EAI Endorsed Transactions
on Self-Adaptive Systems
Research Article ICST.ORG

"Why can't I do that?": Tracing Adaptive Security Decisions

Armstrong Nhlabatsi[1], Thein Tun[2], Niamul Khan[1], Yijun Yu[2], Arosha K. Bandara[2], Khaled M. Khan[1], Bashar Nuseibeh[2,3]

[1]Qatar University, {armstrong.nhlabatsi, niamul.khan, k.khan}@qu.edu.qa
[2]The Open University, {t.t.tun, y.yu, a.k.bandara, b.nuseibeh}@open.ac.uk
[3]Lero, University of Limerick, bashar.nuseibeh@lero.ie

Abstract

One of the challenges of any adaptive system is to ensure that users can understand how and why the behaviour of the system changes at runtime. This is particularly important for adaptive security behaviours which are essential for applications that are used in many different contexts, such as those hosted in the cloud. In this paper, we propose an approach for using traceability information, enriched with causality relations and contextual attributes of the deployment environment, when providing feedback to the users. We demonstrate, using a cloud storage-as-a-service environment, how our approach provides users of cloud applications better information, explanations and assurances about the security decisions made by the system. This enables the user to understand why a certain security adaptation has occurred, how the adaptation is related to current context of use of the application, and a guarantee that the application still satisfies its security requirements after an adaptation.

Keywords: Traceability, Causality, Entailment Relation, Security Requirements, Access Control Policies.

Received on 19 November 2014, accepted on 17 January 2015, published on 28 January 2015

doi: 10.4108/sas.1.1.e2

1. Introduction

Many software applications are now deployed as Cloud Services in order to allow users to access them from a variety of devices, wherever they happen to be. This requires that these applications be able to adapt their behaviour, in order to ensure that requirements continue to be satisfied even when the context of use changes. This is particularly important for critical quality requirements such as security requirements.

For example we may want a cloud application to change its security behaviour depending on where (location) it is used, who (subject) is using it, or when (time) it is being used. We call this *Adaptive Information Security* (AIS). As a result of dynamic context, the assets, their values, and attack scenarios can change easily from one situation to another, increasing the challenge of finding out what the information assets are, who their owners are, where in the system vulnerabilities lie, and the extent to which the security requirements are satisfied.

One of the challenges of any adaptive security system is to ensure that users can understand how and why the security behaviour of the system changes at runtime. For example, a doctor may be able to edit a medical record stored on a cloud server using one device but only able to read the same medical record when it is accessed from a different device. This is because an access control policy for maintaining confidentiality and integrity of medical records may dictate that the doctor is able to gain access to edit rights only when he is on duty. Sensors in the device used when accessing a record determine the contextual property of whether the doctor is on duty. As a result, when using a device with limited capabilities the access control mechanism may not be able to determine that the doctor is on duty resulting in limited access to the

record. In this situation, the doctor needs to understand why his access rights appear to change in a seemingly arbitrary way.

The example illustrates that there is a relationship between the security requirement to maintain the integrity of the medical record and the granting/revoking of access rights. The doctor gets certain privileges when his contextual attributes say that he is on duty and gets a different set of privileges otherwise. In effect, the privileges the doctor gets are indirectly determined by the properties of the device he is using. As such if we know the properties of the device, at the time an access control decision is made, then we can determine which privileges the doctor gets. In this way we can relate the security requirement to a particular set of permitted privileges via device properties. This illustrates the role the context plays in relating requirements and security policies. We call such relationships *traceability*, a well-established concept in the software engineering literature. Traceability is generally defined as the ability to establish and keep track of the relationships between requirements, design artefacts, source code, test cases, etc [1][2]. Users of adaptive applications may not always have detailed knowledge on how security decisions are determined with respect to their context of use. This may lead to loss of confidence and trust in the adaptive application when its behaviour deviates from the user's expectations.

The main contribution of the paper is an approach to providing users of cloud applications *information*, *explanations*, and *assurances* about the security decisions made by an adaptive information security system. This enables the user to understand why a certain adaptation has occurred, how the adaptation is related to current context of use of the application, and steps they (users) may take to guarantee that the application satisfies its security requirements after an adaptation. Our approach is rooted in the well established concepts of *entailment* [3], *traceability* [4], *and causality* [5] as used in the software engineering literature.

The entailment relationship is a framework that relates requirements, domain properties, and specification [3][6] of a software system. We use this framework for establishing traceability relationships relevant to adaptive information security systems and as a means of evaluating satisfaction of security requirements at runtime. Our traceability relationships are augmented with causality relations, namely, the effect of different actions or events on the state of the system. In our example, one can state the causality relation that the event of the doctor arriving at the hospital results in him being on duty and the event of him leaving the hospital results in him being off duty (i.e. not on duty). We enrich such causality relations with contextual attributes of the deployment environment as a means to providing information about the behaviour of an adaptive application to the user. Our proposed approach is demonstrated through an example of a cloud storage as-a-service application.

With the exception of Bencomo et. al. [7], as far as we are aware, we are the first to propose mechanisms for helping the user understand security decisions made by an adaptive application. While Bencomo et. al.'s approach to self-explanation in adaptive applications provides a general framework for explaining adaptive behaviour, in our approach we focus more on explaining security decisions, a task that requires additional treatment. For instance, the explanations must satisfy confidentiality requirements such that they are not exploitable by a potential attacker pretending to be the legitimate user. Our proposed rich traceability links help in making explanations more accurate to the user's context, an important prerequisite for explaining security decisions.

The paper is organized as follows: §2 presents an access control example we use to motivate and explain the problem we are solving. §3 presents background on concepts our approach builds on and notations we use. §4 brings together the concepts and notations from §3 and shows how we use them for representing traceability through an example. §5 describes algorithms we use for tracing security decisions using the traceability information from §4 and we evaluate and discuss the limitations of our approach in §6. In §7 we review related literature and, finally, §8 concludes the paper and discusses pointers to further work.

2. Motivating Example: An Adaptive Access Control Application

Consider a hospital that uses a cloud-based electronic patient record (EPR) system. Bob is a doctor at the hospital and medical records of his patients are stored on the EPR system. The policy of the hospital is that Bob should only be able to access the medical records when he is on duty. Assuming that the EPR system has already authenticated Bob's identity, whether he is on duty or not is based on a combination of three contextual attributes, namely his *location*, *time of day*, and the *identity of the network* he is using when accessing the medical records. We use the values of the attributes to determine the truth-value of a Boolean variable *isOnDuty*. However, the availability of these attributes will depend on Bob's context when he attempts to access certain medical records. Therefore we use the availability of the attributes to determine a confidence level for the computed truth-value.

Because of variations in the features available in different devices Bob may use when accessing a medical record, the confidence level on his on duty status varies. The confidence level can be *very low*, *low*, *medium*, or *high*. The combination of the truth-value and confidence level is used to determine whether Bob is granted access to perform *Read*, *Write*, or *Share* operations on the medical record. Adam is the policy administrator who specifies the policies that control the behavior of the EPR application. One policy he specifies is that if the confidence level of isOnDuty is very low the user is not allowed to perform any of the operations. If it is low he can only read and if it is medium, he can Read and Write a medical record. If it is high he can perform Read, Write and Share operations.

As a user Bob has limited knowledge on why the EPR system behaves in certain ways at runtime as his context of use changes. Sometimes certain operations are permitted and sometimes they are denied. He only knows that when he is on duty the system should grant him full access rights on a medical record. He is not aware of the details of how the system determines that he is on duty, i.e., he does not know that his *isOnDuty* status depends on the three contextual variables.

As a result of the lack of the detailed understanding of the implementation of the system's adaptive behavior, Bob is sometimes confused about certain access control decisions. Bob wants a mechanism to help him understand why a certain adaptation has occurred and how the adaptation is related to the context in which he is using the system.

For example if he is denied the privilege to share a medical record when using his laptop while the same privilege is available when using his iPad, the mechanism could explain why this is the case. When *informing* Bob, the mechanism could say that he is not able to share a record because "*the system is not sure if you are on duty*." If Bob probes further for an *explanation* on why it is uncertain that he is on duty the mechanism would say "*a GPS sensor is not available in the device you are using. As a result, the system is uncertain about your location and cannot determine if you are physically at the hospital premises. The security policy dictates that you should be at the hospital to share a medical record*."

If Bob needs advice on how he can restore the sharing privilege then the *assurance* mechanism in the EPR system could advice Bob to "*use the hospital WiFi network to gain further privileges"*. The explanatory messages given to Bob are specific to his context. In this case, the message is specific to the nature of the device he is using for access.

Adam writes general policy rules that are instantiated for each user. He will not be able understand the specifics of each instance of the policy for each of the hundreds of users of the EPR system. Therefore Adam would benefit from a means of explaining specific security decisions and being assured that security requirements will be satisfied in specific contexts. By developing a way to relate security requirements, policies, and the EPR system's contexts of use, we can provide the information, explanations and assurances needed by users and administrators like Bob and Adam.

3. Background and Notations

In supporting our approach to explaining adaptive security decisions we use Zave and Jackson's entailment relation [8][9] and causality.

3.1 The Entailment Relation

Three sets of properties in Definition 1, namely the requirements R, the specifications S and the domain properties W are related.

Definition 1 *A* phenomenon *can be either a fluent (i.e. a condition that can change over time) or an event of a given domain. A predicate about the phenomena of certain domains is called a* domain property. *The properties of specified domains and those of the other domains are denoted by S and W respectively, the domain properties referred or constrained by requirements are denoted by R.*

The set of *requirements R* describes the properties of the software system as desired by its users, customers, and other stakeholders. Requirements are *optative* descriptions in that they describe how the world would be once the envisioned system is in place. In the EPR system, there is a security requirement controlling access to medical records that says: '*The doctor can read, write, or share a patient's medical record only when he is on duty depending on confidence level about his status of being on duty*'.

The set of *domain properties W* describes the behaviour of the context, which is the environment where the software system will be deployed. Attributes of the context have values that may determine the behaviour of the adaptive application. Unlike requirements, domain properties are *indicative* in that these properties hold both before and after the deployment of the software system. In the EPR example, an indicative property is: '*Working hours at hospital are from 8am to 5pm.'*

The set of specifications[*] *S* describes how the computer should behave in order to satisfy the domain properties described in R, given that the domain properties in W hold. The specification for the EPR system could be: '*After a successful authentication, the doctor can read, write, or share a patient's medical record only when he is accessing the record within working hours'*

In general the problem-solution relationship between the three sets of descriptions explained above is given below.

$$W, S \vdash R.$$

where ⊢ is the entailment operator.

The entailment relation does not prescribe languages for expressing the three artefacts. This has the advantage of giving the requirements engineers the freedom to choose a language of their choice for representing details of the three descriptions. In order to support the entailment we need to know the details about the specific behaviour of S and W. One way to describe the three artefacts (S, W, and R) is in terms of events and fluents.

A requirements traceability link can be derived from explicit requirements problems in Definition 2.

[*] For purposes of examples presented in this paper we regard specification to be equivalent to an access control policy and therefore use the two terms interchangeably.

Definition 2 *A* requirement traceability link (s, r) *is the relation between the specification* $s \in S$ *and the requirement* $r \in R$. *The semantics of requirements traceability links are explicated by a requirement problem that* $w, s \vdash r$ *is satisfied under certain context* $w \in W$, *denoted by* $T = \{(s, r) \mid \exists w \in W : w, s \vdash r\}$.

The entailment relation can also be modelled graphically using the *Problem Diagram* notation [10]. Figure 1 shows a problem diagram modelling the medical record access control example.

The machine to be designed is represented by the rectangle with double vertical lines. The medical record access control (ACM) is an example of a machine. Rectangles represent problem domains whose domain properties the machine rely in to satisfy the requirement. The medical record (MR) is an example of a problem domain. An access control specification for the ACM relies on properties of the medical record such as the ability of the record to be read or written. The access control requirement for allowing access when the doctor is on duty is represented as the dotted oval.

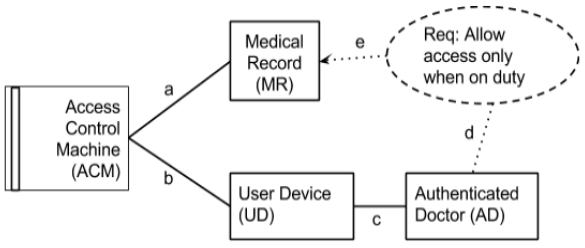

Figure 1: Problem Diagram of the Medical Record Access Control Machine.

The three entities in a problem diagram share phenomena between them through interfaces (see Definition 1). The solid lines *a* and *b* are machine interfaces. The dotted lines *d* and *e* are requirements interfaces. Dotted lines with an arrow (e.g. interface *e*) mean that the behaviour of the domain is constrained by the requirement. For example the ability of a medical record to be read, written, or shared is possible only if the doctor is on duty. Phenomena are either observed or controlled by one of the domains they link. For example, at the interface *a*, the phenomena *readable*, *writeable*, and *sharable* are controlled by the ACM machine. This is denoted by the '!' between the domain name and the phenomena. The MR domain observes these phenomena.

The medical record access control machine (ACM) represents the software that must implement the security requirement and its behaviour is specified using access control policies. To satisfy the requirement, the ACM interfaces with the medical record (MR) and the user device (UD). The medical record is the *asset* to be secured by the ACM by controlling whether the doctor is able to read, write, or share it. The authenticated doctor (AD) interacts with the user device, through interface c, to issue request actions for access to a medical record.

The truth-value and confidence level of the *isOnDuty* phenomena at interface d are computed by the ACM observing the phenomena *isWithinGPSRange*, *isCorrectSSID*, and *isWithinWorkingHours* at interface b. The phenomena *isWithinGPSRange*, *isCorrectSSID*, and *isWithinWorkingHours* are derived from sensors on the user device.

3.2 Causal Relations

The relationship between the events and state changes in domain descriptions can be described using causality relations. A causality relation is a way of describing how the occurrence of one event e_1 leads to the occurrence of a second event e_2 [5]. Event e_2 can be either another event or the change of a fluent [11]. In this paper we are interested in two types of causality relations: direct and transitive causalities. We use the Event Calculus as a notation for representing and reasoning about causality.

Direct and Transitive Causality

Direct Causality: The occurrence of event e_1 directly leads to the occurrence of event e_2, expressed as causal relations, which can be generalised in Definition 3.

Definition 3 *A* causal relation *between the cause phenomena* c *and effect phenomena* e *is denoted by* $c \hookrightarrow e$. *The overall knowledge of all causal relations is denoted by the set* $\mathfrak{E} = \{c \hookrightarrow e \mid c, e \in S \cup W\}$.

For example, when the time of day becomes 8am, the event *isWorkingHours* follows. There is a direct causality relationship between *ClockStrikes8am* and *isWorkingHours*. Using Definition 3, this is expressed as:

$$ClockStrikes8am \hookrightarrow isWorkingHours$$

Note the effect of time in this causality relation. The events *ClockStrikes8am* and *isWorkingHours* do not happen simultaneously but they are consecutive. If event *ClockStrikes8am* happens at some time t_0 then, according to Definition 3, *isWorkingHours* becomes true at a later time t_1.

Transitive Causality: Transitively, one may want to find a trace in Definition 4 as the evidence for their causality.

Definition 4 *A* trace *is a sequence of phenomena caused by one another, from the head to the tail, denoted by* $c \hookrightarrow^+ e$*, where* $\hookrightarrow^+$ *is defined* transitively *on* $\hookrightarrow$*, i.e.,* $c \hookrightarrow e \Rightarrow c \hookrightarrow^+ e$ *and* $c \hookrightarrow m \wedge m \hookrightarrow^+ e \Rightarrow c \hookrightarrow^+ e$.

For example, when the time of day becomes 8am, the Boolean variable *isWorkingHours* become true which leads to Boolean variables *canRead*, *canWrite*, and *canShare* becoming true. Using Definition 3 this can be expressed as follows:

$$isWorkingHours \hookrightarrow canRead \wedge canWrite \wedge canShare$$

There is a transitive relationship between *ClockStrikes8am* and the three events *canRead*, *canWrite*, and *canShare*. Using Definition 4 this can be expressed as:

$$ClockStrikes8am \hookrightarrow^+ canRead \wedge canWrite, canShare$$

Our use of these causality symbols in this paper has the same meaning as the casual relationships used for deriving specifications from requirements through problem reduction [11]. We use the Event Calculus (EC) as a formal language for representing causality. We chose the EC because: (1) it is expressiveness enough for representing the properties of causality that are of interest to the ideas presented in this paper; and (2) it has tool support for reasoning about causality. The next section gives a brief introduction to the EC.

Event Calculus Representation

The Event Calculus (EC) is a logic system for reasoning about how the occurrence of events changes the state of the world. We use the EC in this paper to express domain descriptions and to facilitate traceability.

Basic Constructs of the Event Calculus: The EC consists of three basic sorts: events, fluents, and timepoints [12]. An event represents an action, which may occur to a problem domain. For example, *ClockStrikes8am* is an event in the medical record access control problem. A fluent is a time-varying property describing the state of a problem domain, such as *isWorkingHours*. A timepoint is a time instant, for example t_0 denotes the timepoint 0.

Event Calculus Predicates: A fluent is either true (holds) or false (does not hold) at a timepoint or over an interval. The occurrence of an event at a timepoint may change the truth value of a fluent. Below is the domain description of a clock used to tell time of day in the medical record access problem.

Initiates (ClockStrikes8am, isWorkingHours, t) [CD_1]
Terminates (ClockStrikes5pm, isWorkingHours, t) [CD_2]
¬Initially (isWorkingHours) [CD_3]

When an event results in a fluent being true it is said to *initiate* the fluent. For example the time of day event *ClockStrikes8am* results in fluent *isWorkingHours* being true. Hence *ClockStrikes8am* is said to initiate fluent *isWorkingHours*. If an event causes a fluent to become false then that event is said to *terminate* the fluent. For example, time of day event *ClockStrikes5pm* terminates *isWorkingHours* i.e. when event *ClockStrikes5pm* occurs *isWorkingHours* become false. These relationships between the events and fluents describe the behaviour of a clock, which we will use in the medical record access control problem.

A domain description models a real-world domain and forms the basis for reasoning about the behaviour of the modelled domain. It is therefore important that its behaviour is consistent with the actual state of the real-world problem domain. In the EC all reasoning about future states is based on current states. The initial state of a fluent f1 is expressed with *Initially(f1)* clauses. These state that fluent f1 holds at time 0. For example, the domain description assumes that the clock is initially set to a time that is not within working hours. (CD_3). All other fluents not captured in the *initially* clause are assumed to be (initially) false and changes in their truth values are subject to the *common sense law of inertia.* This law states that a fluent remains false until *initiated* and remains true until *terminated*. Table 1 shows the predicates of the EC we will use and their meanings.

Table 1 Event Calculus Predicates

Fluent	Description
Initiates(e,f,t)	Fluent f starts to hold after event e at time t.
Terminates(e,f,t)	Fluent f ceases to hold after event e at time t.
Initially(f)	Fluent f holds at time 0
Happens(e,t)	Event e occurs at time t.
HoldsAt(f,t)	Fluent f holds at time t.
Clipped(t1,f,t2)	Fluent f is terminated between times t1 and t2.

Event Calculus Meta-Rules: Based on initial conditions, events that have happened, and rules that state how fluents are changed when events happen (domain descriptions), it is possible to determine which fluents hold. This is summarised in the EC rules below.

Clipped(t1,f,t2) ← ∃ a,t1 [Happens(a,t) /\ Terminates(a, f, t1) /\ (t1<t<t2)] [EC1]

HoldsAt(f,t1) ← Initially(f) /\ ¬ Clipped(0,f,t1) [EC2]

HoldsAt(f,t2) ← Happens(a,t1) /\ Initiates(a,f,t1) /\ (t1 < t2) /\ ¬Clipped(t1,f,t2) [EC3]

EC1 states that if an event happens in the period between t1 and t2, and that event terminates fluent f during that period, the fluent is said to be clipped in the period. EC2 states that a fluent holds if it held initially and no event has occurred to stop it holding. EC3 captures the common sense law of inertia described above. These

three rules are referred to as meta-rules since they form the foundation of all reasoning about occurrence of events and resulting effects in the Event Calculus language.

4. Traceability in Context

As illustrated by our example, the traceability between requirements and policies can be established through the context. Sections 3 explained the concepts of entailment and causality and we now bring them together in this section to describe how traceability is established. We also illustrate causal relations in our example.

4.1 Traceability Representation

Policies and Requirements are expressed at different levels of abstraction using different forms of language. At a higher level of abstraction, requirements are expressed in terms of the conditions to be fulfilled by the application. Meanwhile at a lower level of abstraction, policies specify the actions that need to be performed by an application in order to make the conditions stated in the requirements true. The difference in the way the requirements and policies are expressed makes the task of relating them less obvious.

Relating Problem and Solution Entities Through Context

The entailment relation relates requirements (R), context (W), and specifications (S). Our approach leverages this representation to relate security requirements to policies. We assume that policies are in the solution space as their behaviour satisfies the requirements: in other words, we regard them as specifications. We can think about context as the traceability link between requirement and policy which we propose can be represented through the domain properties. Figure 2 presents the general framework of our approach. The figure shows how we propose to relate entities in the *problem space* to those in the *solution space* through facts in the *context*. Requirements and policies are in the problem and solution spaces, respectively.

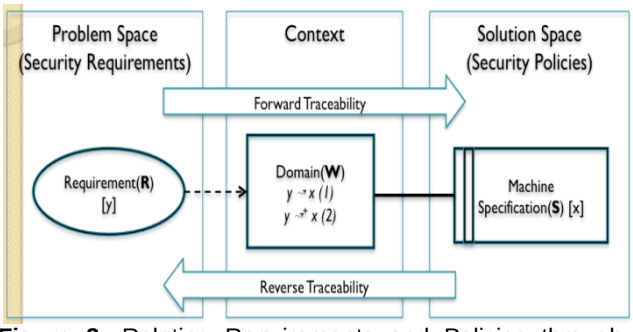

Figure 2: Relating Requirements and Policies through Domain Assumptions in the Context

We can think of the traceability link between requirements and policies in terms of the two links. The first link is between the problem and context domains – indicated by the dotted line between R and W in Figure 2. The second link is between the context and solution domain – indicated by the solid line between W and S. An interesting question is what specific attributes to consider in the requirement, context, and solution domains in order to establish these links. The next two sections address this question.

Traceability Link Between Requirements and Context

According to the entailment relation a requirement is defined as some property that must be exhibited by an application in order to solve some problem in the real world. For this reason we express a requirement in terms of the conditions we would like to be true in the context once the system is in place. The expression of a requirement references some attributes of the context. For example a requirement could say a doctor should have access to a medical record only when he is on duty. But what does being 'on duty' mean? We may say the doctor is on duty if he is within certain GPS coordinates, he is using the hospital WiFi for connection, and the time of day between 8am and 5pm. The explanation of what on duty means is derived from properties of the context.

The example illustrates that the requirement contains references to domain assumptions in the context. It is through such references that we relate the requirement to the context. The relationship between the requirement and context is captured by traceability link L1 in Figure 3.

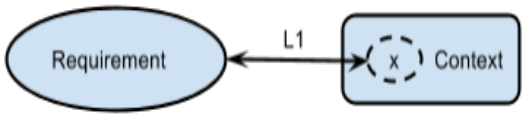

Figure 3: Traceability Link Between Requirements and Context

X represents the attribute in the context that is referenced by the requirement. In our example *isOnDuty* is a concept in the requirements, whereas GPS, WiFi, and Time-of-Day are attributes in the context. We can therefore distinguish between two types of the attribute x: x1 in the requirement and x2 in the context. x1 being the variable *isOnDuty* and x2 being the GPS, WiFi, and Time of Day.

Using domain knowledge, the context (x2) qualifies what *isOnDuty* (x1) means by stating how it is determined. Such reference provides a concrete relationship between the requirement and context thus forming a traceability link.

Traceability Link Between Context and Policies

Link L2 in Figure 4, represents a traceability link between the context and policies. Similar to entities in the problem space, solution space entities also make assumptions

about the behaviour of the context. The solution uses these assumptions to implement behaviour that satisfies the requirements. For instance the policies in our example implement access control mechanisms that determines who can or cannot access a medical record and the circumstances under which access should be granted or denied. Hence policies also references some attributes of the context. Again, we take advantage of these references to establish the traceability link between policies and context.

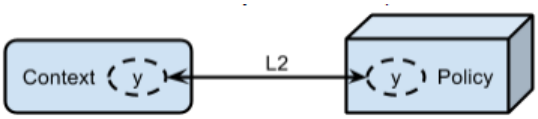

Figure 4: Traceability Link Between Context and Policies

In our example a policy could state that if a doctor is on duty then he should be allowed to read, write, and share a medical record. The policy assumes that some machinery is available for determining that the doctor is on duty or not. An example of such machinery is a location sensing equipment. In our example the location sensing is performed by a GPS component in the user device, which we assume the doctor carries with him. We use the reliance of the policy on properties of the context to establish a traceability relationship between the two.

In our example, the value of y is the data that has been read from the GPS component that is interpreted through the context to determine if the doctor is on duty or not. The relationship between x and y established via the facts given in the context is the traceability link that relates the y in the policy to the x in the requirement. Worth noting in this example is that even though y is part of the policy it is not an action. It is a piece of data that qualifies the condition *isOnDuty* stated on the requirements side. Hence the traceability information on the policy side is not necessarily expressed as actions but can also be any fact that helps us qualify the conditions stated by the requirement.

Required, Observed, and Designed Causality

The distinction between *requirements-to-context* and *context-to-specifications* traceability links does not say anything about the boundary of the access control system. By separating events in the identified causality relations based on whether the event belongs to the machine, requirement or domain, it is possible to further enrich the traceability relations. There are three types of phenomena [10]: required, observed, and designed.

It also follows that there are three types of causality events: (1) required causality; (2) observed causality; and (3) designed causality in Definition 5.

Definition 5 *In relation to requirements, a trace is respectively of* required, observed, *or* designed *if* $e \in R$, $c \in R$ *or* $c \notin R \wedge e \notin R$.

Required Causality: These are causality events at the requirements interfaces d and e. The user will observe the phenomena that he can read, write, or share a medical record as a result of events initiated by the access control machine in changing the access rights on the medical record. Required causality is captured by link L1 in Figure 3.

Observed Causality: These are events describing the behavior of a problem domain. The interaction between the doctor and the device, and the internal behaviour of the device that affects the phenomena is*WithinWorkingHours*, *isWithinGPSRange*, and *isCorrectSSID* are examples of observed causality. In Figure 1 interface c and any events internal to domains MR, UD, and AD are observed causality.

Designed Causality: These are causality events at the machine interface. Such causality events are said to be 'designed' because they come about as a result of behaviour of the machine specification. Machine interfaces a and b contain designed causality events. This corresponds to link L2 in Figure 4.

Refining Traceability Links through Causality

As stated earlier, when establishing traceability relations between requirements and policies we use causal links. However, there are important differences between traceability and causality links, which we explain in this section. Traceability links relate artefacts at different levels of abstraction. For example relating security requirements in the problem space to security policies in the solution space or relating a requirement to a section of source code that implements the requirement. On the other hand causal links relate events at the same level of abstraction such as the events that describe a domain. Definition 6 shows how causality can be used to refine the requirements traceability links.

Definition 6 *The set of* refined requirement traces *is* $\mathbb{E}^+(s,r) = \{c \hookrightarrow^+ e \mid c,e \in R, \exists d \in S \cup W : c \hookrightarrow^+ d \hookrightarrow^+ e\}$. *A traceability* (s,r) *is* causal *if there exists a refined requirement trace* $c \hookrightarrow^+ d \hookrightarrow^+ e$ *such that* $c,e \in r$ *and* $d \in s$. *The set of* contextual domain properties *of the refined requirement traces* $c \hookrightarrow^+ e$ *is* $W_{c \hookrightarrow^+ e} = \{d \mid c \hookrightarrow^+ d \hookrightarrow^+ e, d \in W\}$.

As an illustration, consider the GPS device for determining the doctor's location. Its domain description could say: *A GPS device gives its location in terms of latitude and longitude coordinates*. In this statement latitude and longitude are numbers that, on their own, do not have any meaning. The description gives these numbers a meaning by stating they are a *location*. We can go further and enrich this description by stating that these coordinates are the location of the hospital. Similarly, *location of the hospital* would be meaningless unless we can say exactly what geometric reference system are we using to locate it. The links between the GPS coordinates and the location of the doctor are causal.

The fact that location of the hospital is assigned to certain coordinates enables us to derive further facts such

as whether or not the condition *isOnDuty* holds. This is a traceability link as it relates the isOnDuty concept at the requirements level to contextual attributes (GPS coordinates, SSID, and time of day). Variables x and y are dependent on each other through causality. We exploit this dependency to define traceability within the context through domain descriptions. As stated earlier, x represents the Boolean variable *isOnDuty* while y represents the actual data that helps an access control machine to a medical record determine whether the value of *isOnDuty* should be true or false. In this case y is a combination of the GPS coordinates, WiFi SSID, and TimeOfDay.

In summary, using causal links enriched with contextual information we are able to trace between entities in the problem space to entities in the solution space. We achieve this in two stages: (1) relating requirements to context; and (2) relating the context to policies. The relationship is established through facts in the context that are referenced by both the requirement and policy. Our approach assumes that the domain assumptions (facts) about the context already exist and we use these to establish the traceability relationship of requirements with policies. The domain assumptions are the relations between x and y in the context that binds a requirement to a corresponding policy.

4.2. An Illustration of Causality

The traceability links established through the entailment relation is not rich enough to support the process of tracing security decisions. In order to make the tracing process feasible we enrich the traceability links with additional contextual information on causality. After presenting an example of requirement, domain properties, and policies we discuss the type of contextual information needed for tracing access control decisions.

Requirements, Policies, and Domain Properties

The requirement, context, policy specifications from the architecture in Figure 2 and problem diagram in Figure 1 can be instantiated through a refined version of the example presented in section 2 as follows:

Requirements:

- *R: If the doctor isOnDuty then allow him to perform certain operations depending on the level of confidence of the determination of isOnDuty.*

Context:

- W1: if the doctor is within hospital GPS coordinates, using hospital WiFi, and within working hour, then the doctor is on duty with high confidence level.

  [(latitude=X,longitude=Y)AND(8am<timeOfDay<5pm) AND (WiFiSSID=HospitalWiFi)] ↪ {isOnDuty=True,ConfidenceLevel=HIGH}

- W2: if the doctor is not within hospital GPS coordinates, using hospital WiFi, and within working hours, then the doctor is on duty with medium confidence level.

  [(latitude!=X,longitude!=Y)AND(8am<timeOfDay<5pm)AND (WiFiSSID=HospitalWiFi)] ↪ {isOnDuty=True,ConfidenceLevel=MEDIUM}

- W3: if the doctor is not within hospital GPS coordinates, using hospital WiFi, and not within working hours, then the doctor is on duty with low confidence level.

  [(latitude!=X,longitude!=Y)AND(8am>timeOfDay>5pm)AND (WiFiSSID=HospitalWiFi)] ↪ {isOnDuty=True,ConfidenceLevel=LOW}

- W4: if the doctor is not within hospital GPS coordinates, not using hospital WiFi, and not within working hours, then the doctor is not on duty and level of confidence is very low.

  [(latitude!=X,longitude!=Y)AND(8am>timeOfDay>5pm)AND (WiFiSSID!=HospitalWiFi)] ↪ {isOnDuty=False,ConfidenceLevel=VERYLOW}

Policies:

- P1:

```
if ((isDoctor(subject)=TRUE) AND
   (confidenceLevel=HIGH) AND
   (isOnDuty(subject)=TRUE)) then
          EnableRead() = True;
          EnableWrite() = True;
          EnableShare() = True;
```

- P2:

```
if ((isDoctor(subject)=TRUE) AND
    (confidenceLevel=MEDIUM) AND
    (isOnDuty(subject)=TRUE)) then
          EnableRead() = True;
          EnableWrite() = True;
          EnableShare() = False;
```

- P3:

```
if ((isDoctor(subject)=FALSE) OR
          (isOnDuty=TRUE) OR
          (confidenceLevel=LOW)) then
             EnableRead() = True;
             EnableWrite() = False;
             EnableShare() = False;
```

- P4:

```
if ((isDoctor(subject)=FALSE) OR
          (isOnDuty=FALSE) OR
          (confidenceLevel=VERYLOW)) then
             EnableRead() = False;
             EnableWrite() = False;
             EnableShare() = False;
```

The requirement describes what is to be achieved by the access control machine. It states that if the doctor is on duty and based on confidence level about his on duty status then he will be able to perform certain operations on a medical record. The policies are more specific than the requirement as they state what access rights will be granted to the doctor under different contextual conditions. Both the requirement and policies refer to the *isOnDuty* and *confidenceLevel* variables but they do not say how these variables are to be determined. The context

relates the conditions stated in the requirements to the actions stated in the policies by showing how the entities referenced in both requirements and specifications are derived from contextual attributes. This provides a *rich traceability* between the requirements and policies. For instance, the domain knowledge provided by context is necessary in order to explain what *isOnDuty* means in terms of causality for both the requirement and policy. Therefore, we need to make the context explicit as a way of linking requirements to their policies.

Causality Links, Logs, Domain Assumptions

In order to provide information, explanations and assurance to the user, we collect three types of information: causality links, data logs, and domain assumptions. Using our running example we illustrate the nature of these pieces of information.

Causality Links: The value of the Boolean variable *isOnDuty* and its *confidenceLevel* is determined by a combination of the values of three contextual variables GPS, WiFi, and TimeOfDay with varying levels of confidence for each combination as shown in Table 2. The confidence level can be VERYLOW, LOW, MEDIUM, or HIGH. The confidence level indicates the trust we have in the truth-value of the *isOnDuty* variable. This reflects the uncertainity we may have on the context.

The confidence in the *isOnDuty* variable depends of which sensors are available. For example, according to Table 2, if the GPS sensor is providing accurate coordinates and the doctor is connected to the hospital WiFi but not during working hours the confidence level is assign to HIGH. However, if none of the three sensors are available confidence level is assigned VERYLOW. Note that the ratings in Table 2 about confidence level would typically be formulated with the help of a domain expert. We convert the data in Table 2 into the EC causality links shown in Listing 1.

Table 2: Variable Confidence Determination from Context Variable Values

isWithin-GPSRange	isCorrect-SSID	isWorking-Hours	isOnDuty	Confidence Level
0	0	0	F	VERYLOW
0	0	1	T	LOW
0	1	0	T	LOW
0	1	1	T	MEDIUM
1	0	0	T	MEDIUM
1	0	1	T	HIGH
1	1	0	T	HIGH
1	1	1	T	HIGH

0 → False / 1 → True

The levels of confidence are mapped to three operations on a medical record: Read, Write, and Share as shown in Table 2. If the confidence level is very low the doctor is not allowed to perform any of the three actions. If the confidence level is low, the doctor is allowed to read medical record. If the confidence level is medium, he can only Read and Write a medical record. If the confidence level is high, he can perform all the operations.

Listing 1: isOnDuty Variable Causality Links

1.1 (!HoldAt(IsWithinGPSRange(),time) &!HoldAt(IsCorrectSSID(),time)&!HoldAt(IsWorkingHours(), time)) ↔ Happens(E1(),time).
1.2 Terminates(E1(), IsOnDuty, time).
1.3 (HoldsAt(IsWithinGPSRange(),time) | HoldsAt(IsCorrectSSID(),time) | HoldsAt(IsWorkingHours(),time)) ↔ Happens(E2(), time).
1.4 Initiates(E2(), IsOnDuty, time).

Listing 2: Confidence Level Causality Links

2.1 Initiates (E1(), VeryLowConfidenceLevel(), time).
2.2 HoldsAt(IsWithinGPSRange(),time)&(HoldsAt(IsCorrectSSID(),time) ⊕ HoldsAt(IsWorkingHours(),time))↔HappensAt(E2(),time).
2.3 Initiates(E2(), LowConfidenceLevel(), time).
2.4 HoldsAt(IsWithinGPSRange(),time)⊕HoldsAt(IsCorrectSSID(),time) ⊕ HoldsAt(IsWorkingHours(),time)↔Happens(E3(),time).
2.5 Initiates(E3(), MediumConfidenceLevel(),time).
2.6 HoldsAt(IsWithinRange(),time)&(HoldsAt(IsCorrectSSID(),time)| HoldsAt(IsWorkingHours(),time)↔Happens(E4(),time).
2.7 Initiates(E4(), HighConfidenceLevel(), time).

WHERE ⊕ is the XOR logical operator

Table 3: Matching Confidence Level to Operations

Confidence Level	Operations/Rights		
	CanRead	CanWrite	CanShare
VERYLOW	0	0	0
LOW	1	0	0
MEDIUM	1	1	0
HIGH	1	1	1

0 → Operation Not Allowed / 1 → Operation Allowed

Listing 3: Access Rights Causality Links

3.1 Happens(CanRead(), time) → !HoldsAt(VeryLowConfidenceLevel(), time).
3.2 Happens(CanWrite(), time) → HoldsAt(MediumConfidenceLevel(),time)& HoldsAt(HighConfidenceLevel(),time).
3.3 Happens(CanShare(), time) → HoldsAt(HighConfidenceLevel(), time).

The encoding of the data in the tables to EC is done by taking the input of the tables as the 'causing' event and the output of the table as the effect. For example in Table 2 the states of the three sensors are the inputs and the outputs are *isOnDuty* and confidence level. The first row in Table 2 says that if all three sensors are not available then *isOnDuty* is false and the confidence level is very low. To encode this condition in EC we first define e1 - an event that gets triggered when all sensors are not available. As long as the conjunction of the inputs indicated in row 1 (all sensors not available) is true then the event e1 keeps happening. The occurrence of e1 is indicated by 1.1 in Listing 1 and 1.2 says that the occurrence of event e1 results in *isOnDuty* being false. In

Listing 2, 2.1 says that the occurrence of e1 initiates the fluent *LowConfidenceLevel*. The rest of the rows in the tables are encoded in a similar way.

Tables 1 and 2 are relatively easier to read compared to their EC translations in Listings 1 and 2, respectively. Although this is the case we still have to do the encoding into causality links because they (causality links) are more amenable to automated reasoning with some of the existing tools of the EC.

Data Logs: In our approach, values of contextual attributes and policy decisions made are logged at runtime. A log of contextual attributes records the state of availability of the three sensors at a particular time instant. Policy decisions log is a record of which access rights were granted or denied at a given time instant.

Table 4: Policy Decisions Data Log

TimeStamp	Access Rights Granted		
	CanRead	CanWrite	CanShare
13:45:21	0	0	0
16:09:21	1	1	0
18:33:21	1	1	1
20:57:21	0	1	0
23:21:21	1	0	0
25:45:21	0	1	1
28:09:21	1	1	1
30:33:21	1	1	1
32:57:21	0	0	0
35:21:21	0	0	0
37:45:21	0	1	1
40:09:21	0	1	1
42:33:21	1	0	0
44:57:21	1	0	1
…….	……	……	……

0 → Operation Allowed / 1 → Operation Not Allowed

Table 5: Contextual Attributes State Data Log

Time Stamp	Contextual Attributes		
	isWithinGPS Range	isCorrect SSID	isWorking Hours
13:45:21	0	0	0
16:09:21	0	1	1
18:33:21	1	1	1
20:57:21	1	1	1
23:21:21	1	1	1
25:45:21	0	0	1
28:09:21	0	0	1
30:33:21	0	1	0
32:57:21	0	1	0
35:21:21	0	1	1
37:45:21	0	1	1
40:09:21	1	0	0
44:57:21	1	0	1
…….	…….	…….	…….

0 → False / 1 → True

Policy Decisions: When a policy decision is made either to permit or deny certain access rights we log this information in a policy decisions table. Table 4 is an example of a policy decisions table. According to this table at 16:09:21 the doctor was given the permission to Read and Write a medical record but was not allowed to share it.

Contextual Attributes: The log of contextual attributes keeps information about sensors that were available at different times. In our example the contextual attribute values are the state of the three sensors which can be used in determining the value of the *isOnDuty* variable and its confidence level. A sample sensor data log table is shown in Table 5. According to this table at 16:09:21 the GPS coordinates indicate that the doctor was in the hospital and he was also connected to hospital WiFi. But the time of day was not working hours.

Domain Assumptions: These are conditions that are assumed to be true for the relationship between W, S, and R in the entailment relationship to hold. The argument that the entailment relation holds depends on domain assumptions. These assumptions are assumed to be correct for the correct functioning of the machine specified by the behaviour in S because their control is not within the power of the machine. The assumptions in our example are as follows:

A1: The GPS device is well calibrated to give a correct and valid reading about the location of the doctor.
A2: The doctors always carry the GPS device.
A3: The hospital WiFi has a unique identifier, which makes it possible to uniquely identify it among other WiFi networks.
A4: The authentication device reads the doctor's credentials correctly.
A5: The file system where the medical record is kept has features for reading, writing, and sharing operations.
A6: The clock is set to the correct time of the day corresponding to its time zone
A7: The WiFi router has a unique serial number that enables the access control machine to verify that it is indeed the one that belongs to the hospital.

In the next section we illustrate how we use the causality links, data logs, and domain assumption to derive information for informing, explaining, and assuring the user about access control decisions at runtime.

5. Tracing Policy Decisions to Security Requirements

The tracing process includes three related processes for tracing policy decisions to security requirements, namely, informing, explaining, and assuring. Our approach includes a set of algorithms that take the different types of contextual data collected in section 4 to trace security decisions.

Starting with the denial of the security requirement r_s, at time t_0, a user u (e.g., Bob) wants to understand what's happening. The process includes three kinds of answers in different level of details. Firstly, the *informing* step (see Algorithm 1) classifies the traces into two categories, W_R and $W_{!R}$, depending on whether security requirement can or cannot be satisfied by these contexts. Using the causal traceability between the current policies s and the security

requirement r_s. we trace what events in s could lead to a the phenomena observed in r_s.

Data: A requirement $r \in R$, a specified policy $s \in S$; Causal traceability $W(\mathfrak{E}^+(s,r))$; and for a security requirement r_s, the traceability relation (s, r_s) does not hold.
Result: $W_R(s,r_s) = \{w | w, s \vdash r_s, w \in W\}$, $W_{\neg R}(s,r_s) = \{w | w, s \nvdash r_s, w \in W\}$.

$W_R(s,r_s) = \{\ \}$;
$W_{\neg R}(s,r_s) = W$;
for $c \hookrightarrow^+ e \in \mathfrak{E}^+(s,r_s)$ **do**
  $W_R(s,r_s) = W_R(s,r_s) \cup W_{c \hookrightarrow^+ e}$;
  $W_{\neg R}(s,r_s) = W_{\neg R}(s,r_s) \setminus W_{c \hookrightarrow^+ e}$;
end

Algorithm 1: Informing Contextual Domain Properties for Security Policy Decisions

There are many possible ways to solve this classification problem, e.g., by using the inductive learning procedure for deterministic classification, or using the statistical machine learning on supervised dataset for probabilistic classification. The informing contexts may help narrow the scope of search in the following explaining step.

For example, according to Table 3, if confidence level is LOW, one might not be able to READ, WRITE or SHARE the document. Using Table 3, the LOW confidence level is further traced to three possible situations for the three contextual variables (*isWithinRangeGPS, isCorrectSSID, isWorkingHours*): *(0, 0, 1)* and *(0, 1, 0)*. Note that in both cases *isWithinRangeGPS* is false, which indicates that $W_{!R}$ = *!isWithinRangeGPS* and either *!isCorrectSSID* or *!isWorkingHour*. In fact, because *!isWithinRangeGPS* is false in both cases, it is more informative to tell user whether *isCorrectSSID* or *isWorkingHours* is true or false. After knowing which contexts $W_{!R}$ may cause problems in satisfying r_s the system further fetches from the user the relevant domain properties $W_u(t_0)$ at the time t_0 from the data log, see Algorithm 2.

Data: $W_{\neg R}(s,r_s) = \{w | w, s \nvdash r_s, w \in W\}$ (see Algorithm 1); and for a security requirement r_s, the traceability relation (s, r_s) does not hold in the context $W_u(t_0)$ of user $u \in U$.
Result: 1) $w_0 \in W_{\neg R}(s,r_s) \cap W_u(t_0)$.

for $w_0 \in W_{\neg R}(s,r_s)$ **do**
  break if $w_0 \in W_u(t_0)$;
end

Algorithm 2: Explaining to User u about the Security Policy Decisions at time t_0

For example, by the time when all the permissions are denied in Table 4, i.e., 35h21m21s, the corresponding context domain properties in Table 5 concerning user Bob before that (at 32h57m21s) was *!isCorrectSSID* and *isWorkingHour*. Therefore, it is more informative to the user about the denial of access.

Knowing the situation, the user could react by changing his or her contexts in the next timestamp t_1. This leads to the new context relevant to the change $W_u(t_1)$. To provide assurance that it is possible to satisfy the security requirement r_s in the new context w_1, the system relies on a reasoning tool that evaluates the entailment relation.

Data: $W_R(s,r_s) = \{w | w, s \vdash r_s, w \in W\}$ (see Algorithm 1); and for a security requirement r_s, the traceability relation (s, r_s).
Result: Find a new context $w_1 \in W_R(s,r_s) \cap W_u(t_1), t_1 > t_0$.

for $w_1 \in W_R(s,r_s)$ **do**
  break if $w_1 \in W_u(t_1)$;
end

Algorithm 3: Assuring User u for proper adaptation to satisfactory security policy decisions

For example, we can use an event calculus reasoning tool to compute the abduction of the event sequences that may lead to this.

6. Evaluation and Discussion

The data collected in Section 4 helps us inform and explain adaptive behaviour as well as give assurance that their requirements are still being satisfied after an adaptation. The difference between informing and explaining is in the level of *information* provided. Informing tells the user why a particular access control decision has been taken. Meanwhile, *explaining* tries justifies the security decision by giving the reasons why it was denied with reference to the context of the user. Finally, *assurance* tells the user steps they can take to rectify the problem. We explain how each of these functions is fulfilled using the algorithms from Section 5 through an evaluation of hypothetical scenarios from the access control example presented in Section 2. We also discuss some of the limitations of our approach.

6.1 Evaluation

Informing: Recall in the motivating example that Bob has two devices: a laptop and an iPad. He always uses his iPad when accessing medical record in the hospital because it is portable to carry while attending to patients. The iPad has all three sensors GPS, WiFi, and a clock as a result Bob always get full access rights on medical records using his iPad. Due to a malfunction on his iPad Bob decides to use his laptop instead. On his laptop he is not able to share medical records with his fellow doctors. The access time is 16:09:21. He is not sure why this is the case.

The causality links in Listing 3 suggest that the sharing right is revoked if confidence level is VERYLOW, LOW, or MEDIUM. This information is derived from causality link 3.3, which says that it is only

possible to share a medical record when confidence level is high.

From the three causality links the doctor can be informed that the reason he is being denied sharing a record:

"You are not able to share the record because the system is not sure that you are on duty."

Explaining: As shown above, there are three possible reasons why Bob is being denied the right to share a record. To narrow down to the exact reason we look at the time the access control decision was taken and identity what was the state of the sensors at that time. According to Table 5 at 16:09:21 Bob was using the hospital WiFi, accessing the record within working hours, but was not within hospital premises according to his GPS device. According to 2.4 in Listing 2 this combination of sensors result into a MediumConfidenceLevel about his *isOnDuty* status.

This contextual attribute results in an event e3 whose occurrence makes confidence level to be medium.

2.4 HoldsAt(isWithinGPSRange,t)⊕HoldsAt(isCorrectSSID,t)⊕
HoldsAt(isWorkingHours,t)↔HappensAt(e3,t)
2.5 Initiates(e3, MediumConfidenceLevel,t)

Hence, an explanation message to the doctor could be something like:

"You are not allowed to share this medical record because you are currently not within hospital premises."

Using a combination of causality links and log data accurate information can be provided to the user on why a particular access control decision has been taken. When tracing a policy decision that involves a fluent being made false we inspect all *terminates* causality links involving the fluent. This helps in identifying all the possible events/causes that can make the fluent false. Similarly, when tracing why a fluent is true we inspect all the *initiates* causality links involving the fluent and identify all the events that make it true.

When there are multiple possibilities of events/causes log data is used to identify the time at which a policy decision was made and link this to the status of contextual attributes at the time. This eliminates causality links that do not apply and provides a more precise explanation of the security adaptation performed by the system.

Assurance: The traceability links can also be used to derive information on what steps the user can take to change the outcome of the behaviour of the adaptive application and in particular restore satisfaction of the requirement. The user may be interested to know what he needs to do for the application to grant certain access rights which are currently being denied or vice versa. In our example Bob wants to be advised on what he needs to do in order to regain the rights to share medical records.

Through the traceability links we have established that Bob is unable to share medical records because he is not in the right context, that is, he currently not within hospital premises according to his GPS device. To remedy this situation the system can advice Bob to change his context by relocating to the hospital. The assurance mechanism also needs to advice Bob to make sure that relevant domain assumptions are valid. For example, he needs to make sure that while changing location to the hospital he is still carrying (A2) a well calibrated GPS device (A1).

Changing context triggers an adaptive application to change its behaviour by invoking policies appropriate for the context. For adaptive applications for information security the user may be interested in the question of whether the application still satisfies its requirements after an adaptation. The traceability links are based on the entailment relationship. By (re-) evaluating the entailment relationship we can provide assurance that the applications still satisfies its requirements after an adaptation.

While the analysis and refinement of policies can ensure that policies correctly implement behavior that satisfies the user's security requirements, the highly dynamic contexts in which cloud services are used mean that their policies might not capture all possible security threats. To ensure that the adaptive application can detect when its security requirements are no longer being satisfied at runtime our approach includes requirements satisfaction information as part of the traceability links. Using the entailment relation, we can express the relationship between the requirements, context, and policies as W_s, $S_s \vdash R_s$. This states that the behavior of the policy S_s satisfies the requirement R_s given that some assumptions about the context W_s holds.

In our running example the security requirement is satisfied if the doctor supplies his credentials, a GPS device (carried by the doctor) supplies the correct location of the doctor, he is connected to the office WiFi, and the policy P_s behaves according to its specification. Provided the doctor has supplied valid credentials, his location according to the GPS device is such that he is within the premises of the hospital (i.e. he is on-site), then he should be allowed to read, write, and share the patients' medical record. Otherwise access must be denied. The argument that the security requirement will be satisfied is the *satisfaction argument*.

For our medical record access control problem, the satisfaction argument can informally and generally stated as follows:

"If the doctor supplies valid credentials, the doctor is carrying a GPS device, the coordinates from the GPS device are valid and say that the his location is within the premises of the hospital, he is accessing through hospital WiFi, the time of the day falls within working hours, then allow him to read, write, and share a medical record. Also allow the user to read, write, and share a medical record if he authenticates himself with credentials for an emergency procedure. Otherwise deny the access."

If the above argument is correct we can say that the security requirement is satisfied. The correctness or validity of the argument depends on a number of assumptions we make about the context and the ACM machine. The assumptions are stated in Section 4.

6.2 Discussion

Currently, our reasoning system is implemented on the Event Calculus reasoning tool where the rules are encoded systematically from the domain properties established from the monitors implemented and deployed on the target system. The general procedure proposed in Algorithm 1 needs to be made more efficient at runtime if the domain knowledge is to be exploited for the adaptive systems.

For example, the *Informing* step can be made more efficient by incrementally developing the causal relationships, or by statistical classifications. Incremental update of the data or knowledge structures is feasible because the changes to the physical contextual situations of a given user usually have some degree of continuity. In addition, when the size of the data log may be accumulated to exceed the capacity, it is possible to discard the earliest data logs as long as the learned classification structures are kept. However, statistical classification requires some training and collecting additional logs may help improve the accuracy.

The *Explaining* procedure may be enhanced further by preprocessing and customizing the contexts for individual users' attributes. For example as Bob's normal office locations are known beforehand, we can save the memory consumption for the domain properties and relationships. For example, condensing the consequent events that are in fact causing no changes to the fluents would not cause any loss to the capability of reasoning about the abnormal events. Therefore, providing abstract events to summarize these similar concrete events through a preprocessing step can help with the scalability of the reasoning algorithm.

The information presented by Tables 2 and 3 demonstrate a significant concept in the philosophy of our proposed approach to traceability. It demonstrates that in establishing traceability we need refinement in problem, context, and solution spaces. In the problem space, the requirement about Boolean variable *isOnDuty* has been refined to different confidence levels: VERYLOW, LOW, MEDIUM, and HIGH. The context has been refined to three variables *isWithinGPSRange*, *isWithinWorkingHours*, and *isCorrectSSID*. In the solution space the action of access to a medical record has been refined to operations for reading, writing, and sharing a medical record. Causality links are then established by relating the phenomena in each of the three (problem, context, and solution) sets of descriptions. It is these causality links that makes traceability from requirements to policies possible.

7. Related Work

The novelty of our work is in the use of rich traceability for self-explanation of security decisions in adaptive systems. We review the key related literature in these areas and compare with our work.

Self-Explanation and Diagnosis: With notable exceptions, such as Bencomo et. al.[7], we are not aware of any other approach to providing mechanisms for self-explaining the behaviour of an adaptive system to the user. While Bencomo et. el. propose a general approach to explaining emergent behaviour in adaptive systems, our approach focuses on explaining security decisions in adaptive systems. The focus on explaining security decisions brings with it several additional challenges. One of these challenges is that we need to make explicit the assets to be protected and make the explanations precise to the context of the user. For confidentiality reasons we also need to be careful about the content of our explanations to ensure that the system does not unnecessarily reveal information that may aid a potential attacker in breaching security. In order to address these challenges our approach uses traceability enriched with causality as a mechanism for establishing and reasoning about the relationship between requirements and policies to help understand system behaviour.

Our use of traceability as a tool to explain adaptation decision has similarities with the works on requirements monitoring [13], fault diagnosis [14], and root cause analysis [15]. Our approach to informing, explaining, and assuring is motivated by techniques developed from these areas of research. While these approaches are mainly designed for explaining why a particular fault, problem or emergent behaviour occurs, our approached goes a step further by providing assurance that a requirement is still being satisfied after an adaptation and suggesting ways in which requirements satisfaction can be restored.

Traceability Representation: Traditionally, traceability has been used as tool for supporting software maintenance activities [16][17][18][19]. In this tradition use of traceability, traceability links connect information contained between the artifacts being traced [20]. The links are often based on common keywords [18] between artifacts or numbering systems, such as requirement traceability matrixes [21][22], which are used to associate one artifact with another. While these links are effective in maintaining general explicit relationships between artefacts, they do not contain adequate semantic information that can be used for reasoning about the relationship between requirements (problems) and policies (solutions). For example, when using existing traceability links it is not possible to reason about whether a requirement is still satisfied by a given policy after a change in context. As our links are based on the entailment relation we are able to reason about requirements satisfaction using satisfaction arguments

[23][24]. The links we proposed can also be generalized into traceability rules in the same style as the requirement-to-object-model rules [25].

Attempts have been made to establish semantically rich traceability links [26][27]. However such links are not sufficient for explaining adaptive system behaviour. The links we have proposed in this paper involve capturing domain-specific and *intrinsic* information relating security requirements to policies through causality [11][5]. This representation of links has been motivated by the observations that security requirements are stated in terms of the conditions that need to be true to protect assets from harm, and security policies are expressed in terms of the actions that need to be performed in order to satisfy the conditions stated by security requirements [6][28]. Based on these observations we specify traceability links in terms of causality relationships between conditions stated in the requirements and the actions expressed in policies. This provides richer traceability links.

The problem of relating a security requirement, to policies, and the contextual conditions in which those policies should be enforced is similar to the traceability problem in software product lines [29][30][31]. While for software product lines traceability is about how to relate the various documents produced for the different product variants, our approach for traceability in adaptive applications involves explicitly exposing the contextual conditions that an application depends on as part of the traceability between the requirements and policies. Explicit expression of the contextual conditions/attributes in traceability links is useful for adaptive applications as it helps in reasoning and deriving explanation of adaptive behaviour.

Our representation of traceability links with entailment relations is similar to the idea of knowledge representation for self-adaptive system behaviour [32]. Reasoning on the knowledge is used to establish connection between knowledge, perception, and actions that realize self-adaptive behaviour. Their approach aims at logging execution traces so that the adaptive system can remember where it failed. Our approach is aimed at reasoning on the model representation of an adaptive system in order to explain its behaviour to the user. We are not the first to use the Event Calculus for representing and reasoning about causality. Galton [33] used the EC in causal reasoning for alert generation in smart homes. In our approach we use the EC in a similar way – as formalism for handling the manner in which certain conjunctions of independent states (such as readings from sensors) can be used as trigger of dependent states.

Bruni et. al. [34] proposed a causality framework for allocating new events and relating their causes. They achieve this by modelling relationship between processes and causal relations among the processes' events. Their cause-effect relation approach can be useful in future development of our approach to establish previously undefined causality relations. The causality relations we use in our approach are predefined. The dynamic definition of the causality relations can make the goal of run-time traceability more feasible.

For software maintenance bi-directional transformations [35] have been proposed as a means to establishing and maintaining traceability [36]. With these transformations separate traceability links are required for forward and reverse traces. With our approach a single traceability link can be used for both forward and reverse trace tasks. This is made possible by the fact that our traceability links contain references to entities in the requirements, context, and policies. The satisfaction arguments [23] explicitly relate the requirements to the policies through domain assumptions in context. In this way regardless of what changes occur in the context, the relationship between the requirements and policies is maintained. Our approach also has the potential to allow for a 'live' validation of this relationship by providing the capability of re-evaluating entailment relationships at run-time.

8. Conclusions and Further Work

Information security for applications used in varying contexts need to be adaptive in order maintain satisfaction of security requirements. With such adaptive information security the applications need to give *assurance* about the security requirements they are satisfying as well as *information* about and *explanation* of the access control decisions taken at runtime. We proposed traceability as a mechanism for enabling adaptive applications to provide information, explanations, and assurance about access control decisions taken at runtime. We used traceability as a way of understanding the relationship between security requirements and the policies that enforce those requirements. Our traceability links are rooted in entailment relations – capturing the links with causal relationships described in domain descriptions. Through our traceability links we were able to explain the rationale for access control decisions and reason about what access control decisions would result from a change in contextual attributes at runtime.

We plan to investigate how traceability can be further improved for better self-explanation. In particular, the content of the messages given by our approach for helping the user understand adaptive behavior needs to be crafted with great caution. We believe that traceability can be used as a mechanism for scoping policies to be selected for adaptation to a context that was not completely understood at design time. The feasibility of this idea is being investigated. We plan to capture the runtime event traces of a real-world access control application and map each event in a trace to its effect using causality information. By so doing we hope to identify abnormal behavior that leads to the violation of a security requirement. Using supervised learning mechanisms [37] the adaptive application may then be trained on new behaviors that can be applied to either weaken the requirement or strengthen it to prevent further violation. We may use techniques such as inductive learning procedures [38] to generate

explanations of how the system may be changed to comply with user requests.

Acknowledgements.

We thank Michael Jackson for his insightful review that enriched the ideas in this paper. This work was made possible by the support of a grant (NPRP 05-079-1-018) from the Qatar National Research Fund (QNRF). The statements made herein are solely the responsibility of the authors.

References

[1] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grunbacher, and G. Antoniol, "The quest for Ubiquity: A roadmap for software and systems traceability research," presented at the Requirements Engineering Conference (RE), 2012 20th IEEE International, 2012, pp. 71–80.

[2] I. Santiago, Á. Jiménez, J. M. Vara, V. De Castro, V. A. Bollati, and E. Marcos, "Model-Driven Engineering as a new landscape for traceability management: A systematic literature review," *Inf. Softw. Technol.*, vol. 54, no. 12, pp. 1340–1356, Dec. 2012.

[3] M. Jackson and P. Zave, "Deriving Specifications from Requirements: an Example," presented at the 17th International Conference on Software Engineering, 1995. ICSE 1995, 1995, pp. 15–15.

[4] O. C. Z. Gotel and A. C. W. Finkelstein, "An analysis of the requirements traceability problem," in , *Proceedings of the First International Conference on Requirements Engineering, 1994*, 1994, pp. 94–101.

[5] R. Scherl and G. Shafer, "A logic of action, causality, and the temporal relations of events," in *Fifth International Workshop on Temporal Representation and Reasoning, 1998. Proceedings*, 1998, pp. 89–96.

[6] P. Zave and M. Jackson, "Four Dark Corners of Requirements Engineering," *ACM Trans Softw Eng Methodol*, vol. 6, no. 1, pp. 1–30, Jan. 1997.

[7] N. Bencomo, K. Welsh, P. Sawyer, and J. Whittle, "Self-Explanation in Adaptive Systems," in *2012 17th International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2012, pp. 157–166.

[8] P. Zave and M. Jackson, "Four Dark Corners of Requirements Engineering," *ACM Trans Softw Eng Methodol*, vol. 6, no. 1, pp. 1–30, Jan. 1997.

[9] M. A. Jackson, *Problem frames and methods: structuring and analyzing software development problems*. Harlow: Addison-Wesley, 2000.

[10] M. Jackson, *Problem Frames: Analyzing and Structuring Software Development Problems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[11] L. Rapanotti, J. G. Hall, and Z. Li, "Deriving specifications from requirements through problem reduction," *Softw. IEE Proc. -*, vol. 153, no. 5, pp. 183–198, Oct. 2006.

[12] E. T. Mueller, "Event calculus and temporal action logics compared," *Artif. Intell.*, vol. 170, no. 11, pp. 1017 – 1029, 2006.

[13] M. S. Feather, S. Fickas, A. van Lamsweerde, and C. Ponsard, "Reconciling system requirements and runtime behavior," in *Ninth International Workshop on Software Specification and Design, 1998. Proceedings*, 1998, pp. 50–59.

[14] G. Carrozza, D. Cotroneo, and S. Russo, "Software Faults Diagnosis in Complex OTS Based Safety Critical Systems," in *Dependable Computing Conference, 2008. EDCC 2008. Seventh European*, 2008, pp. 25–34.

[15] M. Leszak, D. E. Perry, and D. Stoll, "A case study in root cause defect analysis," in *Proceedings of the 2000 International Conference on Software Engineering, 2000*, 2000, pp. 428–437.

[16] M. . Hirzalla, A. Zisman, and J. Cleland-Huang, "Using Traceability to Support SOA Impact Analysis," in *2011 IEEE World Congress on Services (SERVICES)*, 2011, pp. 145–152.

[17] M. Mirakhorli and J. Cleland-Huang, "Using tactic traceability information models to reduce the risk of architectural degradation during system maintenance," presented at the 2011 27th IEEE International Conference on Software Maintenance (ICSM), 2011, pp. 123–132.

[18] G. Bavota, A. De Lucia, R. Oliveto, and G. Tortora, "Enhancing software artefact traceability recovery processes with link count information," *Inf. Softw. Technol.*, vol. 56, no. 2, pp. 163–182, Feb. 2014.

[19] K. Welsh and P. Sawyer, "Requirements Tracing to Support Change in Dynamically Adaptive Systems," in *Requirements Engineering: Foundation for Software Quality*, M. Glinz and P. Heymans, Eds. Springer Berlin Heidelberg, 2009, pp. 59–73.

[20] O. C. Z. Gotel and A. C. W. Finkelstein, "An analysis of the requirements traceability problem," presented at the , Proceedings of the First International Conference on Requirements Engineering, 1994, 1994, pp. 94–101.

[21] S. Soonsongtanee and Y. Limpiyakorn, "Enhancement of requirements traceability with state diagrams," in *2010 2nd International Conference on Computer Engineering and Technology (ICCET)*, 2010, vol. 2, pp. V2–248–V2–252.

[22] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, 2010, vol. 1, pp. 95–104.

[23] E. Kang and D. Jackson, "Dependability Arguments with Trusted Bases," in *Requirements Engineering Conference (RE), 2010 18th IEEE International*, 2010, pp. 262–271.

[24] C. B. Haley, R. Laney, J. D. Moffett, and B. Nuseibeh, "Security Requirements Engineering: A Framework for Representation and Analysis," *IEEE Trans. Softw. Eng.*, vol. 34, no. 1, pp. 133–153, Jan. 2008.

[25] G. Spanoudakis, A. Zisman, E. Pérez-Miñana, and P. Krause, "Rule-based generation of requirements traceability relations," *J. Syst. Softw.*, vol. 72, no. 2, pp. 105–127, Jul. 2004.

[26] H. Schwarz, J. Ebert, J. Lemcke, T. Rahmani, and S. Zivkovic, "Using Expressive Traceability Relationships for Ensuring Consistent Process Model Refinement," in *2010 15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2010, pp. 183–192.

[27] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *25th International Conference on Software Engineering, 2003. Proceedings*, 2003, pp. 125–135.

[28] W. Pieters, T. Dimkov, and D. Pavlovic, "Security Policy Alignment: A Formal Approach," *IEEE Syst. J.*, vol. 7, no. 2, pp. 275–287, Jun. 2013.

[29] S. . Ajila and A. Kaba, "Using traceability mechanisms to support software product line evolution," in *Proceedings of the 2004 IEEE International Conference on*

Information Reuse and Integration, 2004. IRI 2004, 2004, pp. 157–162.

[30] L. C. Lamb, W. Jirapanthong, and A. Zisman, "Formalizing Traceability Relations for Product Lines," in *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, New York, NY, USA, 2011, pp. 42–45.

[31] W. Jirapanthong and A. Zisman, "XTraQue: traceability for product line systems," *Softw. Syst. Model.*, vol. 8, no. 1, pp. 117–144, Feb. 2009.

[32] E. Vassev, M. Hinchey, and B. Gaudin, "Knowledge Representation for Self-adaptive Behavior," in *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, New York, NY, USA, 2012, pp. 113–117.

[33] A. Galton, "Causal Reasoning for Alert Generation in Smart Homes," in *Designing Smart Homes*, J. C. Augusto and C. D. Nugent, Eds. Springer Berlin Heidelberg, 2006, pp. 57–70.

[34] R. Bruni, U. Montanari, and M. Sammartino, "Revisiting causality, coalgebraically," *Acta Inform.*, pp. 1–29, Oct. 2014.

[35] S. Hidaka, Z. Hu, K. Inaba, H. Kato, K. Matsuda, and K. Nakano, "Bidirectionalizing Graph Transformations," in *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming*, New York, NY, USA, 2010, pp. 205–216.

[36] Y. Yu, Y. Lin, Z. Hu, S. Hidaka, H. Kato, and L. Montrieux, "Maintaining invariant traceability through bidirectional transformations," in *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 540–550.

[37] C. S. G. Lee and C.-T. Lin, "Supervised and unsupervised learning with fuzzy similarity for neural-network-based fuzzy logic control systems," in , *IEEE International Conference on Systems, Man and Cybernetics, 1992*, 1992, pp. 688–693 vol.1.

[38] A. S. d' Avila Garcez, A. Russo, B. Nuseibeh, and J. Kramer, "Combining abductive reasoning and inductive learning to evolve requirements specifications," *Softw. IEE Proc. -*, vol. 150, no. 1, pp. 25–38, Feb. 2003.