Specification and Enforcement of Dynamic Authorization Policies oriented by Situations

Bashar Kabbani, Romain Laborde, François Barrère, Abdelmalek Benzekri

Specification and Enforcement of Dynamic Authorization Policies oriented by Situations

Bashar Kabbani, Romain Laborde, François Barrere, Abdelmalek Benzekri
Institute of Research in Informatics at Toulouse (IRIT)
University of Paul Sabatier (UPS)
Toulouse, France
{Kabbani, Laborde, Barrere, Benzekri @irit.fr}

***Abstract* — Nowadays, accessing communication networks and systems faces multitude applications with large-scale requirements dimensions. Mobility –roaming services in particular– during urgent situations exacerbate the access control issues. Dynamic authorization then is required. However, traditional access control fails to ensure policies to be dynamic. Instead, we propose to externalize the dynamic behavior management of networks and systems through *situations*. Situations modularize the policy into groups of rules and orient decisions. Our solution limits policy updates and hence authorization inconsistencies. The authorization system is built upon the XACML architecture coupled with a complex event-processing engine to handle the concept of situations. Situation-oriented attribute based policies are defined statically allowing static verification and validation.**

***Keywords*— dynamic authorization, access control, policy-based management, attribute-based*; situation management,* situation-oriented systems, complex event processing**

I. INTRODUCTION

Authorization process itself has a simple objective to obtain. No matter how scenarios get complicated, authorization process will always be to acknowledge users about their rights and limits towards environments' security and privacy. Identity and Access Management (IAM) cares most about the right identity getting authorized to the right resource at the right time with respect to conditions and circumstances of the identity's holder. However, externalizing the authorization is no more enough to protect access in the eyes of the business management. Due to IAM complex scenarios, this authorization process needs to have dynamicity in its decisions, no more static authorizations (permissions).

For instance, IAM business requirements in Virtual Organization (VO) are concerned about networks and tools evolution that promote collaborative work. Multipartite projects need to federate experts on different areas and information systems toward completing the common objective. Communications between distributed parties brings many security issues. Under certain circumstances, administrators have to unlock some of the security doors of parties information systems. During a past European project called VIVACE[1] [6], unlocking security issues was solved through a workflow engine. This article retakes the use case scenario to prove the concept of dynamic authorization using situation orientation simplifies authorization specification and management.

The article proposes a dynamic solution as a generic and a flexible architecture. Therefore, we demonstrate the flexibility by presenting a second use case where situations evolves and changes frequently and in real-time. The use case is known in healthcare as "Break The Glass". A very good example concerns about sharply changing rules for a noble reason of saving people's lives. The scenario is proposed by another European project founded by ITEA2[2], named PREDYKOT[3].

Current research trends are treating both scenarios using what we call *static authorization* policies. Static because whenever a request is made, the evaluation of policies' rules will always return the same result (permit or deny) for this given request. In this case, making authorization dynamic requires modifying the policy; which is a complex task.

This article is about keeping the authorization policy simple and static (immovable: defined once and for good), unless a necessary changement is required as for the case of writing the law. It proposes to follow software engineering best practice [16-17]. The definition of security requirement should be clear and sufficient for this approach to express entities situations inside the environment.

Dynamicity is provided using an externalized architecture that ensures the management of situations. Using attribute-based approach to express the authorization policy side-by-side to the dynamic architecture gives dynamic authorization decisions. Whereas, **dynamic authorization** is: "*the matter of giving (allowing) and taking back rights (denying) for and from entities who want to access resources at anytime, anywhere and for any reason of which conditions and circumstances are no more meeting business requirements*."

[1] Value Improvement thought a Virtual Aeronautical Collaborative Enterprise http://www.vivaceproject.com/
[2] Information Technology for European Advancement http://www.itea2.org
[3] Policy REfined DYnamically and Kept On Track http://www.itea2-predykot.org/

We define ***situations*** as remarkable conditions and circumstances reflecting abstract semantics to describe past, present or future behavior of entities or systems. Security situations must not be limited anymore to system status, but recognizing for example persons and/or tasks as well, i.e., any systems' entities.

We propose to implement our dynamic authorization architecture as a solution for both scenarios by well expressing the *authorization policy*. For that and toward a generic architecture, we have chosen the generic model ABAC[4]. For implementation, we use OASIS[5] XACML as an attribute-based policy language attached to Policy-Based Management (PBM) architecture. We represent situations as attributes that aggregate rules. Situations values are what give dynamicity to the static policy. The values highlight the appropriate decision that will be made after evaluating the associated rules. We use Complex Event Processing (CEP) to identify situations and calculate their values. CEP is an external component of the XACML PBM architecture. The component dynamicity will provide different values that dynamize the authorization.

The article is structured as following: Section II presents two scenarios the first is from previous work about Virtual Organization and the second is related to Healthcare Information Systems. In Section III, we present a study on related works. Section IV is dedicated to present the architecture that we propose as a solution based on situation management and attribute-based access control. Orienting the authorization policy and decisions using situations to provide Dynamic Authorization in both mentioned scenarios is explained in Section V with a brief description about prototypes implementing the architecture. Finally, we conclude our presented work in Section VIII.

II. SCENARIOS

We present two scenarios that require a dynamic authorization solution. In Virtual Organizations (VO), the scenario is about authorizations managed by workflow in a collaborative environment. The second scenario cares about authorizations during abnormal situations in Healthcare environments. Patients' life could reach dangerous levels and the authorization decision becomes a critical process.

A. Authorization in Virtual Organizations

This scenario deals with authorizations management in virtual organizations [6]. We resume it as following:

People from different aeronautical companies wish to collaborate in producing a technical specification document. Each organization has people with specific skills and software to complete specific tasks. The technical document has to be created by a designer, then analyzed and finally validated. This sequence of tasks is structured by a workflow, controlled by a conductor. The collaboration is formalized by a contract that states which company provides what kind of employees. Each company is responsible for managing its people and no constrains on who is doing the job. The contract also specifies access control policies on the shared documents. During design task, only people with designer role are permitted to access (read/write) the shared documents. The same for the analyzers and validator roles, they access only when their task has started. However, any VO members can read the final results when all tasks are finished. A workflow engine (WFE) acts as a conductor. During design task, the engine enables the set of rules that concerns the current document status and when it is completed, workflow engine disables them and enables another. Finally, when the process terminates, the engine informs participants and grant the "read permission" to all roles on the shared documents.

B. Authorization when Breaking Glass

In healthcare, "Break The Glass" (BTG) is another good example that treats the *dynamic authorization* problem through bypassing traditional authorizations (static). In emergency circumstances, unauthorized doctors can break the policy to get rights to access information that they could never have in an ordinary case. Well-known solution for BTG is giving entities administrative rights to break policy and modify rules when it comes to patients' life. *Our scenario is:*

Emma is a doctor in a modern hospital. Patients' information is not an open access for doctors. The intervention should be legalized by a reason, e.g. treatments. Therefore, only doctors with reasons are allowed to access. Patients in such hospitals 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 in such situation for the concerned patient. Assuming that one-day connected sensors to patient Joe show urgent need for a doctor. While his responsible doctor is not available, Emma was ready to involve. Unfortunately, Joe is not one of her patients. Traditionally, she won't have access. The BTG solution gives Emma rights in emergency situations to break general policy in order to saves Joe's life.

III. RELATED WORK

Works on dynamic authorization can be divided into mainly two axes. The existing approaches try to modify the policy and replace part of it frequently. Dynamic access control decisions [1] extended the role-based access control model to keep traditional access control methodology alive. Adaptive access control decisions takes in account different states of *subject*, e.g. requesting and waiting, in the decisions [2,3,4]. It is, however, oriented toward dynamic enforcement rather than authorization [5]. Researches on leveraging healthcare privacy in emergencies are obvious enough, e.g., [7,8].

To the best of our knowledge, using dynamic authorization has presented only through modifying the policy. *Situation management* is mostly active filed within situations and context awareness for decision-making [1,2,4,9,10,13]. It is currently concentrated on the *pervasive computing* trends. Many researches published in aim of anticipating, calculating and identifying situations and context to be proactive [14,15].

In retrospect, dynamic authorization is about energizing the authorization mechanism to be changeable toward new factors. Research trends are focused on having the dynamic authorization as an outcome of modifying the policy, i.e. dynamic policy by change, add and remove rules. None of the

[4] Attribute-Based Access Control
[5] Organization for the Advancement of Structured Information Standards

work was directed to provide dynamic authorization" as an income to the policy, translation of *Law*, without changing it.

IV. SITUATION-ORIENTED AUTHORIZATION

Dynamic authorization is provided using a situation-oriented solution. The solution is an architecture composed of two main parts: situations and authorizations. The Situation-Oriented Authorization Architecture (SOAA) main objective is to evaluate policies (set of rules) to provide dynamic authorization decision when a situation starts, e.g., intrusion detection, or during the period of situation occurrence, e.g., resource under attack. The situation orientation is an independent process that monitors the behavior to deliver dynamicity. To use this dynamicity, the policies expression should contain a place to represent situations. We propose the following representation for authorization policies:

When ***situations*** then Verify ***Conditions*** to Enforce ***actions***

As a result, the situations will indicate the decision-making process to the correct rule-set that match the correct context. When the decision understands the context, the dynamicity is met. Therefore, managing situations is an important phase as it gives accurate and correct indicators about context in order to dynamize the authorization-decision process.

A. Situation Management

In English and French literature, situation is "*A set of conditions and circumstances in which one finds oneself*". Researches in *pervasive systems* are advanced in situation-awareness domain and have proposed a more detailed definition as: "*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 a situation [9].

This article is concerned about detecting and identifying ***situations*** once they appear – not the fact of predicting future situation. Therefore, we refine the definition of situation as: "*a set of contexts detected because of predefined conditions and circumstances within the application over a timeline that affects the current and future system's behavior"*. For any situation, it is essential to determine 1) ***conditions***, e.g., having only designers the right to create the shared technical document, 2) and ***circumstances***, e.g., having readings from sensors telling about the bad status of the patient, in which the healthcare authorization system found itself in need of Emma.

Technically speaking, identification of situation's occurrences depends mainly on ***complex events (CE)*** [12]. Without *context*, events will not have any semantic. Events express instantaneous activities and phenomena, e.g. patient's pulse or doctors' appointments. Complex events are aggregated, filtered and correlated events using logical operations to express a meaningful phenomenon. Complex event could be a single simple event or a collection of complex events. CE has predefined meaning that helps defining situations. For instance, the fact that designers created the shared technical document and started working (writing) on it is considered as the *situation's* starting-point. Another example could be "urgent need for a doctor" that is a situation starts with the detection of complex events like a heart attack, nerves crises and other of what indicate patient in danger and needs a doctor urgently. This situation is an example of individual's situation (the patient). In the VO scenario, the document being in the design task is an example of resource's situations.

Figure 1 illustrates how to technically identify defines a situation. Once a situation starts, its value as an attribute orients the authorizations decisions towards suitable permissions until the situation ends. The ending-point of a situation is identified by complex event(s), e.g. the start of next task in the workflow ends the previous. In Figure 1, the situation manager (SM) is responsible about the identification of start-points and end-points. However, the event-processing engine (as a complex event processing) detects them and informs situation manager. During the situation life, the event-processing engine (EPE) traces its status and changes to keep the situation manager updated. Therefore, the EPE tool defines *patterns*. Patterns are mechanism similar to the polling that keeps the SM updated about its situations' status within a sliding window, time frame. Once the situation is identified, patterns could be used to make sure that the entities are still in the same situation (did not evolve or vanish). Patterns are expressed using SQL-like queries.

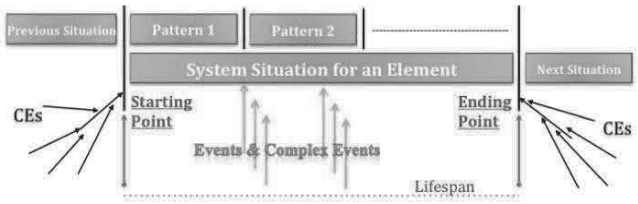

Fig. 1. Identification of Situations

The situation manager should well express business needs as situations. Main interest of the SM is to well configure EPE for the detection and calculation process. Adi et al. in Amit – The Situation Manager [10] already made a remarkable progress in terms of technical situations and event processing. We were inspired by this article to introduce the situation manager as a helpful part in SOAA architecture.

The situation manager has to determine the situation life and to manage it during its life. The situation life is the temporal context during which the situation detection is relevant. It is an interval bounded by starting and ending point. An occurrence of the start-point initiates the situation life and an occurrence of end-point terminates it. Both occurrences should be defined and initiated previously. Two situations may have the same starting and maybe ending dates. However, the semantic of each situation is evidentially different from one to another. Moreover, each situation is coupled with the elements participating in the creation of the situation.

The EPE will detect and monitor each activity in order to determine in which situation the entities are. Therefore, the architecture is built-up to never **mix the management of situations with the policy itself**, but to ensure interactions in-between. Finally, The situation management outcome could be stored in a database where the authorization section could get use of situations values to fill attributes.

B. *Specification and Deployment of Situation-Oriented Authorization*

The following section introduces the language used in our implementation to represent policies. Afterwards, we describe how to deploy the policies inside the situation-oriented architecture. Combined together, the policy

1) eXtensible Access Control Markup Language v3.0

XACML is a generic, flexible and abstract language with an architecture that could express/enforce our expected policy. This article takes advantage of such features to present how it is possible to handle situation oriented authorization policies. 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*, known as "***categories***" in XACML, for which the access request is made.

Thanks to categories tags, XACML v3.0 [11] is not limited to basic authorization entities (subject, resource, action and environment). Attributes are manipulated through predefined data types and functions. Considering attributes makes the language very flexible. We employ XACML V3 flexibility by orienting the security policy rules using defined and recognized situations attributes (see previous section). Aggregation in version 3 is to use abstraction (being close to requirements) in order to express groups of rules. We use this feature to group or aggregate rules by situations. After the Situation Manager provides situations values, the Policy Decision Point (PDP) will consider them in its decisions.

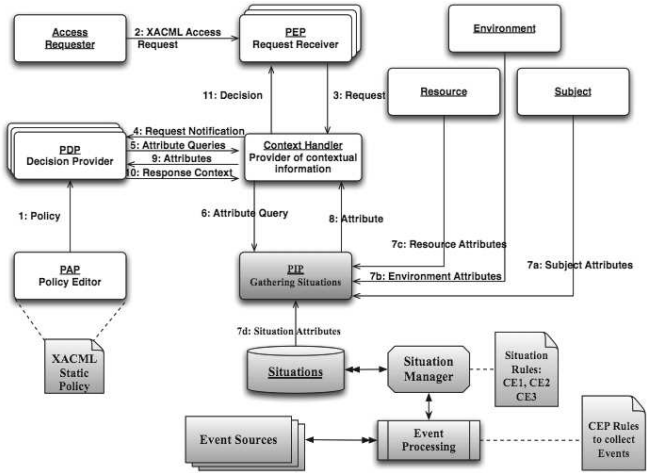

Fig. 2. Situation-Oriented Authorization Architecture

The XACML management architecture describes different entities. The entities' roles participate in the decision-making process; see Figure 2 boxes in white. 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 the attributes from a Policy Information Point (PIP) (step 6), the box is in violet as it is the connection between two architectures. The PIP obtains the requested attributes and returns them to the context handler (step 7, 8). The context handler sends the requested attributes. The PDP evaluates the policy and returns the standard XACML response context (including the 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).

2) Situation-Oriented Authorization Architecture

Figure 2 demonstrates our complete proposed architecture (SOAA) that combines the OASIS XACML (*boxes in white*) architecture and situation management architecture (*boxes in gray*). SOAA monitors the environments' behavior to express a very abstract terms, i.e. situations. Situations of systems or systems' entities play a role in any authorization decision. With a well understanding of situations, the authorization process will be dynamic. Decisions enforcement can be detected as well as part of the environment's behavior and then participate in the situations calculation. As a result, we have a loop of management that would participate in ensuring authorizations dynamically in policies.

V. APPLYING DYNAMIC AUTHORIZATIONS TO SCENARIOS

Our solution is generic enough to be applied to both proposed scenarios. We will apply the architecture by analyzing each scenario apart. First, analyzing possible situations that we should define, identify and calculate. Furthermore, the EPE and SM are configured to take account of these situations. The XACML policy is written based on these situations in a way they can orient authorizations dynamically to grant the right permission.

A. *Virtual Organizations*

Giving this scenario, instead of implementing this scenario by changing the authorization rules like it is suggested, we analyze it through our situation-based approach. We identify several situations that could participate in the authorization process. In Figure 3, all identified situations are from one-type that concerns the ***resource situation***, i.e. shared document.

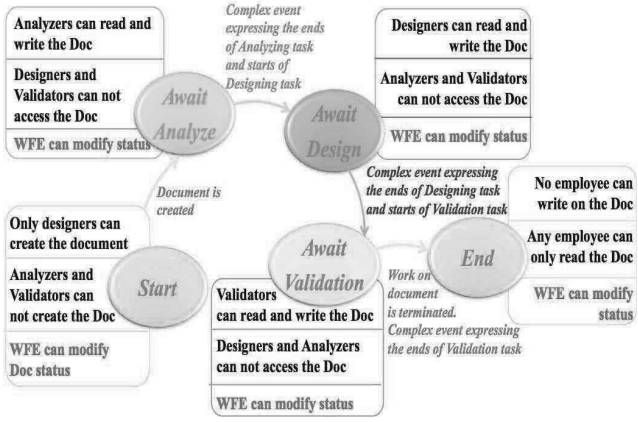

Fig. 3. VO Situations Cycle

The situations are: *work started*, *await analyze*, *await design*, *await validation* and *work terminated*. Initially, the VO authorization system evaluates the access requests towards the

shared document based on the static policy and oriented by the situation of developing the document. During these situations (between their start-point and end-point), only employees from the company that is responsible about the current work are permitted to do actions specified in the policy. For example, the start of the workflow begins with the situation *work started* (start-point) and ends with *work terminated* (end-point). At the simple vision, the start-point of each developing document situation is an end-point for the next one, e.g. the *await designing* is the end-point for *await analyzing* and so on.

We generalize the approach presented in [6]. The situation manager defines situations values. However, workflow engine (WFE) still in charge and it should interact with the SM in order to define the status of each task. The SM however controls the situation of the document. As a result we will have following rules:

Rule 1: IF *Situation (Resource) = Started ∧ Role (Subject) = Designer ∧ Type (Resource) = Specification-Doc ∧ Type (Action)= Create ➔ Permit*
Rule 2: IF *Situation (Resource) = Await-Design ∧ Name (Resource)=Shared-Doc ∧ Role (Subject) = Designer ∧ Type (Action)= (Read ∨ write) ➔ Permit*
Rule 3: IF *Situation ((Resource) = Await-Analysis ∧ Role (Subject) = Analyzer ∧ Name (Resource) =Shared-Doc ∧ Type (Action)= (Read ∨ write) ➔ Permit*
Rule 4: IF *Situation (Resource) = Await-Validation ∧ Role (Subject) = Validator ∧ Name (Resource)=Shared-Doc ∧ Type (Action)= (Read ∨ write) ➔ Permit*
Rule 5: IF *Situation (Resource) = Delivered ∧ Role (Subject) = ANY ∧ Type (Resource) = Shared-Doc ∧ Type (Action)= Read ➔ Permit*
Rule 6: IF *Role (Subject) = WFE ∧ Type (Resource) = Shared-Doc ∧ Type (Action)= Update-Status ➔ Permit -- Static Authorization --*
Rule 7: By default ➔ *Deny*

By considering these five situations, the policy expressing the dynamic authorization can be represented by the **seven rules** above. The *first rule* gives permissions to designers to create the shared document. The *rules 2, 3* and *4* ensure that a user can access the shared document with read or write permission if his role matches the current situation of shared document represented by its situation, i.e. if he is designer during await-design situation, if he is analyzer during await-analyzer situation and if he is a validator during the await-validation situation. The *fifth rule* means that a user can access the shared document with read-only permission if he has the role any, i.e. any designer, analyzer and validator, with the document situation are delivered. The *sixth rule* express static authorization that will allow, with discard to situation, the WFE to modify the status of the document that defines the situation of the work on the document. Finally, to manage conflicts we declare a default *seventh rule* that denies all other access requests.

B. Breaking the Glass

Breaking the general policy is required to change the authorization. To change rules (break policy), one should have administrative permissions to do. 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 article. 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.

Giving the BTG scenario, three situations related to authorization appear (Figure 4). *Identified situations are from* ***two-type*** *that concerns the* ***resource situation****, i.e. Patient's Information, and* ***users' or individuals' situations****.*

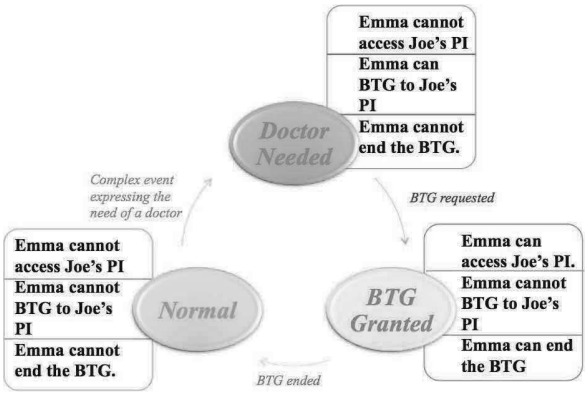

Fig. 4. BTG Situation Cycle

Initially, the healthcare authorization system evaluates the access requests to PI based on the static policy oriented by a normal situation. Within these situations, Emma is not permitted to do any of the following actions: access Joe's PI, request BTG to Joe's PI or end the granted BTG. When Joe's health is in danger and no responsible doctor is nearby to save him, 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, as it is not placed yet. Once the glass is broken by a BTG request, Emma will be able to access Joe's PI. However, she cannot place another request to BTG, as it has been already broken. Once Emma finishes treating Joe, she can end the granted BTG. Then, the situation of Joe will be back to normal and the cycle is completed.

Rule 1: IF *Type (Resource)=PI ∧ Role (Subject) = Doctor ∧ Type (Action)=Access ∧ ID (Subject) ∈ PRPL (Resource) ➔ Permit -- Static Authorization --*
Rule 2: IF *Situation (Owner (Resource)) = Doctor In Need ∧ Role (Subject) = Doctor ∧ Type (Resource) =PI ∧ Type (Action) = BTG Request ➔ Permit*
Rule 3: IF *Situation (Resource) = BTG Granted ∧ Role (Subject) = Doctor ∧ Type (Resource)=PI ∧ ID (Subject) = BTG Requester (Resource) ∧ Type (Action) =Access ➔ Permit*
Rule 4: IF *Situation (Resource) = BTG Granted ∧ Role (Subject) = Doctor ∧ Type (Resource) =PI ∧ ID (Subject) = BTG Requester (Resource) ∧ Type (Action)=End BTG ➔ Permit*
Rule 5: By default ➔ *Deny*

By considering these three situations, the policy expressing the dynamic authorization can be represented by the **five rules** above. Persons Responsible of Patient's information List (PRPL) is a list contains all persons who can access the PI of a patient with the patient himself. The *first rule* gives always a static authorization: if the doctor requesting access to Patient Information (PI) is mentioned in the PRPL. In order for Emma to be able to place a request to break the glass (event BTG Request), the patient should be in situation that needs an external doctor. In this case only, she can break the glass using the *second rule*. The *third rule* is to ensure that only doctors who requested the BTG can access the patient information when the PI situation is BTG granted. The *fourth rule* is to let Emma ends the BTG process during the situation BTG granted. Finally, we declare a default *fifth rule* that denies all other access requests.

C. *Prototyping Example*

We have implemented our both examples 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 situation 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.

Components that participate in the deployment of our prototype are mainly the Complex Event Processor (CEP), 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 CEP tool ESPER[6] and the BALANA[7] XACML v3 implementation. The ESPER engine will detect situations and monitor several activities. The engine will react on situations by updating the values in the situation's database.

ESPER is configured to simulate events like: Fever, Pulse, Status, Patient Position, Doctor Position and Room Occupancy. It composes events and listens to both access and BTG requests. ESPER detects a complex event (CE1) when the Fever is high, the status of the Joe is claiming and there is no one responsible near him. The detection of CE1 forwards an alarm to the SM who understands Joe is in danger. SM stores this situation in the DB. When an event detected about unavailability of responsible doctors for Joe, ESPER CEP engine detects that no one near Joe, he is in danger and no responsible doctors. ESPER CEP generates (CE2) saying that Joe needs a doctor urgently. SM understands that Jos is in a situation "urgent need for a Doctor", i.e. stored in the DB. Emma was available, so she places directly a BTG request about Joe's PI and will be permitted to break the glass. Emma now can access Joe's PI and save his life.

VI. Conclusion & Future Work

Modern systems require dynamic authorizations. This article has presented an approach for specifying and enforcing dynamic authorization policies based on situations. Current solutions propose to make a policy dynamic by modifying authorization rules when the conditions and circumstances are changed. The advantage of this approach is the ability to work with existing systems. However, the drawback is the rule management complexity when considering many situations and many rules. Keeping loaded rules free of conflict is a big challenge. In addition, re/analyzing the policy in this approach requires knowing what rules are loaded at anytime. As a consequence, it seems complicated to follow this approach.

In this article, we have presented a dynamic approach handled by a situation manager that keeps authorization rules static. The presented approach requires authorization policy languages be able to express situations to orient authorizations by their values. The benefits are important since policies are easier to: 1) Understand because they are closer to the security requirements 2) Analyze since rules do not change. Our choice of XACML language v3.0 refers to its capacity to represent any security information using attributes. In addition, the modularity of the XACML architecture facilitates its integration for enforcing situation based authorization policies. The architecture implementation demonstrated how it is possible and simple to provide dynamic authorization without modifying the policy. Based on a static policy, the prototype was able to manage *virtual organizations* and *Break-The-Glass* using situation-oriented authorizations.

[6] http://esper.codehaus.org/

[7] http://xacmlinfo.org/category/balana/

Acknowledgment

The work is thankful to be financially supported by the EU ITEA2, Project 10104 PREDYKOT (Policy REfined DYnamically and Kept On Track).

References

[1] Hu, J., & Weaver, A. (2004). A Dynamic, Context-Aware Security Infrastructure for Distributed Healthcare Applications. Proc. 1st Workshop on Pervasive Privacy Security, Privacy, and Trust (PSPT).

[2] G. Zhang and M. Parashar, "Dynamic Context-aware Access Control for Grid Applications," presented at the GRID '03: Proceedings of the 4th International Workshop on Grid Computing, 2003.

[3] B. Aboba, M. Chiba, D. Mitton, M. Eklund, and G. Dommety, "Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS)," 2008.

[4] Jih, W.-R., Cheng, S.-Y., Hsu, J., & Tsai, T.-M. (2005). Context-aware Access Control in Pervasive Healthcare. Presented at the In EEE'05 Workshop: Mobility, Agents, and Mobile Services (MAM).

[5] Sans, T., CUPPENS, F., & Cuppens-Boulahia, N. (2006). A Flexible and Distributed Architecture to Enforce Dynamic Access Control. IFIP International Federation for Information Processing.

[6] R. Laborde, M. Kamel, F. Barrere, and A. Benzekri, "A secure collaborative web based environment for virtual organizations," presented at the Digital Information Management, 2007. ICDIM, 2007.

[7] Carminati, B.; Ferrari, E.; Guglielmi, M., "Secure Information Sharing on Support of Emergency Management," Privacy, security, risk and trust, 2011 3rd international conference on social computing (socialcom).

[8] Røstad, L. (2009, January). "*Access Control in Healthcare Applications*". Norwegian University of Science and Technology.

[9] J. Ye, S. Dobson, and S. McKeever, "Situation identification techniques in pervasive computing: A review," Pervasive and Mobile Computing, vol. 8, no. 1, pp. 36–66, Feb. 2012.

[10] A. Adi and O. Etzion, "Amit - the situation manager," The International Journal on Very Large Data Bases (VLDB), vol. 13, 2, 177–203, 2003.

[11] eXtensible Access Control Markup Language (XACML) Version 3.0. 22 January 2013. OASIS Standard.

[12] Mozafari, B., Zeng, K., & Zaniolo, C., "*High-performance complex event processing over XML streams,*" Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, Scottsdale.

[13] Freudenthaler, B., & Stumptner, R. 2nd International Workshop on Information Sys. for Situation Awareness and Situation Management.

[14] Situation Management: Basic Concepts and Approaches. In V. Popovich, M. Schrenk, & K. Korolenko, Lecture Notes in Geoinformation and Cartography. Springer Berlin Heidelberg. (2013).

[15] Dynamic situation monitoring and Context-Aware BI recommendations. (M.-A. Aufaure, Ed.). Ecole Centrale Paris, MAS Laboratory, and SAP Research, Business Intelligence Practice.

[16] Anderson, R. J. (2008). Security Engineering: A Guide to Building Dependable Distributed Systems, 2 edition.

[17] Bhatti, R., Bertino, E., & Ghafoor, A. (2006). X-FEDERATE: a policy engineering framework for federated access management. IEEE Transactions on Software Engineering.