACM Trans. Softw. Eng. Methodol., 15, (1), pp. 39-91, 2006

32. J. Jürjens: 'UMLsec: Extending UML for secure systems development' in: '≪ UML≫ 2002—The Unified Modeling Language'Springer2002, pp. 412-425

33. D. Xu: 'A tool for automated test code generation from high-level Petri nets' in: 'Applications and Theory of Petri Nets'Springer Berlin Heidelberg2011, pp. 308-317

34. E. Martin, and T. Xie: 'Automated test generation for access control policies via change-impact analysis'. in. *Third International Workshop on Software Engineering for Secure Systems*, Minneapolis, MN 2007 2007 pp. 5

35. A. Pretschner, T. Mouelhi, and Y. Le Traon: 'Model-based tests for access control policies'. in. *1st International Conference on Software Testing, Verification, and Validation*, Lillehammer 2008 2008 pp. 338-347

36. Y.L. Traon, T. Mouelhi, and B. Baudry: 'Testing security policies: going beyond functional testing'. in. *The 18th IEEE International Symposium on Software Reliability*, Trollhattan, 2007 2007 pp. 93-102