Hindawi Publishing Corporation
Mobile Information Systems
Volume 2016, Article ID 8743504, 16 pages
http://dx.doi.org/10.1155/2016/8743504

Research Article

On the Modelling of Context-Aware Security for Mobile Devices

Tomasz Zurek,[1] Michail Mokkas,[2] and Bogdan Ksiezopolski[1]

[1]*Institute of Computer Science, Maria Curie-Sklodowska University, Pl. M. Curie-Sklodowskiej 5, 20-031 Lublin, Poland*
[2]*Polish-Japanese Institute of Information Technology, Koszykowa 86, 02-008 Warsaw, Poland*

Correspondence should be addressed to Bogdan Ksiezopolski; bogdan.ksiezopolski@acm.org

Received 23 May 2016; Revised 15 August 2016; Accepted 31 August 2016

Academic Editor: Juan A. Gomez-Pulido

Security management in wireless networks has to deal with the changing character of the environment, which can further lead to decision making problem for unexpected events. Among a huge list of devices, the mobile ones are especially vulnerable to this situation. The solution for adapting systems and applications to dynamic environments can be context-aware description of the user actions, which gives a possibility to take into account the factors that influence these actions. In the article, we propose a context-aware security adjusting model, which is based on proposition logic and incorporates mechanisms that assist in the reasoning process. The main benefits that differentiate our approach from similar ones are a formal representation of the model, the usage of the whole spectrum of context attributes, the detection and analysis of contextual data integrity, and conflicting rules' eradication capability. All these traits transcribe into a more effective way of adjusting security measures in accordance with existing circumstances. To illustrate the proposed approach, we present the case study of context-aware security management for mobile devices.

1. Introduction

Nowadays, a lot of effort is put into providing new methods of ensuring appropriate security for mobile systems and applications. The most popular approach is to apply the strongest possible security measures, which guarantee maximal security level. This approach, however, leads to an increase in system load and hinders its performance [1, 2]. The system can become unusable, causing a decrease in the quality of experience (QoE) perceived by users. Therefore, finding a balance between the security level and performance of an IT system is an essential goal that we seek to achieve. For this reason, we investigate the benefits of adaptable security [3]. In order to be adaptable, a security mechanism needs to take into consideration a lot of additional parameters that can influence the outcome of a security evaluation. These additional parameters can be categorized as contextual data and other facts that describe system. In other words, a feature called context-awareness (CA) has to be introduced. CA is a term coined mostly for the capability of pervasive computing systems, which allows acting in certain ways according to already gathered sensor data. The sensor data constitute what we define as context. Mobile devices currently have limited resources (such as energy supply, computational power, or memory) but are equipped mostly with a vast amount of sensors. This is what makes them an ideal environment for adaptable context-aware security mechanisms, which attempt to find the optimal level of quality of protection (QoP) and performance ratio [4, 5].

Security mechanisms should be adjusted following the importance of the resources processed by the device, its parameters, and external circumstances in which the given device works. Adjustment of quality of protection mechanisms to the context in which the device works should contain a couple of important steps. One of these steps is a full evaluation of quality of protection mechanisms; another one is an analysis of the context, in which the given device operates resources, which are processed by the device, its technical parameters, and all external circumstances which influence the security of the device resources. Analysis of the context should result in the definition of the quality of protection requirements, which should be compared to previously prepared evaluation of quality of protection mechanisms. If all requirements are met by the device's security mechanisms, then a chosen activity can be performed. It has to be noted that such adaptable mechanisms have already

been in the focus of many researchers, and some of them are described in greater detail in the next section. However, the main contribution of our paper is the addition of certain elements in the process of security evaluation based on a model of propositional logic. The main elements, which we introduced and that are missing from similar approaches, are

(i) a context consistency analysis mechanism, which investigates whether provided contextual data is correctly interpreted and obtained,

(ii) a conflicting rule reasoning eradication mechanism, to deal with issues that might occur during the reasoning process,

(iii) a holistic approach, which takes into account all contextual factors rather than just a couple of them,

(iv) a formal representation of the model with definitions of its elements and detailed reasoning algorithms.

The article is divided into six sections. Section 1 is an overall introduction to the subject of context-aware security. Section 2 focuses on the comparison of similar works, briefly summarizing their main shortcomings which we intend to rectify. Section 3 introduces our model and familiarizes with the definitions of its elements. Section 4 describes in detail the algorithm of the procedure, while Section 5 illustrates a couple of use case scenarios to thoroughly present the reasoning process. Additionally, Section 6 briefly describes the developed implementation of the model that acts as a proof of concept and finally concludes the article. In the Appendix, the diagrams of the proposed algorithms are presented.

2. Related Work

This section describes the state-of-the-art ideas and concepts which are related to the field of study. There are quite many existing models that describe context-awareness in systems. However, most of them lack mechanisms that allow dynamic reasoning capable of contextual consistency analysis and coping with conflicting rules. One of them is [6] which describes a context-awareness framework (webinos) allowing webinos-enabled applications to collect and provide contextual information. This information is used to enhance multiple aspects related to human-machine interactions of everyday life instead of being limited only to a device-centered model for context deduction. There is not, however, much detail provided about the mechanism of rule forming and evaluation that our approach relies on.

Another article [7] proposes a context-aware security framework which enforces the execution of mobile applications inside security incubators for the purpose of controlling the exchange of information between mobile applications and mobile device resources. The contextual data is gathered from mobile devices equipped with sensors. Whenever an application requests access, the framework analyzes the contextual information and security configuration to decide on policy enforcement. The method used for this analysis is the Analytic Hierarchy Process (AHP), which is a popular choice for policy selection, decision making, adaptive learning, and recommendation and feedback systems. The main advantage of AHP is its ability to break down complex decision problems into smaller ones, which enhances the reasoning process. Unfortunately, the article does not mention any conflict eradication mechanisms.

In [8], the authors describe a new emerging research area, referred to as the Context-Aware Mobile and Wireless Networking (CaMoWiN). The article is basically a survey of existing solutions from both networking and computing environments, where context acts as a form of a "glue-point" that allows for their integration. As part of this work, which deals with context evaluation (policies, decision making), several frameworks are described (through omitting any conflict solving mechanisms).

The article [9] introduces a context-aware scalable authentication (CASA) as a means of balancing security and usability for authentication. The idea is to perform authentication by selecting a form of active authentication based on a combination of multiple passive factors, some of which are contextual (such as location). This procedure uses a probabilistic framework as well as a naive Bayes classifier to combine many factors, calculating risk and determining the required level of active authentication. As with previous solutions, there are no conflict eradication or context consistency mechanisms involved.

The article [10] describes a context-aware security and trust framework for Mobile Ad hoc Networks (MANET) called CAST. Its primary objective is to accurately classify whether the nodes in MANETs that misbehave do so because of a faulty or malicious cause. The main addition of this framework is that it takes into consideration contextual data, such as communication channel status, battery status, or weather conditions. In addition, it uses policies to detect whether the contextual information is intentionally falsified or the current environmental conditions cause their reporting to malfunction and provide incorrect data. Unfortunately, the article does not mention any methods of detecting context inconsistencies.

In [11], the authors proposed a cloud-oriented context-aware framework for ambient assisted living (AAL) called CoCaMAAL. Their model tries to address issues such as the complexity in management of sensor data or the derivation of contextual information, as well as the constant monitoring of the activities that are performed by users. The procedure is enhanced through the implementation and use of a context management system (CMS). Similar to other works, it lacks conflicting rule and context consistency analysis mechanisms.

In [12], an automated context classification for context-aware access control (ConXsense) is proposed. The authors claim this is the first framework to provide context-awareness in access control of mobile devices which is based on context classification. Previous solutions relied too much on statically predefined policies. ConXsense is a probabilistic approach that overcomes previous deficiencies. Context is automatically classified through machine learning and context sensing relying on their appropriate security properties. The approach

does not involve capabilities for solving rule conflicts during its reasoning.

The paper [13] introduces an intelligent context-aware system (iConAwa) providing mobile users with the ability to obtain information and services related to their current location. It describes the context and point of interest ontologies in OWL. One of the similarities with other systems is the utilization of rule-based context reasoning. However, it lacks a formal representation.

An approach proposed in [14] uses context related preferences to personalize various procedures (such as queries). It deals with the problem of identification of preferences that have the most similar context states with those of a chosen query (context resolution). The authors provide a solution for the problem in the form of a preference graph and profile tree data structures, which decreases the complexity of representation of the context related preferences. This article does not incorporate any method for dealing with conflicting rules.

In the paper [15], a context-aware RBAC model for business processes is introduced. Previous solutions related to business processes did not support any contextual constraints. The context constraints specify the conditions that have to be fulfilled in order to allow the execution of a certain task. Unfortunately, there is no mention of any conflicting rules solving mechanisms.

In [16], two automated analysis mechanisms which examine modelling errors in contextual requirements models are proposed. The first one deals with the detection of inconsistent specification of contexts in a goal model (used in early stages of software development), while the second one deals with the detection of conflicting context changes. The proposed model is in some points similar to mechanisms introduced by us (especially in the point of detection of inconsistent facts). Generally speaking, the model presented in [16] is much more complex than ours, but the mechanisms introduced are created to support a goal-driven software development. Therefore, the system lacks all the mechanisms connected with the evaluation of the context of the environment in which the device works, especially the mechanism of defeating of conflicting requirement rules. Moreover, also the concept of requirements which appears in the paper has a different meaning than in our model. Most notably, the aspect of taking into consideration quality of protection is missing. Also, there are some differences in the specifics of the implemented mechanisms such as ordering of rules instead of facts to decide about actions.

Another article [19] describes a context-aware and rule-based reasoning solution for mobile devices. Context is represented by a key-value pair, where the key is the name which defines a context property and the value represents its current state. An example of such pair would be (room, kitchen). The article compares common approaches to context modelling and concludes that most available languages offer limited support for advanced inference tasks. The authors also claim that rules are the best choice for advanced context reasoning. The problem of inconsistent contextual data is not dealt with.

The paper [20] presents a context-aware security policy framework for user preferences in e-services. It supports role-based access control and its aim is to create policy-based security measures in cross-organizational scenarios. One of the advantages of the framework is the integration of contextual information and user preferences, which increases flexibility. It incorporates policy conflict resolution mechanisms. However, the authors do not mention detecting inconsistent contextual data.

Lastly, in [21], the authors provide a survey on energy-aware security mechanisms. The discussed articles and solutions are not completely context-aware, since they take into consideration only the energy consumption related aspects. The aspects of conflict prevention and context consistency analysis mechanism are unfortunately omitted and therefore assumed nonexistent.

The models in the mentioned related work are compared with regard to the following main attributes:

(i) "Context consistency analysis" refers to the ability of detecting inconsistent context specification.

(ii) "Conflicting rule reasoning" stands for the ability of dealing with contradictory specified rules.

(iii) "Context-awareness" refers to the acquisition and use of contextual information.

(iv) "Formal representation" refers to whether the model is presented by the use of mathematical formulae.

(v) "Holistic" specifies the possibility of using all contextual factors instead of a set of selected ones.

(vi) "Considers security requirements" indicates that the model takes into account the security aspects of the system.

The comparison is presented in Table 1.

3. The Model

3.1. General View. Our model is mainly based on proposition logic with some additions, which allows for a better representation of specific features of the security context evaluation.

3.2. Facts and Rules. We assume that the system context is represented by means of a set of propositions, which are referred to as facts:

$$F = \{f_1, f_2, f_3, \ldots, f_n\}, \tag{1}$$

where $f_1, \ldots, f_n$ are the facts describing a system.

The facts represent circumstances in which the analyzed device works and they can have various levels of generality.

We assume a set of operators OP $= \{\neg, \sim, \vee, \wedge, \Rightarrow, \rightarrow\}$, where

(i) $\neg$ is a classical (strong) negation;

(ii) $\sim$ represents a negation as failure;

(iii) $\vee$ stands for a disjunction;

(iv) $\wedge$ serves as a conjunction;

(v) $\Rightarrow$ is a defeasible implication;

(vi) $\rightarrow$ means a strict implication.

Table 1: Model comparison.

	[6]	[7]	[17]	[8]	[9]	[10]	[11]	[18]	[12]	[13]	[14]	[15]	[16]	[19]	[20]	Our
Context consistency analysis	—	—	—	—	—	—	—	—	—	√	—	—	√	—	—	√
Conflicting rule reasoning	—	—	—	—	—	√	—	—	—	√	—	—	√	√	√	√
Context-awareness	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
Formal representation	—	—	—	—	√	√	—	—	√	—	√	√	√	—	√	√
Holistic	√	√	√	√	—	—	√	√	√	√	√	√	√	√	√	√
Considers security requirements	√	√	—	—	√	√	—	√	—	—	—	—	—	√	—	√

Table 2: The list of main security attributes.

Integrity	Prevention against improper information modification
Confidentiality	Guarantee of only authorized information access and disclosure
Authentication	This is the process of verifying or testing that the claimed identity is valid
Authorization	Ensures that the requested activity or object access is possible given the rights and privileges assigned to the authenticated identity
Accountability	Steps of protocols (access to services) are registered to restore past threats
Availability	Ensuring timely and reliable access to services and data and use of information

By negation as failure, we understand the kind of negation used in the PROLOG language: $\sim p$ is satisfied if it is impossible to prove that p.

By defeasible implication, we understand a weaker, challengeable kind of implication, widely used in formal models of argumentation (e.g., in [22]): the utilization of defeasible implications allows for defeating one of the conflicting formulae during the inference process.

Definition 1 (literals). Facts (negated in a strict way or nonnegated) are literals. A set of all literals is denoted as $L = \{l_1, l_2, \ldots, l_m\}$.

For example, if $F = \{f_1, f_2\}$ is a set of facts, then $L = \{f_1, f_2, \neg f_1, \neg f_2\}$ is a set of literals.

Definition 2 (security attribute). Security attribute is an attribute which describes the system behavior in the case of information security requirements.

For example, one can enumerate the following security attributes (Table 2): confidentiality, integrity, availability, authentication, authorisation, or accountability [3, 23].

The security attributes (SA) set consists of unlimited but finite number of security attributes.

3.3. Security Attributes. As described above, the security attributes set SA consists of an unlimited but finite number of security attributes. Each of them has its own evaluation value expressed by a positive integer number. The measurement process of security attributes, referred to as an evaluation, is discussed in [24].

Definition 3 (evaluation of security attributes). S is a set of pairs $O = \langle \text{sa}, o \rangle$, where $\text{sa} \in \text{SA}$ is a security attribute and o is its evaluation value.

For instance, we have three security attributes with their evaluation values:

$$\text{SA} = \text{confidentiality, integrity, authorisation;}$$

$$S = \{(\text{confidentiality}, 10), (\text{integrity}, 20), (\text{authorisation}, 30)\}.$$

By $\text{Val}(S, \text{sa}_m)$ we denote function which returns evaluation of a given security attribute. For example, Val(S, confidentiality) = 10.

It is important to notice that security attribute can have positive or negative character, in the sense that the bigger value of security attribute evaluation can mean better (for positive) or worse (for negative) evaluation.

Definition 4 (context). A context is a model of an external and internal environment in which evaluated system works and is represented by a set of literals, denoted as $C = \{l_a, l_b, \ldots, l_s, l_z, \ldots\}$, which can also be expressed by a set of positive or negated facts: $C = \{f_a, f_b, \ldots, \neg f_s, \neg f_z, \ldots\}$.

Definition 5 (rule). The rule is a formula in the form of

$$\text{Conditions} \rightarrow \text{Conclusion},$$

where

(i) Conditions is a list of rule conditions.

List of conditions is in the following form: wl_a **func** wl_b **func**$\cdots wl_d$, where **func** are the operators from the set $= \{\vee, \wedge\}$, and $\{wl_a, wl_b, \ldots, wl_d\}$ are the literals (nonnegated or negated by a classical negation).

(ii) Conclusion is a rule conclusion in the form Conclusion $= (l_x \wedge l_y \wedge \cdots)$, where $(l_x, l_y, \ldots) \in L$.

Conditions and conclusions can be negated by a classical (logical) negation. It is forbidden to use negation as failure, which allows for preservation of monotonicity of inference. The set of rules is denoted as RF.

The rules allow us to represent relations between various facts. They also allow us to express which facts are exclusive

in the sense that the existence of one of them causes the inexistence of the others. It is important because it helps to preserve consistency of the model.

Definition 6 (fact-based inference mechanism). As a fact-based inference mechanism, one understands a forward chaining mechanism. As C' one denotes a set of conclusions whose inference mechanism concludes from a context C and a set of rules RF. One can also denote it as $C \vdash C'$. The union of sets $C \cup C'$ is denote as P and describes the full context.

Let us illustrate the mechanism with an example.
Set RF contains the following rules:

$r1$: $f_1, f_6 \rightarrow f_2$.
$r2$: $f_2 \rightarrow \neg f_3$.
$r3$: $f_4 \rightarrow f_5$.
$r4$: $f_7 \rightarrow f_8$.
$r5$: $f_5 \rightarrow f_3$.
$r6$: $f_3 \rightarrow \neg f_2$.

Set C contains facts: $C = \{f_1, f_4\}$.

The steps of the forward chaining inference mechanism will be as follows:

(1) In the first step the mechanism checks if the conditions of rule $r1$ are satisfied. Since f_6 is not declared ($(C \cup C') \nvdash f_6$), the mechanism skips the rule and moves to the next step.
(2) Since $(C \cup C') \nvdash f_2$, the mechanism skips rule $r2$ and moves to the next step.
(3) On the basis of rule $r3$, the mechanism infers f_5 and adds f_5 to set C'.
(4) Since the condition of rule r_4 is not satisfied ($(C \cup C') \nvdash f_7$), the engine does not add f_8 to set C' but moves to the next step.
(5) On the basis of rule $r5$, the mechanism infers f_3 and adds f_3 to set C'.
(6) On the basis of rule $r6$, the mechanism infers $\neg f_2$ and adds $\neg f_2$ to set C'.
(7) The system moves to rule $r1$ and checks if its conditions are satisfied.
(8) The system checks the remaining rules.
(9) Since there are no possibilities to satisfy the conditions of any other rule (except $r3$, $r5$, and $r6$), the system returns $P = \{C \cup C'\}$.

In the case of inconsistent input data, the possibility of utilization of negation in a rule base RF entails the possibility of inconsistencies in set P and infinite loops during the inference (by an inconsistency we understand situation in which $P \vdash f_x \wedge \neg f_x$). How do we overcome these problems and interpret such inconsistencies? If we assume that rules from a set RF are well formed and they represent real life dependencies, then inconsistencies in a set C or P suggest that something is wrong with the sensors or the device has been hacked and someone is trying to take control over the device. In both situations, the device should alarm the user that something is wrong and start safety procedure which should increase the level of security of the device or switch the device into the offline mode. In order to detect and prevent the system from such suspicious situations, we assume the existence of the consistency guard, the module which controls consistency of the contextual data collected by our device. A more detailed description of the consistency guard will be provided the next section.

3.4. Consistency Guard. A mobile device with context-aware security system can be an object of an attack targeted to deceive sensors of the system. Such an attack may lead to a decrease in the level of quality of protection by deceiving sensors in order to convince the device that the context it works in is safe enough to decrease QoP level.

How do we overcome such risk? Obviously, deceiving all sensors is much more difficult and less plausible than deceiving only one of them. The attack on one of the sensors can lead to inconsistency in the indications of the device sensors. For example, GPS shows that the device is in the safe place (office), but the temperature is too low (or too high) for the interior of the office. Moreover, the device can connect to access points which cannot be accessed inside the office. Such inconsistency can suggest that the device has been hacked and someone changed indications of GPS positioner. The key point of detection of such kind of attack lies in the analysis of the consistency in indications of the device's sensors.

Consistency guard will be a module of the device controlling whether the sensors indicate facts which cannot be true simultaneously. If coexistence of such facts appears, then the system should indicate a dangerous state to the user.

From a formal point of view, if $P \vdash f_n \wedge \neg f_n$, then there is an inconsistency in our sensors' indications and our device should alarm the user and stop all actions, except the possibility of direct switching the guard off by the user (in the case of mistaken alarm). The system should not perform the rest of context analysis until the conflict disappears or the user switches the guard off (procedure ALARM).

Unfortunately, such model of inconsistency detection is not sufficient; if such incompatible literals appear during the inference process, then inference engine can fall into an infinite cycle before the reasoning process terminates. How do we overcome such a problem? We have assumed the usage of a classical, monotonic, forward chaining, modus ponens-based inference engine. Such an engine can fall into infinite loop in the case of cycles in the set of rules. Such a case can be reduced to a situation in which RF will contain two rules:

$f_1 \rightarrow \neg f_2$.
$f_2 \rightarrow \neg f_1$.

If $C \vdash f_1, f_2$, then the inference engine will fall into an infinite loop. The monotonicity of our inference engine gives us an important property: if a set of rules (RF) is well formed and input data (a list of facts) is consistent, then the conclusion of every step of inference should be consistent with the declared and previously inferred list of facts. Thereby, in order to

overcome the possibility of falling into infinite loops, we have to detect inconsistencies in the conclusions of every step of reasoning. For example, if $C \vdash f_1, f_2$, then the first step of the inference engine will add to the set C' fact $\neg f_2$. In the next step, the consistency guard tests if there is inconsistency in sets $C \cup C'$. Since C contains f_2 and C' contains $\neg f_2$, then there is inconsistency; the inference engine interrupts its functioning and consistency guard starts the ALARM procedure.

Let us illustrate the mechanism with a more complex example:

Consider set RF from the previous section (RF = $\{r1, r2, r3, r4, r5, r6\}$), where

$r1$: $f_1, f_6 \rightarrow f_2$,
$r2$: $f_2 \rightarrow \neg f_3$,
$r3$: $f_4 \rightarrow f_5$,
$r4$: $f_7 \rightarrow f_8$,
$r5$: $f_5 \rightarrow f_3$,
$r6$: $f_3 \rightarrow \neg f_2$.

Let us assume that set C contains facts: $C = \{f_1, f_4, f_6\}$.

One can easy recognize that facts f_3 and f_2 are incompatible: they cannot be simultaneously true. For example, f_2 means that GPS positioner indicates that the device is in the safe place (office); f_3 means that temperature is very low (too low for the interior of the office). The steps of the forward chaining inference mechanism will be as follows:

(1) In the first step, on the basis of rule $r1$, the mechanism concludes f_2 and adds f_2 to set C', and the consistency guard checks if C' is consistent. Since neither C nor C' contain conflicting facts ($(C \cup C') \nvdash \neg f_2$), the inference engine moves to the next step.

(2) On the basis of rule $r2$, the mechanism infers $\neg f_3$ and adds $\neg f_3$ to set C'. The consistency guard checks if the new conclusion is consistent with existing knowledge. Since $(C \cup C') \nvdash f_3$, the inference engine moves to the next step.

(3) On the basis of rule $r3$, the mechanism infers f_5 and adds f_5 to set C'. The consistency guard checks if the new conclusion is consistent with existing knowledge. Since $(C \cup C') \nvdash \neg f_5$, the inference engine moves to the next step.

(4) Since the condition of rule r_4 is not satisfied ($(C \cup C') \nvdash f_7$), the engine does not add f_8 to set C' but moves to the next step.

(5) On the basis of rule $r5$, the mechanism infers f_3 and adds f_3 to set C'. The consistency guard checks if the new conclusion is consistent with existing knowledge. Since $(C \cup C') \vdash f_3, \neg f_3$, then the consistency guard interrupts the work of the inference engine and begins procedure ALARM.

Concluding the above, the consistency guard will be a part of a fact-based inference mechanism and it will start the ALARM procedure in the case of the incompatibility of facts on any step in the process of inference.

3.5. Activity. The system behavior can be described by activities which are performed by entities, either explicitly by the user while interacting with the host or implicitly by the host while running various applications.

Definition 7 (activity). A is a set of activities:

$$A = (a_1, a_2, \ldots, a_n), \tag{2}$$

where $a_1, \ldots, a_n$ are the activities describing system behavior.

As an activity one can enumerate sending/checking e-mail, getting data from the database, setting a VPN connection, and so forth.

3.5.1. Requirement Rules. The level of security protection for a given activity is regulated by requirement rules.

Definition 8 (requirement rule). Requirement rules are formulated as follows:

$$\text{Conditions} \Rightarrow \text{Req}_a,$$

where

(i) Conditions is a list of rule conditions in the form $a \wedge (wl_a \ \textbf{func} \ wl_b \ \textbf{func} \cdots wl_d)$, where a is activity, **func** are the operators from the set = $\{\vee, \wedge\}$, and $\{wl_a, wl_b, \ldots, wl_d\}$ are literals (nonnegated or negated by a negation as failure);

(ii) Req_a are security requirements of activity a: $\text{Req}_a = \{\text{req}_1, \text{req}_2, \ldots, \text{req}_n\}$, where $\text{req}_l = (\text{sa}_l, \text{relation}, \text{value}_l)$, sa_l is a security attribute, relation is a relation from the set $\{<, >\}$, and value_l is a threshold value.

We denote a set of all requirement rules as R. By RR we denote a set of requirement rules (RR $\subset R$) with satisfied conditions.

The requirement rule establishes the desired level of protection of a device (requirements Req_a) allowing for the performance action a in an external environment (context) described by the conditional part of the rule (wl_a **func** wl_b **func** $\cdots wl_d$).

Definition 9 (satisfaction of a requirement). Predicate $\text{Sat}(S, \text{req}_k)$ denotes that the evaluations of security attributes in set S meet the requirement req_k.

If sa_m is a security attribute evaluated in S, req_k is a requirement (sa_m, relation, value_k), and (Val(S, sa_m) relation, value_k) is satisfied, then $\text{Sat}(S, \text{req}_k)$ is true (where value_k is a threshold value from req_k and relation is a relation from req_k).

Definition 10 (satisfaction of action requirements). If for security evaluation S every req $\in$ Req_a is satisfied ($\forall_{\text{req} \in \text{Req}_a} \text{Sat}(S, \text{req})$), then an action a can be performed.

Quality of protection evaluation and context-based requirement rules are the grounds for a decision whether an action can be allowed or banned. If security requirements

```
(1) SET C
(2) SET C' ← RES(C, RF)
(3) P ← C ∪ C'
(4) SET INTENTION(a)
(5) S ← EVAL
(6) SET R
(7) CONTEXT(P, a, S, R)
(8) if VSE = EMPTY then
(9)    STAT(a) ← ALLOW
(10)   EXE(a)
(11) end if
(12) if VSE ≠ EMPTY then
(13)   do
(14)   foreach {req[i] in VSE} do {
(15)   if NOTPOSSIBLE[i] then
(16)      STAT(a) ← DENY
(17)      EXIT
(18)   end if
(19)   INCREASE[i]
(20)   }
(21)   S ← EVAL
(22)   CONTEXT(P, a, S, R)
(23)   if VSE = ∅ then
(24)      STAT(a) ← ALLOW
(25)   end if
(26)   while (STAT(a) ≠ ALLOW)
(27) end if
(28) if STAT(a) = ALLOW then
(29)   EXE(a)
(30) end if
```

Algorithm 1: Security attributes adaptation algorithm.

are not met, the system should inform the user which of the facts describing context violate the requirement (such as GPS position) and which can help the user to change dangerous environment (e.g., to leave an insecure place). The procedure of evaluation of a possibility of performing given action a is presented in Algorithm 1.

3.6. Conflicts between Rules. Requirement rules for a given activity establish a minimal level of quality of protection of the system, by means of security attributes' evaluation thresholds. Since each of the requirement rules describes complete requirements of a given action, only one of them can be used to establish requirements. In specific conditions, a conflict between such rules can appear.

Definition 11 (conflicting rules). There is a conflict between two or more requirement rules if these rules cannot be executed together.

Such conflicts appear when there are two rules with satisfied antecedents, which establish requirements for the same action: If two rules rr_1 and rr_2 are in the set RR ($\mathrm{rr}_1, \mathrm{rr}_2 \in$ RR) and they establish requirements for the same action a, then these rules are in conflict.

The problem of conflicting and subsuming rules is the main reason for utilization of defeasible implication. In this work as defeasibility of the evaluation rules we understand the possibility of exclusion from the reasoning process of a chosen rule by another rule. If antecedents of two conflicting rules are satisfied, only one of them can be executed (but such a rule can also be defeated by another one).

To represent priorities between evaluation rules, we assume partial order OR between rules from a set R. Such order allows us to express that if $\mathrm{rr}_1 > \mathrm{rr}_2$ and $\mathrm{rr}_1, \mathrm{rr}_2 \in$ RR, then rules rr_1 and rr_2 are in conflict and when the conditions of both of these rules are satisfied, rule rr_1 should defeat rule rr_2. Our model of conflict resolution mechanism is built on the basis of theoretical models discussed in the papers devoted to formal modelling of legal reasoning and argumentation, for example, in [22, 25–27].

How does the conflict resolution mechanism work? For example, if set P contains facts $P = \{f_1, f_2, f_3\}$, set RR contains two rules with satisfied conditions ($\mathrm{rr}_1 : a \wedge (f_1 \wedge f_2) \Rightarrow \mathrm{Req}_{a1}$ and $\mathrm{rr}_2 : a \wedge (f_1 \wedge f_2 \wedge f_3) \Rightarrow \mathrm{Req}_{a2}$), the user is going to perform action a, and $\mathrm{rr}_1 > \mathrm{rr}_2 \in$ OR, then rule rr_1 defeats rule rr_2 (the mechanism excludes rule rr_2 from the reasoning process).

The issue of ordering of conflicting rules certainly requires further discussion. We realize that there may be a number of hardly predictable sources of conflicting rules' orderings, which makes a fully automated mechanism of ordering generation very difficult (or even impossible) to construct. In our model we assumed that ordering is declared in advance by the constructor of a system. We do not give any restrictions to the constructors of the ordering, assuming their rationality and high-quality expert knowledge.

However, there is one kind of conflict which allows us to detect and recognize a special kind of orders between conflicting rules, which will be discussed in future work. In some cases two conflicting rules may have subsuming conditions, for example, if $\mathrm{rr}_1 : a \wedge (f_1 \wedge f_2) \Rightarrow \mathrm{Req}_{a1}$ and $\mathrm{rr}_2 : a \wedge (f_1 \wedge f_2 \wedge f_3) \Rightarrow \mathrm{Req}_{a2}$, then every case which satisfies the conditions of rule rr_2 also satisfies the conditions of rule rr_1. Usually, in such a situation, a more specific rule is stronger than a general one, because it regulates a specific case of a standard situation regulated by a more general rule. This mechanism comes from the theory of law and is called *lex specialis derogat legi generali* (specific act (provision) derogates from (prevails over) the general regulation). Nevertheless, the implementation of the mechanism in such a complicated matter requires further elaboration, which will be performed in future work.

3.7. Process of Establishing of Minimal Requirements of Security Attribute Protection Level. We assume that the estimation of the quality of protection of security attributes of a given device is based on the system described in [24]. Relying on the analyzed device's parameters, the system makes an evaluation and returns the set of security attributes with their estimation. Since estimation of each security attribute is a positive integer number, the result obtained from the system can be easily transformed into the set S.

The main aim of the current work is to decide if quality of protection of the analyzed device (established on the basis of

the system described in [24]) is enough to meet the requirements. Since we have an estimation of a quality of protection of our device, we need to estimate security requirements for a given action and resource in a given context. The process of establishing the minimal requirements of security attribute protection level is based on requirement rules. For every security attribute we assume starting level of quality of protection and satisfaction of every requirement rule's condition to cause an adequate change of requirement level. The final level of quality of protection of a chosen security attribute is a minimal requirement of level of protection of this security attribute.

If the requirements necessary to perform a given action are not met, then the system returns conditions (security attributes) which are violated. If the system receives security attributes which do not meet requirements, the process of adaptation of the system begins.

3.8. Adaptation Process. The process of adaptation of quality of protection of a given device is based on the assumption that every security attribute has assigned a set of quality of protection parameters and external context factors (facts) that influence its QoP estimation, as well as a set of possibilities of increasing them. Such parameters or facts can be changed in order to increase evaluation of quality of protection (e.g., increase the key length) or to decrease context requirements (e.g., to move to a safer place).

4. Algorithms

The implementation of context-aware security system for industrial applications can be divided into two major subalgorithms. Algorithm 1 is the general algorithm responsible for adaptation process. Algorithm 2 represents a specific algorithm responsible for estimation if the quality of protection of the device meets requirements caused by context in which the device is working.

The notation used in the algorithms is presented below:

(i) SET is indication of making a choice.

(ii) EXCLUDE is a procedure excluding requirement rule rr[k] from the set RR.

(iii) READ is a reading indication.

(iv) CONTINUE means processing statement will be skipped.

(v) RES(C, RF) is the reasoning function based on a set of facts C and rules RF (the reasoning function works on the basis of inference mechanism described earlier). The reasoning function contains consistency guard, the procedure controlling consistency of sensors which, in the case of inconsistency, interrupts the inference process and starts procedure ALARM.

(vi) ALARM is the procedure of alarm; if consistency guard returns that there are inconsistencies in sensors' indications (false) the system stops the context analysis, stops actions, and waits for user reaction.

(vii) C is a case expressed by a set of facts.

```
(1) RR = ∅
(2) SET OR
(3) ADD(RR, SATISFIED(P, R, a))
(4)
(5) for k = 1 to COUNT(RR) do
(6)    for m = 1 to COUNT(RR) do
(7)       if  (rr[k] > rr[m]) ∈ OR then
(8)          EXCLUDE rr[m] from RR
(9)       end if
(10)   end for
(11) end for
(12)
(13) if COUNT(RR) > 1 then
(14)    ERROR
(15) end if
(16) SET RULE ← RR
(17) SET Req_a ← CONC(RULE)
(18) for i = 1 to COUNT(Req_a) do
(19)    if SAT(S, Req_a, req[i]) = false then
(20)       VSE ← req[i]
(21)    end if
(22) end for
(23) RETURN VSE
```

Algorithm 2: Algorithm of the security attributes context evaluation: CONTEXT(P, a, S, R).

(viii) C' is a set of facts obtained from the inference mechanism.

(ix) i is the index of the current security attribute.

(x) P is full description of a case.

(xi) RF is a set of rules.

(xii) a is activity a.

(xiii) INTENTION(a) indicates that the device is going to perform action a.

(xiv) STAT(a) is the status of an action a which can have two values: allow or deny.

(xv) k, m, lindicates a current requirement rule.

(xvi) t indicates a current requirement.

(xvii) R is a set of all requirement rules.

(xviii) SATISFIED(P, R, a) is a function which returns set of requirement rules with satisfied conditions made on the basis of description of a case P, intention a, and set of requirement rules R.

(xix) RR is a set of requirement rules with satisfied conditions.

(xx) ADD(RR, SATISFIED(P, R, a)) is a function which adds results of SATISFIED(P, R) to the set RR.

(xxi) rr[m] is mth requirement rule from the set RR.

(xxii) OR is the order between rules from a set RR.

(xxiii) Req_a is conclusion of a given requirement rule which is a set of requirements concerning activity a.

(xxiv) req[t] is tth requirement in a set Req_a.

(xxv) COUNT(RR) is a function which returns a number of requirement rules in the set RR.

(xxvi) COUNT(Req_a) is a function which returns a number of requirements in Req_a.

(xxvii) n is the quantity of security attributes.

(xxviii) rr[x] is requirement rule x.

(xxix) S is the evaluation of security attributes.

(xxx) EVAL is a procedure which returns evaluation of security attributes of the device.

(xxxi) SAT(S, Req_a, req[t]) is a function which returns true if evaluation of security attribute in set S meets requirements req[t] from the set Req_a.

(xxxii) RULE is a rule which remains after exclusion of the conflicting ones from the set RR[i].

(xxxiii) CONC(RULE) is a function which returns conclusion of a rule RULE.

(xxxiv) EXE(a) is the execution of action a.

(xxxv) VSE is a list of violated security attributes.

(xxxvi) NOTPOSSIBLE[i] is a function which checks if it is not possible to increase the level of security attribute i protection (e.g., if there are no possibilities to increase key length).

(xxxvii) INCREASE[i] means increasing the level of protection of security attribute i.

(xxxviii) **foreach** {i in VSE **do** {...} is "foreach" loop: for each security attribute i in the list VSE do {...}.

(xxxix) **do** ... **while**(...) is "do-while" loop.

4.1. Algorithm 1: Adaptation Process. The general algorithm responsible for adaptation process is the main one and can be divided into eight main steps:

Step 1. In the first step, the system infers the general description of the case (on the basis of the raw facts obtained from the device sensors, rules, and the fact-based inference mechanism). During the inference, the context guard checks if there are inconsistent indications from the sensors. If not, the main process of adaptation of security attributes protection level begins. In the case of inconsistency between facts obtained from sensors, the ALARM procedure begins (Steps (1)–(4) in Algorithm 1).

Step 2. In the second step, an estimation of quality of protection of a device is performed (Step (5) in Algorithm 1, mechanism of QoP evaluation is described in a detailed way in [24]).

Step 3. In the next step, the algorithm of the security attributes, context evaluation tests if the security attributes' protection meets the requirements based on the context in which the device is working (Steps (6)-(7) in Algorithm 1).

Step 4. If requirements are met, the intended action is allowed. If not, the system proceeds to another step (Steps (8)–(11) in Algorithm 1).

Step 5. In the next step, for each security attribute with insufficient protection (failing to meet context requirements) the system tries to increase the level of protection of each violated security attribute (Steps (12)–(14) and (19) in Algorithm 1).

Step 6. If there are no possibilities of increasing the level of protection of any of the violated security attributes, the system denies the execution of the intended activities (Steps (15)–(18) in Algorithm 1).

Step 7. After the process of increasing the level of protection, the system performs the estimation of quality of protection and tests whether the security attributes meet the requirements based on the context in which the device is working (Steps (20)-(21) in Algorithm 1).

Step 8. If the requirements are met, the intended action can be performed. If not, the system tries to increase the level of protection again (Steps (22)–(29) in Algorithm 1).

The flowchart of Algorithm 1 is presented in Figure 4.

4.2. Algorithm 2: Security Attributes Context Evaluation. The process of estimation if a device, working at a certain level of quality of protection mechanisms, in a given context (external and internal environment in which a device is currently working) fulfills security requirements of an action which a user intents to perform can be performed on the basis of described below algorithm. Generally speaking, the mechanism of the estimation can be divided into three main steps.

Step 1. In the first stage, the system adds requirement rules with satisfied conditions to the set RR (Steps (1)–(4) in Algorithm 2).

Step 2. In the second stage, the system detects which of the requirement rules with satisfied conditions devoted to an action which the user is going to perform are in conflict. Following that, the system defeats the conflicting rules, leaving the most suitable one (Steps (5)–(12) in Algorithm 2).

Step 3. In the next stage, the system checks if requirements derived from requirement rules are satisfied by a given state of a system (described by quality of protection evaluation from [24]). If yes, the algorithm returns an empty set of violated security attributes requirements. If not, the algorithm returns a set of violated security attributes requirements (Steps (13)–(23) in Algorithm 2).

The flowchart of Algorithm 2 is presented in Figure 5.

5. Case Study: Context-Aware Security for Mobile Devices

For the purpose of presenting and explaining our model, the case study will be illustrated by steps described in Section 4, where the algorithms are presented. In order to demonstrate our approach, we present two scenarios.

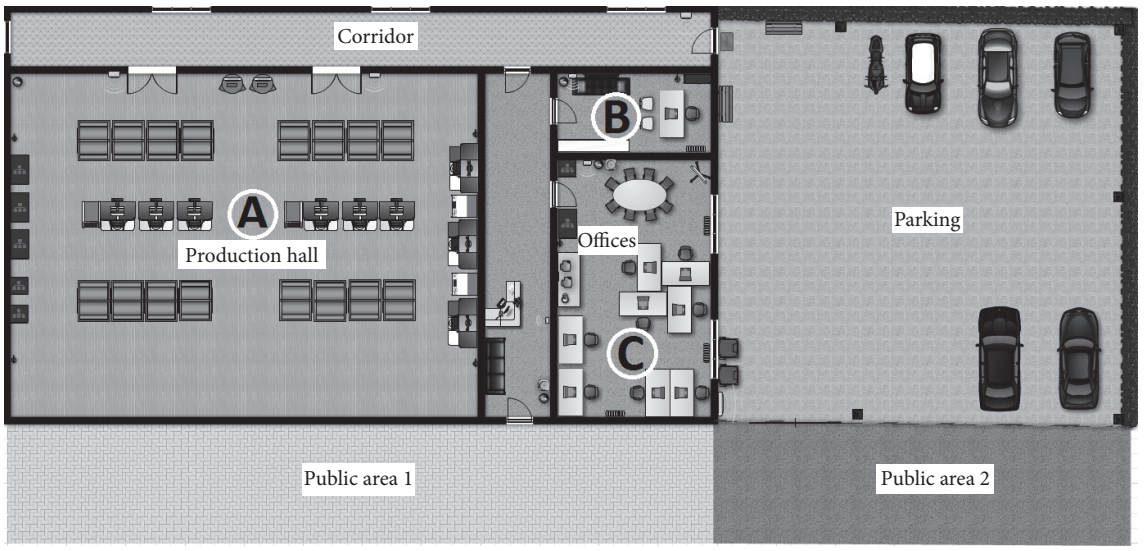

Figure 1: The floor plan of the organization's seat.

The case study incorporates the following actors: employee and manager. The events take place inside a working environment (an organization's headquarters) presented in Figure 1. The location consists of 3 rooms: manager's office, employees' room (working quarters), and guest room/corridor. Only the manager and employees are allowed in the manager and employee rooms. The guest room/corridor is accessible to anyone. Each room is equipped with a different access point for WiFi connections (signal strength is measured to predict location). It is forbidden to take photos in the manager's room and only managers are allowed to enter. The working hours of the organization are Monday–Friday, from 8:00 am to 5:00 pm. The IT services provided by the organization can only be accessed by the manager and employees during working hours and only when located indoors (this includes the database access). We assume that the manager and each of the employees and guests possess a smartphone with the following capabilities: sensing temperature (sensor), brightness level (camera, sensor), position (GPS, AP), directional movement (accelerometer, gyroscope), launching applications, network connection (WiFi), and time management (calendar, clock).

5.1. Case Study: Scenario 1. An employee uses his smartphone during working hours to access the database (TLS is used to secure the connection) [28]. He is connected to the production hall access point. The sequence of steps of the context-aware security analysis mechanism is presented below.

Algorithm 1

ALG1, Step 1. The following facts belong to the environment of the case:

f_1(user) = employee.

f_2(user) = guest.

f_1(time) = working hours.

f_2(time) = nonworking hours.

f_1(APL) = production hall access point (safer).

First, we define the facts of the case.

Case 1. $C_1 = \{f_1(\text{user}), f_1(\text{time}), f_1(\text{APL})\}$.

The fact-based rules set RF:

$f_1(\text{user}) \rightarrow \neg f_2(\text{user})$.

$f_2(\text{user}) \rightarrow \neg f_1(\text{user})$.

$f_1(\text{time}) \rightarrow \neg f_2(\text{time})$.

$f_2(\text{time}) \rightarrow \neg f_1(\text{time})$.

The inference engine infers the set C':

$$C' = \{\neg f_2(\text{user}), \neg f_2(\text{time})\};$$

hence

$$P = \{f_1(\text{user}), f_1(\text{time}), f_1(\text{APL}), \neg f_2(\text{user}), \neg f_2(\text{time})\}.$$

During every step of the inference process, the procedure of the facts consistency analysis (by a consistency guard) was performed. It turned out that there are no conflicts, so the alarm was not activated.

Then we define the intention as follows.

The intention of the device is to allow the employee to access and gather data from the database (INTENTION(a) = access and gather data from the database).

ALG1, Step 2. QoP estimation is performed by the mechanism described in [24]:

sa_1 = confidentiality.

sa_2 = integrity.

sa_3 = authorisation.

sa_4 = authentication.

sa_5 = availability.

sa_6 = anonymity.

$$S = (sa_1, 1), (sa_2, 2), (sa_3, 3), (sa_4, 4), (sa_5, 5), (sa_6, 6). \quad (3)$$

ALG1, Step 3. Next we evaluate if the security attributes protection meets the requirements based on context in which the device is working. Having declared 2 requirement rules

$$R = \{rr_1, rr_2\}$$
$$rr_1 = a \wedge f_1(\text{user}) \wedge f_1(\text{time}) \Rightarrow \text{Req}_a 1$$
$$rr_2 = a \wedge f_1(\text{time}) \Rightarrow \text{Req}_a 2,$$

where

$$\text{Req}_a 1 = \{req_1, req_2, req_3, req_4, req_5\},$$
$$\text{Req}_a 2 = \{req_6, req_7, req_8, req_9, req_{10}\},$$

and

$$req_1 = sa_1, >, 0$$
$$req_2 = sa_2, >, 1$$
$$req_3 = sa_3, >, 2$$
$$req_4 = sa_4, >, 3$$
$$req_5 = sa_5, >, 5$$
$$req_6 = sa_1, >, 2$$
$$req_7 = sa_2, >, 3$$
$$req_8 = sa_3, >, 4$$
$$req_9 = sa_4, >, 5$$
$$req_{10} = sa_5, >, 6$$

execution of Algorithm 2 begins.

Algorithm 2

ALG2, Step 1. Since both rules have satisfied conditions, we obtain the set $RR = \{rr_1, rr_2\}$.

Now we check if the QoP of the device meets requirements caused by context.

ALG2, Step 2. We assume order $OR = \{rr_1 > rr_2\}$.

And therefore rr_2 is excluded:

$$RR = \{rr_1\}.$$

The remaining rule is set to be rr_1.

Requirement rule $\text{Req}_a 1$ becomes the binding conclusion.

ALG2, Step 3. Now it is the time to check if requirements derived from requirement rules are satisfied.

For every req_x in $\text{Req}_a 1$

req_1 in $\text{Req}_a 1$ is satisfied since $1 > 0$ (true),

req_2 in $\text{Req}_a 1$ is satisfied since $2 > 1$ (true),

req_3 in $\text{Req}_a 1$ is satisfied since $3 > 2$ (true),

req_4 in $\text{Req}_a 1$ is satisfied since $4 > 3$ (true),

req_5 in $\text{Req}_a 1$ is not satisfied since $5 > 5$ (false);

req_5 is added to the list of violated security attributes VSE.

Return to Algorithm 1 with the list of violated security attributes VSE.

Algorithm 1

ALG1, Step 4. Since the requirements were not met, we cannot allow the action yet and proceed to the next step.

ALG1, Step 5. Do the following until the action a is finally allowed.

For each violated security attribute in VSE do the following.

We check whether it is possible to increase the level of protection of req_5's security attribute and find out that it is possible and so increase it iteratively to 6.

Next we evaluate again the security attributes increasing the security level of req_5's security attribute (which is sa_5):

sa_1 = confidentiality.

sa_2 = integrity.

sa_3 = authorisation.

sa_4 = authentication.

sa_5 = availability.

sa_6 = anonymity.

$$S = \{(sa_1, 1), (sa_2, 2), (sa_3, 3), (sa_4, 4), (sa_5, 5). (sa_6, 6)\}. \quad (4)$$

ALG1, Step 6. We skip this step since it is possible to increase the security level of req_5.

ALG1, Step 7. We check again for not meeting requirements regarding security attributes by executing Algorithm 2. Since all the requirements are met, we can move to step (8).

ALG1, Step 8. Since we did not find any violated security attributes, we allow the action a.

The action is allowed; therefore, the user is finally able to establish connection with the database and gather the data that he needs.

5.2. Case Study: Scenario 2. An employee tries to take a picture in the production hall. The mobile phone's clock points at 10:30 am and the device rapidly switches between two APs (production hall and manager's room, signal strength also switches) indicating whether the constant movement between the rooms is malfunction or a possible attack on the device.

Algorithm 1

ALG1, Step 1. The following facts belong to the environment of the case:

$f_1(\text{user}) = \text{employee}.$

$f_2(\text{user}) = \text{guest}.$

$f_1(\text{time}) = \text{working hours}.$

$f_2(\text{time}) = \text{nonworking hours}.$

$f_1(\text{APL}) = \text{production hall access point}.$

$f_3(\text{APL}) = \text{manager's office access point}.$

First, we define the facts of the case.

Case 2. $C_2 = f_1(\text{user}), f_1(\text{time}), f_1(\text{APL}), f_3(\text{APL})$.

The fact-based rules set RF:

$f_1(\text{user}) \rightarrow \neg f_2(\text{user}).$

$f_2(\text{user}) \rightarrow \neg f_1(\text{user}).$

$f_1(\text{time}) \rightarrow \neg f_2(\text{time}).$

$f_2(\text{time}) \rightarrow \neg f_1(\text{time}).$

$f_1(\text{APL}) \rightarrow \neg f_3(\text{APL}).$

The inference engine infers the set C'. At each step of the forward chaining mechanism, the consistency guard checks if sets C and C' are consistent.

First of all, the mechanism infers that $C' = \{\neg f_2(\text{user})\}$. Since there are no inconsistencies in sets C and C', the engine continues with its work.

Now the set C' consists of two facts: $C' = \{\neg f_2(\text{user}), \neg f_2(\text{time})\}$. Both sets are consistent.

Finally, a new fact $\neg f_3(\text{APL})$ is added to the set C':

$$C' = \{\neg f_2(\text{user}), \neg f_2(\text{time}), \neg f_3(\text{APL})\}; \quad (5)$$

hence, a conflict appears in the consistency of $f_3(\text{APL})$ (it is simultaneously true and false). The ALARM procedure is invoked and therefore all subsequent steps are omitted. The employee gets notification about the inconsistency and the system lets him decide about the action a (taking the picture).

5.3. Implementation. As part of the project, an actual implementation of the mechanisms described in this paper was developed for the purpose of providing a proof of concept. To make it accessible for a wide variety of interested users, an application for the most popular mobile operating system (Android) was created. The version on which it is targeted is 6.0; however, due to Android's backward compatibility it should be functional on most previous versions of the operating system. The application's source can be found on [29].

The application strictly follows the algorithms and rules of the presented model in order to reason and decide on whether to allow or block certain actions of the user. For simplicity we have added the functionality of preloading the settings of our described theoretical cases. The application's architecture is depicted in Figure 2.

The experiments that were performed involved a facility equipped with 6 APs, each in range and forming different networks. For both of the intentions the following experimental scenarios were conducted:

(i) The user intends to open the browser on his device in order to view a website. The intention involved 5 experiments which differed by the user's location and distance to the associated APs. When the user was in range and connected to the trusted network he was also able to open the website; else he was blocked from doing so. Three out of five times he was in range and, at two out of them, the application evaluated and decided successfully in all cases.

(ii) The user intends to take a picture with his device. This intention involved 5 additional experiments which also differed by the same user's location and distance to the associated APs as the previous intention. Whenever the threshold of −60 dBm or higher was reached and the signal was strengthening the application blocked the users intention of taking a picture since he was in the vicinity of a photo restricted area. Four experiments were conducted with varying distance to the specific AP that indicated the photo-free area, two of them with better signal strength than that of the threshold and two with lower. The fifth variation of the experiment was conducted with an additional fact indicating that it was simultaneously true and false (another AP in range within the threshold) and this time it raised an inconsistency alarm as seen in Figure 3. In all five cases, the final evaluation and resulting action of the application were correct.

The possible outcomes of such intentions were to get correct permissions to the actions and raise the alarm because of the inconsistencies found between facts. In the case of disallowance of certain actions, the application displays the reason and blocks the action. The performed tests and evaluations validate proposed mechanisms for context-aware security management for mobile devices.

6. Conclusions

In the article we propose the model of context-aware security, adjusting system which is built on the basis of propositional logic, with the aim of providing an advanced and effective method for balancing the level of security of a system. QoP parameters and external context factors influence the security level estimation in the model. Thanks to the proposed approach, the decision support system can infer whether the actions that the user is going to perform in a particular external environment meet security requirements. The main contributions of the proposed approach are summarized as follows:

(i) consistency guard: a specialized module for the control of the consistency of sensory/contextual data which recognizes and counteracts certain dangerous states;

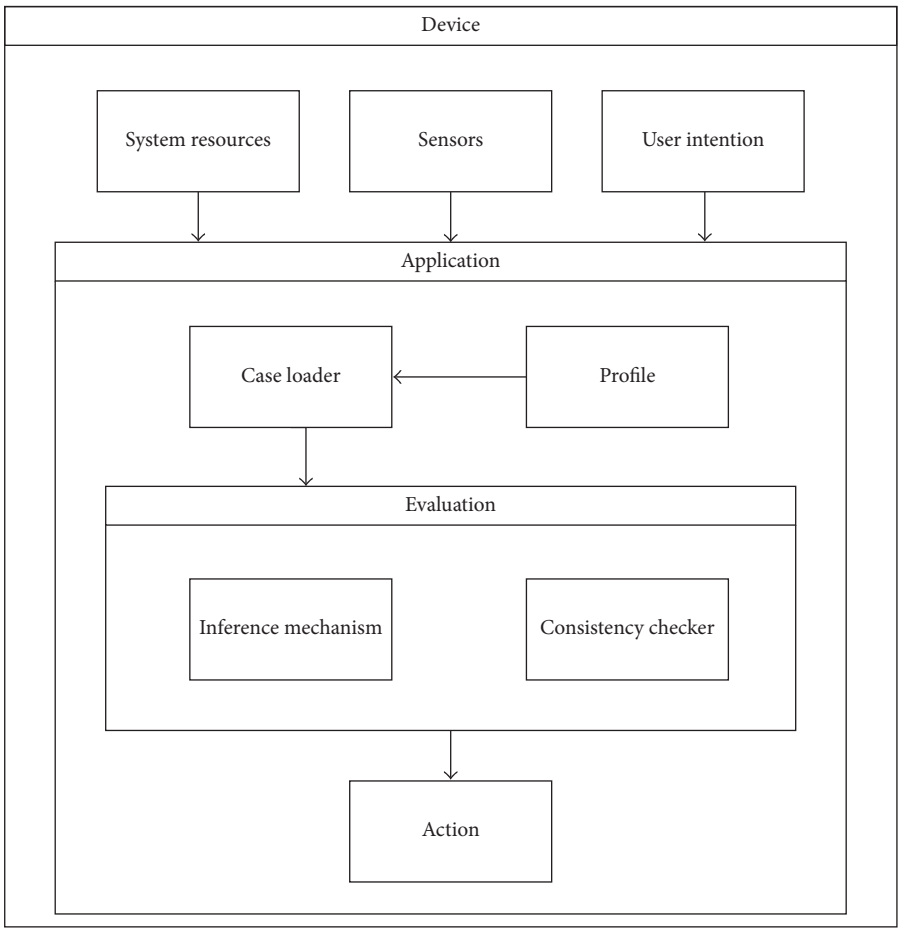

Figure 2: Diagram depicting the application's architecture.

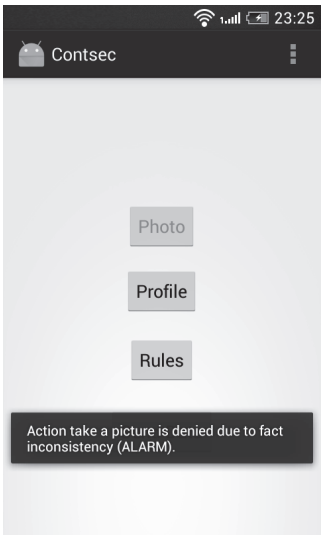

Figure 3: Screenshot of the application.

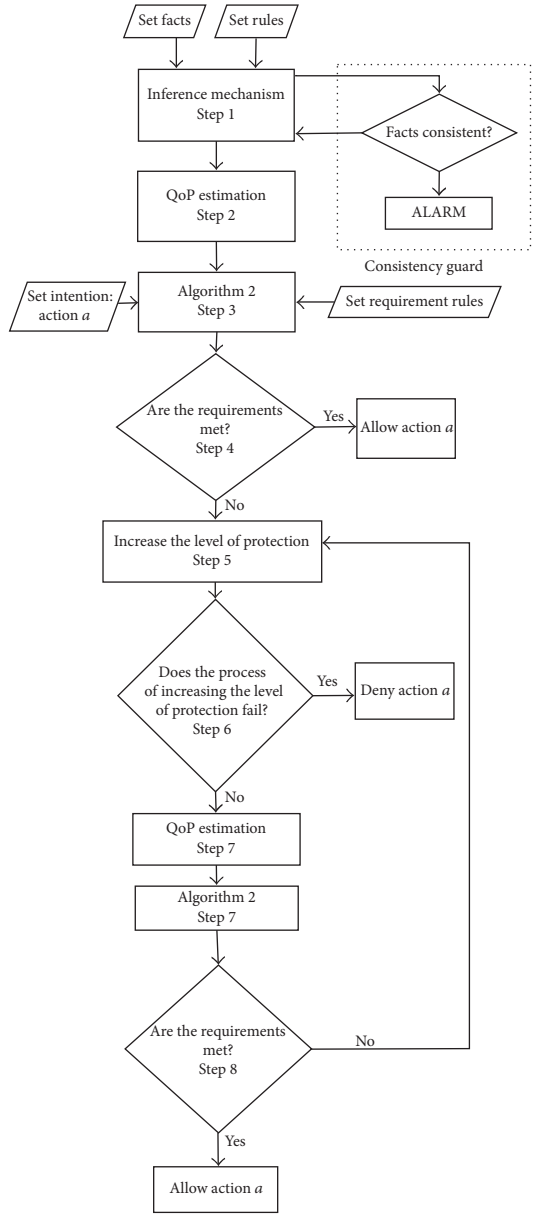

Figure 4: Diagram depicting the subsequent steps of Algorithm 1.

(ii) conflicting rules exclusion: a mechanism that finds, compares, and eradicates clashing rules through use of the defeasible implication;

(iii) formal representation: describing and defining in detail all of the elements and mechanisms of the model;

(iv) holistic: taking into account all of the possible contextual data instead of just a predefined small set of factors.

In order to better illustrate the process of reasoning and to provide an overall working example, we prepared and described case study scenarios, which show various elements of the model in action under specific circumstances. We expect that our model will enhance the process of security evaluation and bring about promising results.

Appendix

See Figures 4 and 5.

Competing Interests

The authors declare that they have no competing interests.

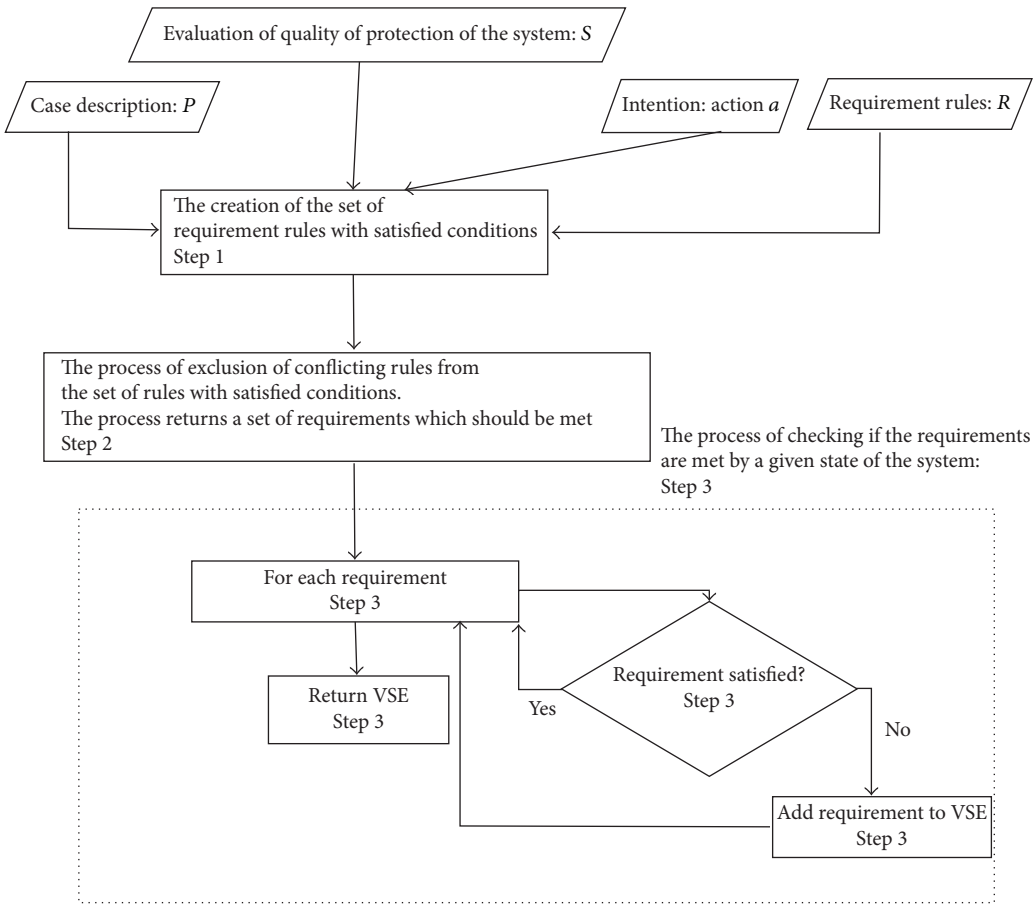

Figure 5: Diagram depicting the subsequent steps of Algorithm 2.

Acknowledgments

This work is supported by the Polish National Science Centre Grant 2012/05/B/ST6/03364.

References

[1] B. Ksiezopolski, Z. Kotulski, and P. Szalachowski, "Adaptive approach to network security," in *Computer Networks*, vol. 39 of *Communications in Computer and Information Science*, pp. 233–241, Springer, Berlin, Germany, 2009.

[2] B. Ksiezopolski, Z. Kotulski, and P. Szalachowski, "On QoP method for ensuring availability of the goal of cryptographic protocols in the real-time systems," in *Proceedings of the European Teletraffic Seminar*, pp. 195–202, 2011.

[3] B. Ksiezopolski and Z. Kotulski, "Adaptable security mechanism for dynamic environments," *Computers & Security*, vol. 26, no. 3, pp. 246–255, 2007.

[4] B. Ksiezopolski, D. Rusinek, and A. Wierzbicki, "On the modeling of Kerberos protocol in the Quality of Protection Modelling Language (QoP-ML)," *Annales UMCS Informatica AI XII*, vol. 4, pp. 69–81, 2012.

[5] B. Ksiezopolski, D. Rusinek, and A. Wierzbicki, "On the efficiency modelling of cryptographic protocols by means of the Quality of Protection Modelling Language (QoP-ML)," in *Information and Communication Technology*, vol. 7804 of *Lecture Notes in Computer Science*, pp. 261–270, Springer, Berlin, Germany, 2013.

[6] C. Ntanos, C. Botsikas, G. Rovis, P. Kakavas, and D. Askounis, "A context awareness framework for cross-platform distributed applications," *The Journal of Systems and Software*, vol. 88, no. 1, pp. 138–146, 2014.

[7] Y. Mowafi, D. Abou-Tair, T. Aqarbeh, M. Abilov, V. Dmitriyev, and J. M. Gomez, "A context-aware adaptive security framework for mobile applications," in *Proceedings of the 3rd International Conference on Context-Aware Systems and Applications (ICCASA '14)*, pp. 147–153, 2014.

[8] P. Makris, D. N. Skoutas, and C. Skianis, "A survey on context-aware mobile and wireless networking: on networking and computing environments' integration," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 362–386, 2012.

[9] E. Hayashi, S. Das, S. Amini, J. Hong, and I. Oakley, "CASA: context-aware scalable authentication," in *Proceedings of the Symposium on Usable Privacy and Security (SOUPS '13)*, ACM, Menlo Park, Calif, USA, 2013.

[10] W. Li, A. Joshi, and T. Finin, "CAST: context-aware security and trust framework for mobile ad-hoc networks using policies," *Distributed and Parallel Databases*, vol. 31, no. 2, pp. 353–376, 2013.

[11] A. Forkan, I. Khalil, and Z. Tari, "CoCaMAAL: a cloud-oriented context-aware middleware in ambient assisted living," *Future Generation Computer Systems*, vol. 35, pp. 114–127, 2014.

[12] M. Miettinen, S. Heuser, W. Kronz, A. Sadeghi, and N. Asokan, "ConXsense - Automated Context Classification for Context-Aware Access Control," *Computer and Communications Security (ASIACCS*, pp. 293–304, 2014.

[13] Ö. Yılmaz and R. C. Erdur, "IConAwa—an intelligent context-aware system," *Expert Systems with Applications*, vol. 39, no. 3, pp. 2907–2918, 2012.

[14] K. Stefanidis, E. Pitoura, and P. Vassiliadis, "Managing contextual preferences," *Information Systems*, vol. 36, no. 8, pp. 1158–1180, 2011.

[15] S. Schefer-Wenzl and M. Strembeck, "Modeling context-aware RBAC models for business processes in ubiquitous computing environments," in *Proceedings of the 3rd FTRA International Conference on Mobile, Ubiquitous, and Intelligent Computing (MUSIC '12)*, pp. 126–131, Vancouver, Canada, June 2012.

[16] R. Ali, F. Dalpiaz, and P. Giorgini, "Reasoning with contextual requirements: detecting inconsistency and conflicts," *Information and Software Technology*, vol. 55, no. 1, pp. 35–57, 2013.

[17] M. Younas and S. Mostefaoui, "A new model for context-aware transactions in mobile services," *Personal and Ubiquitous Computing*, vol. 15, no. 8, pp. 821–831, 2011.

[18] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: a survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[19] G. J. Nalepa and S. Bobek, "Rule-based solution for context-aware reasoning on mobile devices," *Computer Science and Information Systems*, vol. 11, no. 1, pp. 171–193, 2014.

[20] L. Sliman, F. Biennier, and Y. Badr, "A security policy framework for context-aware and user preferences in e-services," *Journal of Systems Architecture*, vol. 55, no. 4, pp. 275–288, 2009.

[21] A. Merlo, M. Migliardi, and L. Caviglione, "A survey on energy-aware security mechanisms," *Pervasive and Mobile Computing*, vol. 24, pp. 77–90, 2015.

[22] H. Prakken and G. Vreeswijk, "Logics for defeasible argumentation," in *Handbook of Philosophical Logic*, D. Gabbay, Ed., Kluwer Academic Publisher, 2000.

[23] ISO/IEC, "Information technology—security techniques—information security management systems—requirements," ISO/IEC 27001:2005, 2005.

[24] B. Ksiezopolski, T. Zurek, and M. Mokkas, "Quality of protection evaluation of security mechanisms," *The Scientific World Journal*, vol. 2014, Article ID 725279, 18 pages, 2014.

[25] H. Prakken and G. Sartor, "A dialectical model of assessing conflicting arguments in legal reasoning," *Artificial Intelligence and Law*, vol. 4, no. 3-4, pp. 331–368, 1996.

[26] T. Zurek, "Model of argument from social importance," in *Legal Knowledge and Information Systems: JURIX 2014*, R. Hoekstra, Ed., vol. 271 of *Frontiers in Artificial Intelligence and Applications*, pp. 23–28, IOS Press, 2014.

[27] S. Modgil and H. Prakken, "The ASPIC$^+$ framework for structured argumentation: a tutorial," *Argument and Computation*, vol. 5, no. 1, pp. 31–62, 2014.

[28] IETF, "The transport layer security (TLS) protocol v.1.2," RFC 5246, IETF, 2008.

[29] M. Mokkas, "The source code of the context-aware application for mobile devices," 2016, https://github.com/MikeMokkas/Contsec.