Managing Break-The-Glass using Situation-Oriented Authorizations

Bashar Kabbani (bashar.kabbani@irit.fr)*
Romain Laborde (romain.laborde@irit.fr)*
Francois Barrere (barrere@irit.fr)*
Abdelmalek Benzekri (abdelmalek.benzekri@irit.fr)*

Abstract: The patient's life is a redline in Healthcare environments. Whenever it comes to danger, such environments reject static authorizations . A common problem "Break The Glass" is known as the act of breaking the static authorization in order to reach the required permission. Healthcare environment is full of different contexts and situations that require the authorizations to be dynamic. Dynamic Authorization is a concept of giving the choice to E-Health authorization system to choose the most suitable permission by considering one's situation. This paper aims at preventing the matter of modifying the policy to make authorizations dynamic. It introduces a simple solution to provide Dynamic Authorization by orienting the authorization system decision using situations. Situations, which are calculated using Complex Event Processing, are integrated to XACML architecture. A Healthcare example proves the efficiency of our approach.

Keywords: Dynamic Authorization, Access Control, Policy-Based Management, Attribute-Based, Break The Glass, Situation-Awareness, Complex Event Processing

1 Introduction

Healthcare domain is full of emergency cases. Sometimes doctors are limited to access their patients' private information only whereas they can access any other patient private information when it is a life or death issue. As consequence, doctors' authorizations must be dynamic to handle these life or death scenarios. In this domain, dynamic authorization is the matter of prohibiting access for doctors to access patient's information (PI) and giving them rights in exceptional conditions and circumstances, e.g. emergencies. At the opposite, it is concerned about giving the right for doctors to access patient's information and take this right back when conditions and circumstances are no more met. This problem is known as Break-The-Glass. However, the question is "How to bestow such dynamicity?"

Authorizations are usually expressed within authorization policies. A policy is a set of rules that are evaluated when a doctor requests an access to a protected information system. Based on this evaluation, the decision-making process returns an authorization decision that represents the actual doctors' authorization. E-health applications can get exceptional situations once a year; where patient's life is in danger and no authorized doctor can save the patient's life. Two approaches can be applied to setup a solution for the "Break-The-Glass" problem depending on the capabilities of the decision making process

* University of Paul Sabatier (UPS), Institute of Research in Informatics at Toulouse (IRIT), 118 Route de Narbonne, 31062 Toulouse, France

and the expressiveness of the authorization policy language. The first approach deals with static authorizations policies. We call static authorization policies, policies whose evaluation will always return the same result (permit or deny) for a given request whenever this request is made. In this case, making authorization dynamic requires modifying the policy. For instance, if we want that the eHealth authorization system allows only responsible doctors to access their concerned patients' information during working days (policy1: if *Doctor ID = Philippe* and he is a non-responsible doctor of the *patient Joe* then ***deny***, i.e. the normal authorization is to deny non-responsible doctors). On Emergencies, non-responsible doctors will use an external system to break this rule by a new rule (e.g., policy2: if *Doctor ID = Philippe* then ***permit***). This new rule will then be replaced by the first rule when the emergency situation ends. The advantage of this approach is that it works with existing systems. It gives a real-time activity to break the policy and save the patient's life. However, the drawback is the rule management complexity when considering many circumstances and many rules. Keeping loaded rules free of conflict is a big challenge. In addition, analyzing such dynamic authorizations is much more complicated because it requires knowing what rules are loaded. Also, giving doctors over responsibly to manage his authorizations. As a consequence, employing such an approach for dynamic authorization seems complicated.

Another approach consists in building authorization decision-making systems aware of these circumstances in order to evaluate dynamic authorization policies. This approach requires authorization policy languages be able to express circumstances based authorizations. In this case, authorization policies do not need to be replaced. For instance, our "new" eHealth authorization policy may consist of two rules:

Rule1: When patient is in normal situation and *Doctor ID = Philippe* and a non-responsible doctor of the *patient Joe* then deny.

Rule2: When patient is in danger and *Doctor ID = Philippe* then permit.

In this case, the policy is static as it doesn't change but authorizations are dynamic. The drawback of this approach is that it requires the authorization decision-making process to be aware of emergency situations. However, the benefits are important since policies are easier to understand and analyze because they are closer to the security requirements and they do not change.

In this article, we follow the second approach. However, circumstances must not be limited to patients only. Current systems provide various sensors such as presence detection, temperature, network events, security events, etc. As a consequence, circumstances to consider in authorization policies are related to information coming from these sensors. We propose to formalize circumstances through the concept of *situation*. Situations are remarkable conditions and circumstances reflecting abstract semantics to describe past, present or future behavior of entities. A more technical definition is *"A situation is a relevant time frame calculated based on events generated by available sensors"* [AE03].

We propose to implement situations based authorization policies using XACML that provides a policy language based on attributes and a policy based management architecture. Using XACML, we represent situations as attributes that aggregate rules and give through their values dynamicity to the policy. By changing the value of situations, the choice for an appropriate decision will be made after evaluating associated rules. Finally,

we present how to make use of Complex Event Processing (CEP) to realize the identification and calculation of situations. In order to prove the efficiency of our architecture, we develop our idea through an example taken from the healthcare domain known as "break the glass" that consists in managing authorization of doctors confronting life critical issues.

The remainder of this paper is structured as follow. Section II presents the concept of "Break The Glass" together with a scenario. In Section III, we present a study on related works. Section IV is dedicated to present the situation management. Orienting the authorization policy using situations is presented in Section V. Section VI details our implementation. Finally, we conclude in Section VII.

2 Break The Glass Scenario

In Healthcare, "Break The Glass" (BTG) is a very good example that treats the dynamic authorization problem through bypassing traditional authorizations. In emergency circumstances, unauthorized doctors can break the policy to get rights to access information that they could never have in an ordinary case. BTG is about giving entities administrative rights to break in the policy and modify rules when it comes to patients' life.

Our scenario:

Emma is a doctor in a modern hospital. Patients' Information is not an Open Access for doctors. The intervention should be justified by a reason, e.g. treatments. Therefore, only the treating or the responsible doctors are allowed to access their patients' information. One day, sensors connected to a patient, Joe, showed an urgent need for a doctor. The problem is that his responsible doctor is not available. Emma was the only one available, ready and able to act. Unfortunately, Joe is not one of her patient that mean she would not have access to his information. The BTG solution gives Emma the right in emergency situations to break the general policy to save Joe's life.

Breaking the general policy is required to change the authorization. To change rules (break the policy), one should have administrative permissions to do so (In the worst case Emma can change the current policy). Emma will break the policy and give herself authorization to access Joe's files.

It is important to highlight the importance of this example for our paper. The objective of this contribution is to avoid Emma from having administrative permissions, which does not belong to her role as a doctor. All what Emma should concern about is Joe's life.

3 Related Work

Works on dynamic authorization can be divided into mainly two main axes. On the first hand, modify the policy and considers it as an electronic board where plugs (parts of the policy) are replaced frequently. Dynamic Access Control (AC) Decisions by Junzhe HU, extended the RBAC model to keep traditional access control methodology [Eys01], [HW04]. Adaptive AC Decisions takes in account different states of the subject, e.g. requesting and waiting, in the AC decisions, T. SANS [SCCB06]. It is, however, more oriented towards dynamic enforcement rather than authorization. In access control management, "Break The Glass" [SSH+ 08] is a common use case to consider when evaluating policies to provide dynamic authorization. However, related works are oriented towards breaking the policy and involve doctors in the administration role. We also point out papers working on leveraging the privacy of healthcare information in case of emergencies

such as: Carminati et al. [CFG11] and Ferreira et al. [FACC10]. To the best of our knowledge, the use of dynamic authorization for the BTG use case has presented only through modifying the policy in: Schefer-Wenz et al. [SWS13], Marinovic et al. [MCMD11] and Brucker et al. [BP09].

On the other hand, energizing the decision engine to provide dynamicity in its authorization mechanism and consider the policy as a static holy reference. Laborde et al. [LKBB07] initiated a work where the dynamicity is presented as changes of permissions based on a static policy. The changes are made after the assignment of a new role within the Role Workflow. AC Authorization Oriented has been also studied by a Workflow, Ma et al. [MWZL12] and Ferreira et al. [FACC10].

In conclusion of this section, dynamic authorization is about energizing the authorization mechanism to be changeable according to new factors, i.e. the definition of “dynamic”. The research trends are focused only on having the “Dynamic Authorization” as an outcome of modifying the policy (change, add and remove rules). None of them worked on providing the “Dynamic Authorization” as an income to the policy without changing it.

4 Situation-Oriented Approach

Emma being in charge for a patient who is not under her responsibility is one result of having the emergency situation generated by the alarming system. Given this way of reasoning, we see that the ***situation*** itself orients the authorization decision towards giving Emma the right to access the patient’s information, the one under urgency. All events arriving try to explain the abnormal situation of the patient and that his/her doctor is not available. The situation to deal with by the eHealth system is the immediate need for a doctor. It is important then to define *situations* that orient the system’s behavior and identify/detect them.

4.1 Situation Definition and Identification

The Collins Dictionary defines situation as *“A set of conditions and circumstances in which one finds oneself”*. Pervasive Systems are context-aware systems that concern about future predictions. In such systems, situation is: *“A set of contexts in the application over a period of time that affects the future system’s behavior”*. A context is any instantaneous, detectable, and relevant property of the environment, system, or user, e.g. location, available bandwidth and user’s schedule. Pervasive Systems collect data to predict or anticipate situations [YDM12]. This article is only interested about situations once they appear. Therefore, we refine the definition to be *“a set of contexts detected upon matching predefined conditions and circumstances within the application and over a timeline that affects the current and future system’s behavior”*.

Giving the BTG example, we can link the provided definition by determining 1) the conditions, i.e. having no responsible doctor available who can access normally or legally patients’ information in order for treatment, 2) and circumstances, i.e. having many readings from sensors telling how bad the status of the patient is, in which the healthcare authorization system found itself in need of Emma.

Identifying a situation’s occurrence depends on three main elements: Events, Composite Events and Complex Events. All previous elements should be aware of *context* in order to have sense. Events are instantaneous activities or phenomena. For instance, sensors

are events sources that send readings about the instantaneous patient's pulse. Events also could be doctors activates like scheduled appointments. Composite events consist of a collection of events that satisfy some patterns. Patterns are predefined queries that contain events, composite events and complex events in a query language, i.e. similar to SQL. Complex events (CEs) are abstraction of events. Composite events and events are aggregated, filtered and correlated using logical operations to express a meaningful phenomenon (complex event). CE writers predefine the meaning expected. For instance, the fact that the patient got symptoms, e.g. a heart attack, is a complex event generated based on 1) readings about the patient's heartbeat and 2) composite events (a worning alarm aggregated with nurses calls). There comes the *situation's* starting-point (SP). "Urgent need for a doctor" is a situation starts with the detection of complex events like a heart attack, nerves crises and other of what mean that the patient is in danger and needs a doctor urgently. Plus, another event indicates that the responsible doctor is not available. Figure 1 is a general picture on how to identify a situation, where "Urgent need for a doctor" is an example of individual situation (the patient).

Once the situation starts, it orients the authorizations decisions towards suitable permissions until and the situation ends. The ending-point (EP) of a situation is similar to its SP in terms of participants. Complex Events ends the situation as well, e.g. the doctor has sent a notification to validate the end of the treatment and devices confirmed through reading the status of the patient.

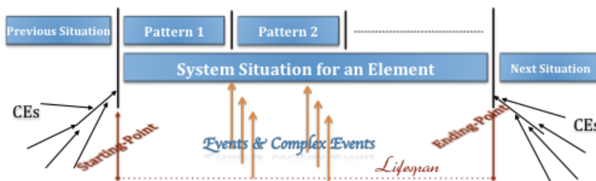

Fig. 1: Identification of Situations

During the situation life (lifespan), it is useful to trace the status of a situation in terms of Situation Management. Therefore, patterns exist as a mechanism similar to the "polling" that keeps the system informed about the situation's status. Once the situation is identified, the patterns could be used to make sure that the entity still in the same situation (did not evolve or vanish). For instance, the situation of the emergency room needs to be traced, as it is a critical and demanded room.

4.2 Situation Management

Complex Event Processing presents tools *off-the-shelf* that accomplish several operations to process events: Correlation, Filtering, Aggregation, Monitoring, Generation and so on. The main objective is to enable on the one hand developers to customize CEP processing engines to meet business needs and on the other hand users, e.g. managers, to express there queries, i.e. events, complex events and situations. Nevertheless, Adi et al. demonstrated in Amit – The Situation Manager [AE03] that CEP engines are able to manage concrete situations of the system as well.

The Situation Management is concerned about two main operations: determining the lifespan of a situation and managing the situation during its life (lifecycle). The Lifespan

is the temporal context during which the situation detection is relevant. The lifespan is an interval bounded by starting and ending point. An occurrence of the SP initiates the situation life and an occurrence of EP terminates it. Both occurrences should be defined and initiated previously. SP and EP values are assigned automatically by the detection of patterns or manually by administrative requests. For instance, the emergency situation ends when doctors finish the patient treatment. However, the start of a situation could be automated as the example shows in Section II.

Two situations may have the same starting and/or ending dates (lifespan), but not necessary the same lifecycle. Moreover, the semantic of each situation is different from one to another. Each situation is coupled with the elements participating in the situation creation like: Subjects (Users), Resources, Actions and Location (Environment). Using the semantic and the elements concerned about a situation we can distinguish it from others. For instance, two situations could have the same time but could not concern the same entity, e.g. a patient could not be in normal and abnormal situations in the same time. However, the system could be in the normal situation regarding the patient Y and behave in abnormal, e.g. emergency, situation for the patient Joe.

The situations lifecycle is a story that the CEP engine tells while capturing situations' patterns, i.e. during the lifespan. For instance, the emergency situation of Joe started after capturing readings telling that he is in danger. Using the CEP engine, E-Health application recognized that there is no one else in the room. Emma is informed about the situation. At the same time, the system knows about the need for an unauthorized doctor. She does not have access, so she asks for breaking the policy. Then, she gets access to the PI and starts the preparing of a treatment. Emma sends a confirmation to the eHealth system that the treatment is done and that situation is terminated (Joe is no more in danger).

Many kinds of situation could be defined and attached with many entities and for several contexts. The complexity that this term could reach invited us **not to mix the management of situations with the policy itself,** but to dedicate a situation management paradigm apart from the authorization policy one. Situations types could be degraded in terms of priority, e.g. Emergency, Urgency, Recommended, Demanded, etc. They could be extended in terms of surface as well, e.g. for a fire in a location: Partial, Minimal, Grand Scale, etc. The context can really play with situations. Finally, each situation is coupled with a subject concerns and the related behavior, see the definition. For example, *"a patient is in a dangerous situation"* here the situation concerns the patient who is not a user, a resource, an action nor an environment. Therefore, from the authorization system point of view, situations are decoupled from the access control requests.

5 Authorization Situation-Oriented Policy

XACML is a generic, flexible and abstract language with an architecture that could express/enforce our expected policy. In this section, we present how we handle situation based authorization policies using XACML.

5.1 eXtensible Access Control Markup Language

XACML is an XML-based language for access control that has been standardized by OASIS. The XACML policy language describes general access control requirements in term of constraints on attributes, where an attribute could be any characteristic of any *Security Related Object (SRO)* on which the access request is made. XACML V3 [XAC13] is not limited anymore to the version 2 elements (subject, resource, action and environment). Attributes are manipulated through predefined data types and functions. Considering attributes makes the language very flexible. Moreover, the XACML language is natively extensible (new attributes, new functions or new data types). We employ XACML V3 by orienting the security policy rules using situations attributes. The most two important features presented by this version are Categories and Aggregation. The main idea of aggregation is to use abstraction (being close to requirements) to express groups of rules and situations.

The aggregation is possible using the new fashion of target section in V3, becomes independent of the four tuples (Subject, Resource, Action and Environment). Thanks to the XACML V3 *category* property, it is possible to target the policy to new attribute categories, such as *situations*. So, we define the category filled within the target section of each XACML policy to be attached to situation attribute. Therefore, the value of situations will target the Policy Decision Point (PDP) towards the correct set of rules to be evaluated. As a result, rules were aggregated by situations. XACML provides also a management architecture that describes the different entities and their roles related to the Decision-Making process (Figure 2).

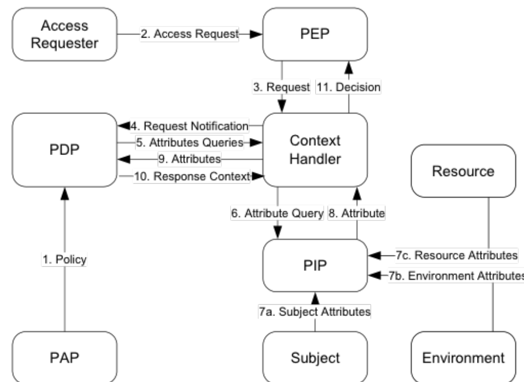

Fig. 2: Simplified XACML data flow model

Policy Administration Points (PAP) write policies and make them available to the PDP (step1). An access requester sends an access request to the Policy Enforcement Point (PEP) (step2), and the PEP forwards it to the context handler (step 3). The context handler constructs a standard XACML request context and sends it to the PDP (step 4). The PDP can request any additional subject, resource, action and environment attributes from the context handler (step 5). The context handler requests attributes from a Policy Information Point (PIP) (step 6). The PIP obtains requested attributes and returns them to the context handler (step 7, 8). The context handler sends requested

attributes. The PDP evaluates the policy and returns standard XACML response context (with an authorization decision) to the context handler (step 9, 10). Finally, the context handler returns the response to the PEP that enforces the PDP's decision (step 11).

5.2 XACML Employment

For the next sections, we enhance the previous BTG scenario to be as follows: Patients are observed through many sensors that send information about: *Fever, Pulse, Blood Pressure, Conscience Status and numerous bio-medical signals. These sensors are linked to a Symptoms Diagnosis System (SDS). Contextual Sensors (CSs) are connected to capture physical movements and positions: Room Occupancy, Patient's Position and Doctors' Position. SDS and CS are connected to an Alarm System. Once an alarm is launched expressing an emergency situation, a notification message is sent to the doctor in charge of such situation for the concerned patient.*

Giving the BTG scenario, three situations related to authorization appear (Figure 3). Initially, the healthcare authorization system evaluates the access requests to PI based on the static policy oriented by a normal situation, i.e. when all systems elements are in normal situations. Within these situations, Emma is not permitted to do any of the following actions: accessing Joe's PI, BTG request to Joe's PI and ending the BTG request. When Joe's health is in danger and no responsible doctor nearby to save him is found around, Joe will be in a situation named "Urgent need for a Doctor". As Emma is the only available doctor, she will receive a notification to take in charge the treatment of Joe. Emma cannot access Joe's PI directly, so she will present a BTG request to Joe's PI. Logically, she won't have permission to end the BTG request, as it is not placed yet. Once the glass is broken "BTG Requested", i.e. *the situation of the PI*, Emma can access Joe's PI. However, she cannot place another request to BTG, as it is already broken. Once Emma finishes treating Joe, she can end the BTG request. Then, the situation of Joe will be back to normal and the cycle is completed.

Rule 1: IF *Situation (Owner (Resource)) = Normal ∧ Type (Resource)=PI∧ Role (Subject) = Doctor ∧ Type (Action)=Access ∧ Owner (Resource) ∈ PRPL (Resource) ➔ Permit*
Rule2: IF *Situation (Owner (Resource)) = Doctor In Need ∧ Role (Subject) = Doctor ∧ Type (Resource) =PI ∧ Type (Action) = BTG Request ➔ Permit*
Rule3: IF *Situation (Resource) = BTG Granted ∧ Role (Subject) = Doctor ∧ Type (Resource)=PI ∧ ID (Subject) = BTG Requester ∧ Type (Action) =Access ➔ Permit*
Rule4: IF *Situation (Resource) = BTG Granted ∧Role (Subject) = Doctor ∧ Type (Resource) =PI ∧ID (Subject) = BTG Requester ∧ Type (Action)=End BTG ➔ Permit*
Rule5: *Default ➔ Deny*

By considering these three situations, the policy expressing the dynamic authorization can be represented by the **five rules** above. The *first rule* states: if the doctor requesting access to Patient Information (PI) is one of the persons responsible of the patient who owns this information. PRPL is the List of Persons Responsible who has access to the Patient's information (with the patient himself). In order for Emma to be able to place a request to break the glass (BTG Request), the system should be in situation that needs an external doctor ("Urgent need for a Doctor"). In this case only, she can break the policy using the *second rule*. The *third rule* is to ensure that only doctors who placed a BTG request can access the Patient Information. The objective of this rule is to avoid other doctors taking advantage of such situation. The *fourth rule* is to let Emma end the BTG process or request on Joe's PI once she finishes the treatment. Finally, to manage conflicts

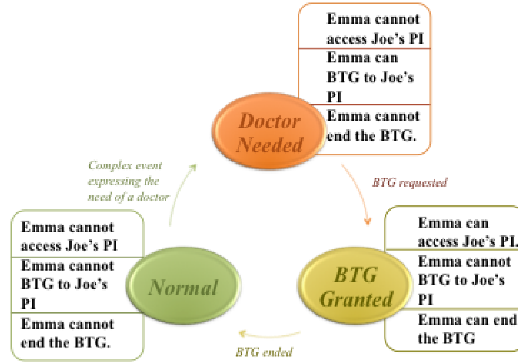

Fig. 3: BTG Situation Cycle

we declare a default *fifth rule* that denies all other access requests.

6 Prototyping Example

We have implemented our BTG example to prove the concept of Dynamic Authorization using Situation Orientation approach. We required from this prototype to 1) Collect Events from sources 2) Analyze and Process Events 3) define patterns to detect Complex Events 4) translate complex events into situations and store them in a database 5) use situations as values for the XACML situation attribute 6) respond to access requests based on the provided situations. The solution expects as a result from this prototype to have different decisions for the same access request, but in different situations, i.e. different contexts as well.

Players of our prototype are mainly the Complex Event Processor, all the elements of the XACML Architecture (PAP, PDP, PEP and PIP), the XACML Policy and a database to store the situations. We have implemented our situation manager using the complex event processor ESPER [Esp06] and the XACML V3 implementation that we employed can be found in [Bal12].

The ESPER engine will detect situations and monitor patients and doctors activities. The engine will react on situations by updating the values in database, notifying the users and regenerating more meaningful events, e.g. after receiving two events saying that Joe is complaining, generate an event to the eHealth application informing the non-satisfaction of the patient.

We have defined a set of ten types of events with ESPER. Events Types are: E1) Patient's Fever, E2) Patient's Status, E3) Patient's Pulse, E4) Patient's Position, E5) Room Occupancy, E6) Doctor's Position, E7) BTG Request, E8) Alarm Detection, E9) Access Request, E10) Situation Detection. Each event type should have at least values describing one element of the system (Subject, Resource, Action and Environment).

The instances of these event types are generated and structured in event streams using ESPER Events Simulator (ESPER-ES). ESPER-ES is a mechanism of event streams generation that also keeps trace inside text-like files. Event Sources are simulating sensors of the Patient Care Monitor (PCM). PCM sends for example readings about the fever

of the patient and his physiological status, etc. An example of event type sent by the PCM is the Patient's Fever (E1). The information contains for each reading (i.e. event): Patient ID, PCM Sensor ID, Sensor Readings and a Timestamp. For instance, an event occurs saying that Joe has fever degree reading of (40° C) from the PCM sensor PCM2340 at timestamp 130915221345, date followed by hour. The other sensor to simulate is the Movement Detection (DM) that captures Positions of subjects. So, events represent either readings of sensors or calculated values from sensors' readings such as the number of persons occupying a room. Access and BTG Requests are captured from the network once they are sent to the PEP.

There are three examples of Complex Event types or patterns that aggregate events from the ten event types mentioned earlier. In order for the complex event to be triggered, it should meet the conditions of a query that represent either a pattern, as the sliding window [MZZ12], or a simple SQL-Like query:

CE1 – **Patient In Danger:** the Fever should be high, the status of the Patient is claiming and there is no one responsible near her/his.

CE2 – **Responsible Doctor Unavailable:** the e-health application should contact the responsible Doctor or simply check her/his Schedule to see the availability. Also, the CEP engine can check the position of the doctor if not in the hospital.

CE3 – **Urgent need for a Doctor:** a threshold is assigned to the number of persons inside the room, e.g zero. Also, the CE1 and CE2 should have been triggered.

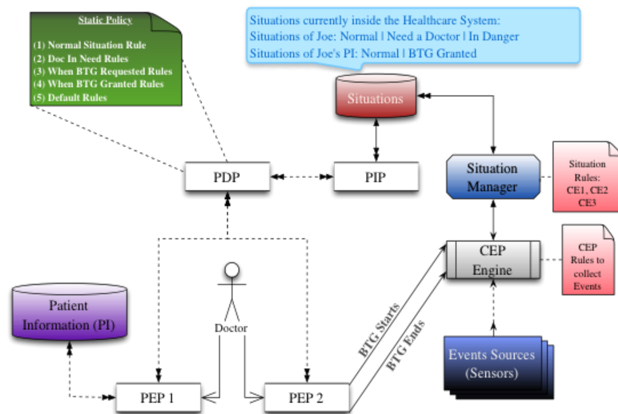

Fig. 4: Prototype Technical Architecture

The prototype's architecture, illustrated in Figure 4, explains the different elements participating in the solution and gives an idea on the sequence or the flow of actions to encounter. First of all, the policy writer should use the PAP to write the mentioned XACML Policy, see XACML Employment. The PAP will then inject the written policy in a repository to be used by the PDP. At the same time, the situation manager should write the CEP rules that should govern the CEP engine behavior. Once configured, the CEP engine should start monitoring the systems' behavior and evaluate the activities based on the provided rules.

The simulation process starts by using ESPER-ES that represents the event sources. The prototype identifies three scenarios: there is no situation concerning Joe, Joe is urgently needs a doctor and BTG granted on Joe's PI. First, a doctor places an access requests to Joe's PI. The XACML architecture will deal with this request as explained earlier in Section V (B). We assume that the first scenario is when the PIP gets the value of the situation attribute as ***empty*** (normal situation). Based on the doctor permissions, the decision will be a traditional authorization loop to permit or deny the access.

We configured the ESPER-ES to generate events from the provided event types. ESPER-ES creates six streams to be generated with 20 events in each. The 120 events are from the following six types: Fever, Pulse, Status, Patient Position, Doctor Position and Room Occupancy. During the detection of events, the CEP engine composes events and listens to both access and BTG requests going to the PEP. Once the CE1 detected by the CEP engine, it forwards an alarm to the Situation Manger that understands that a Patient (Joe) is in Danger. The Situation Manager stores this situation in the Database. In case of the unavailability of responsible doctors for Joe, the CEP engine generates the CE2. The CEP engine directly detects the CE3 after ensuring that no one is involved yet by checking room occupancy. The situation manager stores both situations as well in the database, i.e. "Urgent need for a Doctor" and "Doctors Unavailability". Emma was available to treat Joe and she knows the rules; Joe is not her patient hence she needs a specific authorization. Therefore, Emma places directly a BTG request to the authorization system, i.e. PEP 2. The traditional loop starts, but this time it is "Urgent need for a Doctor". Emma will be permitted to break the glass. So, the PDP sends the response to PEP 2 letting him inform Emma about the decision. At the same time, the PEP 2 enforces the decision by starting the new BTG situation. The CEP engine detects that and informs the Situation Manager who stores this situation in the database. Emma now can access to Joe's PI by a request to the PEP 1. The PDP allows Emma to access and PEP 1 enforces this decision.

Finally, the CEP engine detects that Joe's information is now broken by a BTG request. Once Emma finishes the treatment, she asks PEP 2 to end the BTG situation on Joe's information. The PDP allows Emma to do so as she is the one who placed the BTG request in the first place. As we said earlier, this information is gathered by the PIP about the identity of Emma and the associated situations related to her person and to her patients. PEP 2 enforces the decision by sending an event to the CEP engine to end the BTG situation. The CEP engine informs the Situation Manager who removes all associated situations ("Urgent need for a Doctor", Joe in Danger and BTG Situation). The system goes back to treat Emma and Joe normally.

We demonstrated using a prototype how it is possible and simple to provide dynamic authorization without modifying the policy. Based on a static policy, the prototype aims to manage *Break-The-Glass* using situation-oriented authorizations.

7 Conclusion & Future Work

Modern systems and usages require dynamic authorizations. We have presented in this article an approach to specifying and enforcing dynamic authorization policies based on situations. This approach allows simplifying the task of writing and analyzing dynamic authorizations. Although policies are static, as rules do not change, authorizations are

dynamic. Situations are relevant time frames calculated using complex events processing. We have chosen XACML because it provides a language where any security information can be represented by attributes. In addition, the modularity of XACML architecture facilitates its integration for enforcing situation based authorization policies. We proved that our approach enforces dynamic authorization by implementing "Break Glass" scenario.

We believe considering situations during authorization policies definition is the appropriate approach. Thanks to ITEA2 project PREDYKOT, we have tested our approach on other use cases.

8 Acknowledgement

This work has been funded by ITEA2, Project PREDYKOT.

References

[AE03] Asaf Adi and Opher Etzion. Amit - the situation manager. *The VLDB Journal The International Journal on Very Large Data Bases*, 13(2):177–203, September 2003.

[Bal12] Balana - The Open-Source Xacml V3.0 Implementation, http://xacmlinfo.org/, August 2012.

[BP09] Achim D Brucker and Helmut Petritsch. Extending access control models with break-glass. In *SACMAT '09: Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 197–206, Stresa, Italy, 2009. ACM Request Permissions.

[CFG11] Barbara Carminati, Elena Ferrari, and Michele Guglielmi. Secure information sharing on support of emergency management. *Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom)*, pages 988–995, 2011.

[Esp06] Esper contributors and espertech inc., http://esper.codehaus.org/, 2006.

[Eys01] G Eysenbach. What is e-health? *Journal of Medical Internet Research*, 3(2):e20, 2001.

[FACC10] Ana Ferreira, Luis Antunes, David W Chadwick, and Ricardo Correia. Grounding information security in healthcare. *International Journal of Medical Informatics*, 79(4):268–283, April 2010.

[HW04] Junzhe Hu and Alfred Weaver. A Dynamic, Context-Aware Security Infrastructure for Distributed Healthcare Applications. In *Proc. 1st Workshop on Pervasive Privacy Security, Privacy, and Trust (PSPT)*, Boston, MA, USA, 2004.

[LKBB07] Romain Laborde, Michel Kamel, Francois Barrere, and Abdelmalek Benzekri. A secure collaborative web based environment for virtual organizations. In

Digital Information Management, 2007. ICDIM '07. 2nd International Conference on, pages 723–730, 2007.

[MCMD11] Srdjan Marinovic, Robert Craven, Jiefei Ma, and Naranker Dulay. Rumpole: A Flexible Break-glass Access Control Model. In *the 16th ACM symposium*, page 73, New York, New York, USA, 2011. ACM Press.

[MWZL12] G Ma, K Wu, T Zhang, and W Li. A Flexible Policy-Based Access Control Model for Workflow. *Przegląd Elektrotechniczny*, 2012.

[MZZ12] Barzan Mozafari, Kai Zeng, and Carlo Zaniolo. High-performance complex event processing over XML streams. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 253–264, Scottsdale, Arizona, USA, May 2012. ACM Request Permissions.

[SCCB06] Thierry Sans, Fréderic CUPPENS, and Nora Cuppens-Boulahia. A Flexible and Distributed Architecture to Enforce Dynamic Access Control. In *IFIP International Federation for Information Processing*, pages 183–195–195. Springer US, 2006.

[SSH+08] Matthew A Scholl, Kevin M Stine, Joan Hash, Pauline Bowen, L Arnold Johnson, Carla Dancy Smith, and Daniel I Steinberg. An Introductory Resource Guide for Implementing the Health Insurance Portability and Accountability Act (HIPAA) Security Rule. *An Introductory Resource Guide for Implementing the Health Insurance Portability and Accountability Act (HIPAA) Security Rule*, October 2008.

[SWS13] Sigrid Schefer-Wenzl and Mark Strembeck. Generic support for RBAC break-glass policies in process-aware information systems. *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1441–1446, 2013.

[XAC13] eXtensible Access Control Markup Language (XACML) Version 3.0. OASIS Standard http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html, 22January 2013.

[YDM12] Juan Ye, Simon Dobson, and Susan McKeever. Situation identification techniques in pervasive computing: A review. *Pervasive and Mobile Computing*, 8(1):36–66, February 2012.