Capturing Dynamicity and Uncertainty in Security and Trust via Situational Patterns

Tomas Bures[1], Petr Hnetynka[1], Robert Heinrich[2], Stephan Seifermann[2], Maximilian Walter[2],

[1] Charles University, Czech Republic
{bures,hnetynka}@d3s.mff.cuni.cz
[2] Karlsruhe Institute of Technology (KIT), Germany
{robert.heinrich,stephan.seifermann,maximilian.walter}@kit.edu

Abstract. Modern smart systems are highly dynamic and allow for dynamic and ad-hoc collaboration not only among devices, but also among humans and organizations. Such a collaboration can introduce uncertainty to a system, as behavior of humans cannot be directly controlled and the system has to deal with unforeseen changes. Security and trust play a crucial role in these systems, especially in domains like Industry 4.0 and similar. In this paper we aim at providing situational patterns for tackling uncertainty in trust – in particular in access control. To do so, we provide a classification of uncertainty of access control in Industry 4.0 systems and illustrate this on a series of representative examples. Based on this classification and examples, we derive situational patterns per type of uncertainty. These situational patterns will serve as adaptation strategies in cases when, due to uncertainty, an unanticipated situation is encountered in the system. We base the approach on our previous work of autonomic component ensembles and security ensembles.

Keywords: Dynamic systems · security · access control · uncertainty

1 Introduction

Smart systems (such as smart manufacturing in Industry 4.0, smart traffic, smart buildings, etc.) are becoming more and more ubiquitous. With this advent and their direct influence on human lives, also the problem of their security and trust in them is becoming highly relevant.

As the smart systems strive towards being more intelligent and being able to cope with various situations, they are becoming highly dynamic and rely on dynamic and ad-hoc collaboration not only among devices constituting a single system, but also among systems, humans and organizations. Such a collaboration typically introduces uncertainty in the system, due to faults in system components, unexpected behavior of humans, and not fully understood behavior of other systems and the environment. This all means that a smart system increasingly needs to deal with unforeseen situations and changes.

This uncertainty has a significant impact on the security and overall trust. While the security and trust are normally modeled in rather a strict (and often static) manner, the introduction of uncertainty demands loosening the strict boundaries of security and requires a system to inventively self-adapt to meet new and not fully foreseen situations.

In this paper, we take a first step towards such self-adaptation of security to not fully foreseen situations. We scope our work to access control in Industry 4.0 settings (as here we can derive experience from our completed project Trust4.0 and our ongoing project FluidTrust). To create a frame for such self-adaptation, we provide a classification of uncertainty of access control in Industry 4.0 systems and illustrate this on a series of representative examples. Based on this classification and examples, we derive situational patterns per type of uncertainty. These situational patterns will serve as adaptation strategies in cases when, due to uncertainty, an unanticipated situation is encountered in the system. We base the approach on our previous work of autonomic ensembles and security ensembles [4].

The structure of the paper is as follows. Section 2 analyzes state of the art and related work and then presents a classification of uncertainty. In Section 3, we discuss and analyze the representative examples of uncertainty and then, based on the classification and analysis of the examples, Section 4 defines the adaptation patterns. Section 5 overviews the adaptation framework where the patterns are employed and Section 6 concludes the paper.

2 Classification of uncertainty in security and trust

In this section, we first discuss existing related approaches and then, based on them, we build a classification of uncertainty regarding access control. The classification will serve as a foundation of run-time analyses.

2.1 State of the art in access control and uncertainty

As confirmed in [33], security is a critical attribute in dynamic and adaptive systems (among whose Industry 4.0 systems belong). The need for dynamicity and self-adaption in these systems stems from the constantly changing context in which the system operates. The survey in [29] discusses context-based middlewares targeting systems like Internet of Things (which are a special type of dynamic and adaptive systems). Security and privacy is dealt only by three middlewares out of eleven. As access control is one of most important aspects of security and confidentiality, RBAC and similar approaches are discussed below.

Access Control is one of established means to enable security and trust. The classical access control systems are DAC [40] and MAC [17] but they are applicable to the simplest solutions only. More advance is Role-based access control (RBAC) [1], which employs groups to gather access rights for similar users. Through this abstraction, the rules are more comprehensible. However, the strict static relationship from groups to rules does not fit dynamic situations and there is no horizontal composition supported between multiple organization.

Thus, the Organisational Based Access Control (OrBAC) has been introduced and recently enhanced by support of horizontal composition [9]. However, it does not support the inclusion of confidentiality analysis and uncertainty. Another well-known access control system is Attribute Based Access Control (ABAC) [25], where access is managed over attributes, which need to be satisfied for accessing data. In [7], an approach based on ABAC is described, which targets also dynamic situations, nevertheless only unexpected and uncommon user behavior (that might represent an attack) is considered. In [39], an approach targeting access policies generation for dynamically established coalitions is described, nevertheless, the coalitions are meant only as groups of people with the same goal but by themselves are not dynamically described. In [42], an approach for security and access control in health care IoT systems is described, but from the point of dealing with uncertainty, it supports only emergency like situations, for which it offers a "break-glass key" approach, i.e., there is a predefined set of persons that know the particular "break-glass key" and in the case of unexpected emergency situation, they have to be contacted. In [36], an adaptive access control approach based on answer set programming targeting context-based systems is shown, but uses predefined access control policies with predefined exceptions.

In summary, there are approaches targeting dynamic access control but only for anticipated changes (i.e., no uncertainty) and with rigid and predefined access rules. However, the increase in dynamicity leads to not anticipated changes, which results in uncertainty. Thus, it is important to take a look to at the uncertainty research area and try to combine it with access control approaches.

An important step to quantify the uncertainty is to realize its source. Depending on the classifications presented on [31], the uncertainty exists on different levels of the system, which are in modeling phase, adaptation functions, goal, environmental and resource uncertainty. Regarding the uncertainty in adaption, the authors in [19] define an uncertainty taxonomy, classify the uncertainty types and match them to MAPE-K stages. These requires to investigate the source of uncertainty and involve the uncertainty handling in the current techniques for performing self-adaptation [27], which are based on using parameters that change the behavior, changing the structure (i.e., reconfiguration), or changing the context. For instance, some frameworks [37,8] investigate the context that introduces uncertainty in behavior. In [30], the authors present a context taxonomy in addition to a 3-layer framework to design context-aware systems. The authors in [32] aim at reducing the impact of uncertainty in quality evaluation. This is done by defining uncertainty taxonomy and study their sources. The study shows that multiple uncertainties could impact model-based quality evaluation. In [34], the study aims at defining taxonomy of uncertainty types, template for their sources, occurrence, and their effect on requirement, design and runtime levels.

Regarding uncertainty in requirements engineering, the classical RELAX [41] approach captures weakening requirements according to environmental conditions (i.e., uncertainty) in runtime. Even though combing RELAX with SysMLKaos [2] allows the developer to consider non-functional requirements, it does not consider the development of the system nor provides mechanisms to fulfill the conditional requirements.

In self-adaptive systems, many works are handling different kinds of uncertainty using probabilities and learning. For instance, the Stich language [14] is used in Rainbow framework [16] that employs MAPE-K model. It introduces tactics and strategies as basic concepts for supporting dynamicity. It allows the developer to describe the likelihood of evaluating the condition of strategy selection to true. Using formal approach, [38] presents stochastic multi-mode systems (SMMS) that approximate the action of a moving vehicle, so it satisfies almost-sure reachability, which is the movement within a certain safety corridor. As for uncontrollable entities, [6] introduced a proactive adaptation to capture possible dangerous situations using prediction over historical data (i.e., fire prediction). For unforeseen situations, NiFti project [26] uses human-robot cooperation to achieve the goal in rescue missions. More specifically, the robots can alter a predefined plan to utilize the resources, and depending on robot updates on a 3D map the rescue can change the path for robots to avoid obstacles. Even though the previous work considered controllable/uncontrollable entities, they do not consider evaluating the risk/loss tradeoff.

Systems of the Industry 4.0 domain can be seen as a special case of Smart Cyber-Physical Systems (sCPS) [11], which also exhibit a high level of uncertainty. This has been already partially studied, e.g., in the scopes of the ASCENS and Quanticol projects [28,35]. Also, we partially addressed statistical modeling of human behavior in sCPS [10] and adaptation via meta-adaptation strategies [21] however a complete approach for sCPS is yet missing.

In summary, there are classifications and approaches for uncertainty. However, so far to our knowledge no combination of access control approaches and uncertainty exists.

2.2 Classification of uncertainty in access control

We applied, adopted, and condensed the classification of uncertainty from Perez-Palacin et al. [32] to better fit the needs of uncertainty in access control in Industry 4.0 systems. The adapted classification consists of three dimensions: Levels of uncertainty, Nature, and Source. The first two dimensions are taken from [32]. The last one is added to better categorize software architecture.

Levels of uncertainty categorize uncertainty by the degree of awareness. There are four levels. We removed the fifth from the original classification because it is—in our eyes—not practically applicable. The first level is that no uncertainty exists. In our case, this means that the system can decide without guessing what the correct access rules are. The second level introduces uncertainty but the system is aware of it and has appropriate measures to handle it. One solution to handle this is fuzzy logic for access control like used in [13]. The third level adds situations where the system is unaware of the existence of uncertainty. In the field of security, a component of an access control system can fail and deny access for everyone. In that case, the uncertainty is about the operability of the access control system. One solution for this might be a continuous monitoring approach similar to [23], which will trigger an adaptation process. This moves the uncertainty to level two because the system becomes aware of the uncertainty. The fourth and last level is that there exists no process to find out that the system has uncertainties. In general, this should be avoided [32].

Nature distinguishes between epistemic and aleatory. We reuse this category from [32] unchanged. Epistemic means that uncertainty exists because there is not enough data available. In policy mining approaches such as the one in [15], uncertainty might exist if the log data does not consist of every necessary case. Aleatory describes a situation, which is too random to consider. For instance, this might be the break-down of a security sensor because of vandalism.

Source describes where uncertainty for access control can be found in the modeled system. We distinguish between *system structure*, *system behavior*, and *system environment*. We used this three subcategories, since systems consist of at least a structure, a behavior and an environment and in everyone of these uncertainty can exist. However, this is not an exclusive categorization, because scenarios could fall into multiple of these categories. System structure is comparable to the *model structure* from [32]. The *system structure* describes the design of the system. It consists of for example components, hardware resources, the wiring of components via required and provided interfaces. The *system behavior* describes the uncertainty in the actual behavior of the system. This can be for example the uncertainty about the intended usage. In access control the behavior is often regulated by access control rules. These rules might introduce uncertainty, if they are incorrect. However, they might also help to handle uncertainty. Therefore, we would count access control rules to the system behavior. The last subcategory is the *system environment*. The *system environment* describes the context in which a system is executed. This includes also the input data for the system. For instance this might be that due to bad sensor data, which is the input data the system cannot produce an accurate result for the location of an user and therefore decides s/he is not in compound and marks her/him as unauthorized.

3 Representative examples/use-cases

As a basis of representative examples, we are using a use-case [5] from our previous project Trust4.0[3], which focused on dynamic security. The use-case is simple however fully realistic as it has been developed together with and is based on interviews with industrial experts in the project. Within the project we created an approach for managing access control suitable for highly dynamic environment of Industry 4.0. The approach is based on application of dynamic security rules. Nevertheless, during the project we encountered several important situations, where uncertainty prevented formulating or even foreseeing strict access control rules. This requires a foundational change in the approach how the access control rules can be designed, verified and enforced, such that the uncertainty can be explicitly represented, tackled and reasoned about.

[3]`http://trust40.ipd.kit.edu/home/`

In the light of the classification shown in Section 2.2, we compare these categories against experiences of these situations we gathered together with our industry partners. We examined how the different types of uncertainty in the scenarios can be located within the given categories. We are focusing on the second and third *level* of uncertainty (the first level represents no uncertainty whereas in the fourth, uncertainty cannot be managed). As for *nature*, we handle both epistemic and aleatory uncertainty – this is necessary because if in a decentralized system an unexpected situation occurs, it is imperative to make a reaction regardless whether the situation is completely random or just not fully known. Importantly, we also assume that as the reaction typically has to be immediate there is no scope to obtain unknown data which would normally be one way to handle epistemic uncertainty. Similarly, we address the different subcategories in the *source* section. We describe the application of the *source* categorization with each example.

The use-case [5] assumes a factory with multiple working places where groups of workers work in shifts. The workers have access only to the workplace to which they have been assigned. Also, before the shift, they have to collect protective gear from a dispenser, without which they are not allowed to enter the workplace. The workplace is equipped with machines that can monitor their state and also can be reconfigured. The actual detailed log of what the machine did and what its internal settings are is confidential as it constitutes the intellectual property of the factory. To support the production in the factory, there is a constant in-flow of trucks bringing in material and taking out finished products. The truck is allowed to enter the factory only when it is designated so in the schedule and to enter the factory it must use a designated gate within a designated time interval.

Primarily, the access-control permissions are of two kinds: *allow* and *deny* (in a case of inconsistencies, evaluation order is allow–deny).

The identified examples of situations, in which the system encounters a state that was not anticipated and the access-control rules do not count with it, are as follows:

Example 1 A dispenser breaks and stops distributing the protective gear which is required to enter a shift. The system has to allow a foreman to open the dispenser so as to distribute the protective gear manually. Applying our source categorization, this falls into system structure and behavior, since the failing dispenser would be a structure problem and the opening and the distribution of gear by the foreman would be a different system behavior.

Example 2 An access gate is disconnected from the central authorization service and thus prohibits anyone to pass through because the access cannot be verified (this is actually quite a commons situation of gates letting the trucks inside the factory). The system has to allow the security personnel on the gate to manually define who can pass through. This would also fall into the system structure since the access gate would be a missing component and the system behavior category since the default behavior is changed.

Example 3 A machine is broken and repair requires that a third-party repairman has access to internal machine logs. In order to do the job, the repairman requires access data summaries which are anonymized over several shifts. As the repairman arrives at the place, it turns out that access to the data cannot be given because the data cannot be properly anonymized because the last operation was not long enough to collect the required data points that are needed to ensure proper anonymization. The source categorization would categorize this example into system structure since the broken machine introduces uncertainty, and system environment since the input data adds uncertainty whether machine can be repaired by the technician or not.

Example 4 An unexpected and unauthorized person appears at a workplace. By the system design this cannot happen because the person would have to pass a security gate. In this case, the system should dynamically enable the foreman or some other trusted person in the vicinity to access information allowing them to determine the person identity and reason to be there before the security personnel is called. As for our source categorization this would be in the category environment, since the context (here attendance of person) introduces uncertainty to the system.

In all the examples above, the system needs to autonomously take a decision which is beyond its pre-designed access control rules. In doing so, it has to evaluate how the access-control rules should be adapted in order to minimize the potential risk and loss (i.e., what is risked if the access-control rules are weakened and what can be lost, if the rules are strictly followed and not weakened).

3.1 Examples analysis

Here, we analyze the presented situations from multiple different views in order to build the situational patterns in the next section.

In *Example 1*, the system gives rights to someone, who is already trustworthy. Particularly, the foreman is responsible for the whole shift and has rights to access personal information about all the workers in his/her shift and has overall responsibility of the shift. Thus, assigning him/her the rights for the dispenser does not represent a significant security issue. On the other hand, not to assign the rights means that the shift cannot be executed without the protective gear and there might be a significant loss for the company.

In *Example 2*, the situation is similar from the risk/loss view. Again, the access right (for the gate now) is assigned to someone, who is trustworthy and in fact already has a superior access right (the security personnel is responsible for all the entries to the factory area anyway).

Nevertheless, there is an important difference between *Example 1* and *Example 2* which is characterized by the question which component in the security chains is broken. In the *Example 1*, the dispenser is a terminal component in the chain. Thus, the foreman needs to be assigned with a additional access right (open the dispenser), however no one (foreman, workers) has it currently assigned. In *Example 2*, the gate is disconnected and cannot verify access rights. The broken part here is an intermediate component in the security chain which assigns the access rights. The security personnel thus replaces a component (the gate) in the chain and the scenario is as before.

The *Example 3* is a different one. Here, the security risk is that the repairman can see unanonymized data and the loss is that the shift cannot proceed (which can lead to loss of profit for the company). However, the repairman typically has signed a kind of NDA (non-disclosure agreement) as even only via his/her presence in the company, he/she is eligible see proprietary information. Thus, relaxing on having the access right for seeing unanonymized data does not represent a significant issue (the anonymized data are a second level of protection—the first one is NDA).

Example 4 is quite close to *Example 1*. Here, the system has to give additional rights to someone, who is already trustworthy (the foreman is responsible for the whole shift and all workers in the shift).

3.2 Summary

Based on the analysis from the previous section, we can identify two dimensions defining a space, in which new rules are created: (i) whether something is allowed or denied (ii) what is done with access rights.

For the first dimension, the options are obvious, either the new rule works with: (1) the *allow* permission, or with (2) the *deny* permission.

For the second dimension, the options are: (A) a permission is given to a component, (B) decision about a permission assignment is delegated to a component, (C) a permission is removed.

Table 1 maps the examples to the space of above defined dimensions. The cases not covered by the examples above, can be exemplified as follows (in the table marked as *Post-hoc* examples): For the 1xC case (removing the *allow* permission, *Post-hoc C* example)—if the foreman tries to read information not accessible to him/her, it is evaluated as a potential security attack and all his/her access rights are removed. For the 2xA case (adding the *deny* permission to a component, *Post-hoc A* example)—if the repairman starts to read data unrelated to the machine/shift, new rule with the deny permission for him/her is created. For the 2xB case (delegating the *deny* permission to a component, *Post-hoc B* example)—as in the previous one, if the repairman starts to read data unrelated to the machine/shift, new rule delegating the *deny* permission to the foremen is created.

	A	B	C
1	Example 1, 4	Example 2	Post-hoc C
2	Post-hoc A	Post-hoc B	Example 3

Table 1: 1^{st} vs 2^{nd} dimension

Also, from the analysis, we can observe that the typical reasons that someone obtains new permissions is

(a) he/she already has a role that implies governance over part of a system for which the new permission is to be granted – thus the new permission does not extend the scope of governance of the subject, it only completes it,

(b) he/she has an equivalent role to someone who already has the permission,
(c) a risk connected with obtaining the new permission is low compared to the loss connected with not obtaining the permission.

Similarly, we can observe that the typical reasons that someone is loosing permissions is that he/she is trying to perform a suspicious operation and thus, as a preventive measure, he/she looses access.

Note that item (c) subsumes (a) and (b). However, we still list (a) and (b) separately because these conditions are easier to establish. Whereas the comparison of risk vs. loss is typically difficult to do. In situations when this ratio cannot be reliably done, it is necessary to assume that the risk is too high.

4 Situational patterns for uncertainty

With the analysis performed, we can define the situational patterns, which serve as a strategy for dynamic adaptation of security access rules.

For describing the patterns, we use a format inspired by the classical books on patterns [18,12], however we have updated the format to our needs (which is a common and recommended practice [20], i.e., to update the format to own needs as the content is more important than the form). Our format is: (i) *Name of the pattern*, (ii) *Solution* (description of the pattern), (iii) *Context* (determination of components and their behavior where the pattern is applied), (iv) *Consequences* (in our cases, mainly the risk discussion), and (v) *Example*.

There are three identified patterns following the second dimension from Section 3. Plus, there are sub-variants following the first dimension in those case where the division is necessary.

4.1 Pattern 1a – Adding an *allow* rule

Solution A new situation cannot be handled with currently assigned permissions — a new *allow* permission needs to be assigned, i.e., a new security access rule assigning the *allow* to a component is added to the system.

Context The *allow* permission, i.e., a rule with the *allow* permission, is assigned to a component, which either has: (a) such a role in the system that the new rule does not fall outside the component's area of competence, or (b) a similar role in the system as a component that already has the same rule.

Consequences By adding the *allow* permission, the affected component can have higher access within the system than originally intended and it might lead to a potentially dangerous situations. Thus the trade-off has to be greater for adding the *allow* permission than for not adding it (and therefore leaving the system in a non-functional state).

Example The *Examples 1* and *4* are direct representatives of this pattern.

4.2 Pattern 1b – Adding a *deny* rule

Solution A potentially dangerous situation occurs in the system. The *deny* permission is assigned to the component (i.e., a new security access rule assigning the *deny* to a component is added to the system).

Context A component has started to misbehave—accessing more than is usual and/or necessary for it. As a security measure, the *deny* rule is assigned to the component.

Consequences The situation here is reversed to the *Pattern 1a*, i.e., the trade-off has to be greater for limiting access right for the affected component.

Example The *Post-hoc A* example is direct representative of this pattern.

4.3 Pattern 2a – Removing an *allow* rule

Solution A potentially dangerous situation occurs in the system. The *allow* permission is removed from the component (i.e., an existing security access rule assigning the *allow* to a component is removed from the system).

Context A component has started to misbehave and or is broken. As a security measure, the *allow* rule is removed from the component. The pattern is very similar to the *Pattern 1b*—the difference is that the *Pattern 1* is used when there is no rule to be removed.

Consequences The situation here is the same as for the *Pattern 1b*

Example The *Post-hoc C* example is direct representative of this pattern.

4.4 Pattern 2b – Removing a *deny* rule

Solution The system runs in a situation that is blocked by a rule with the *deny* permission. The *deny* permission is removed from the component (i.e., an existing security access rule assigning the *deny* to a component is removed from the system).

Context The system can continue in the common operations only if a component can access an entity (e.g., another component) but there is a rule *denying* the access. The rule is removed.

Consequences The rule can be removed only in the case the rule represents redundancy in the security chain.

Example The *Example 3* is direct representative of this pattern.

4.5 Pattern 3 – a new access rule validator

Summary The system runs in a situation that is blocked by a component that validates access for other components (e.g., the component is broken). Another component is chosen as a replacement and serves as a new validator.

Context The selected component has to already have a supervisor-like role in the system.

Consequences As the selected component has to already have a supervisor-like role, the risk of assigning additional permissions to it is minimized.

Example The *Example 2* is direct representative of this pattern for the *allow* permission and the *Post-hoc B* for the *deny* permission.

5 Applying patterns in an adaptation framework

zAs we described in [3], we model dynamic security rules as ensembles. This allows us to target dynamic security in collective adaptive systems. Ensembles are instantiated dynamically to reflect ever changing situations and collaborations in a system.

An ensemble definition is static in terms which permission it assigns and the predicate identifying components it applies to (i.e., subjects and objects of the permissions). The dynamicity comes from the fact that an ensemble is instantiated at runtime for each group of components that match the roles and constraints in the ensemble. The components are identified by their state. As this state changes throughout the lifetime of the system, the selection is dynamic.

From the architecture perspective, we model the system as an adaptive system, where security ensembles generate access control rules that are understood by legacy systems. This is the first-level of adaptation as shown in Figure 1 – controlled by an Adaptation Manager. In this paper, we see the adaptation as decentralized. As such, we assume multiple Adaptation Managers, each of which instantiates ensembles in its domain of control and determines access control rules pertaining to particular subjects and objects.

To account uncertainty that is addressed by the situational patterns as presented in this paper, we build on our approach to architectural homeostasis [22] and incorporate the patterns described in this paper as a meta-adaptation layer (i.e., a layer that adapts the ensembles themselves) as visualized in Figure 1.

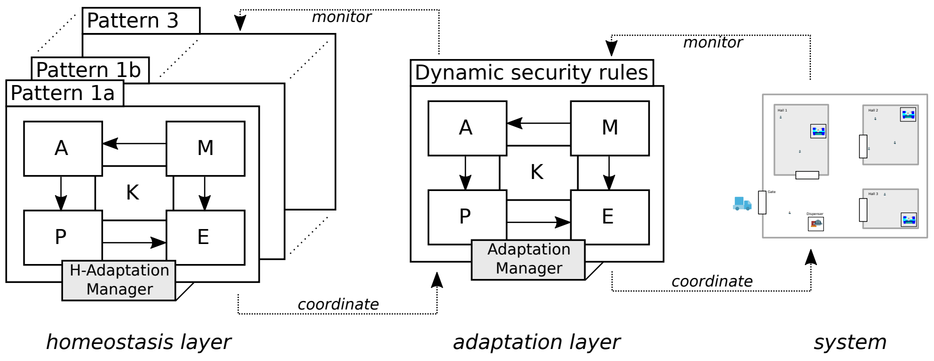

Fig. 1: Meta-adaptation framework

The idea is that each pattern is reflected as one strategy of the meta-adaptation layer. The strategy modifies existing ensembles that implement the dynamic security of the collective adaptive system. This extra layer extends the adaptation space of the system and helps tackling situations that were not fully anticipated and lie beyond the scope of the system states addressed by the security ensembles (i.e., the middle layer in Figure 1)

Each pattern is represented as a MAPE-K adaptation loop that monitors the system for unanticipated situations targeted by the particular pattern. The pattern also determines the dynamic security rules that have to be introduced to tackle the unanticipated situation.

Listing 1 shows a brief excerpt of the security specification via ensembles for our factory use-case. The specification is written in our DSL, which is created as an internal DSL in the Scala language.

```
class TestScenario() extends Model {
  object CAS extends Component { /*...*/ }
  class Gate(/*...*/) extends Component { /*...*/ }
  /* ... */
  class FactorySystem(factory: Factory) extends RootEnsemble {
    class GateAccess(gate: Gate) extends Ensemble {
      initiatedBy(CAS)
      val assignedTransports = transports.filter(tr => tr.assignedGate == gate)
      class TransportAccessThroughGate(transport: Transport) extends Ensemble {
        situation {
          (now isEqualOrAfter (transport.scheduledArrival minusMinutes 5)) &&
            (now isEqualOrBefore (transport.scheduledArrival plusMinutes 15)) &&
            transport.assignedGate == gate
        }
        allow(transport, Enter, gate)
      }
      val transportAccesses = rules(transports.filter(tr => tr.assignedGate == gate).map(tr => new
            TransportAccessThroughGate(tr)))
    }

    class ShiftTeam(shift: Shift) extends Ensemble {
      initiatedBy(CAS)
      object AccessToFactory extends Ensemble {
        situation {
          (now isEqualOrAfter (shift.startTime minusMinutes 45)) &&
            (now isEqualOrBefore (shift.endTime plusMinutes 45))
        }
        allow(shift.foreman, Enter, shift.workPlace.factoryBuilding)
        allow(assignedWorkers, Enter, shift.workPlace.factoryBuilding)
      }
      object AccessToDispenser extends Ensemble {
        situation {
          (now isEqualOrAfter (shift.startTime minusMinutes 40)) &&
            (now isEqualOrBefore shift.endTime)
        }
        allow(shift.foreman, Use, shift.workPlace.factoryBuilding.dispenser)
        allow(assignedWorkers, Use, shift.workPlace.factoryBuilding.dispenser)
      }
      object AccessToWorkplace extends Ensemble { /* ... */ }
      object AccessToMachine extends Ensemble { /* ... */ }
      object NoAccessToMachineSensitiveDataOtherThanFromWorkplace extends Ensemble { /* ... */ }
      object AccessToBrokenMachine extends Ensemble {
        val assignedRepairmen = repairmen.filter(rm => rm.machine == shift.workPlace.machine)
        allow(assignedRepairmen, Read("logs"), shift.workPlace.machine)
      }
      deny(repairmen, Read("*"), shift.workPlace.machine, PrivacyLevel.SENSITIVE)
      /* ... */
      rules(
        AccessToFactory, AccessToDispenser, AccessToWorkplace,
        AccessToMachine, AccessToBrokenMachine, CancellationOfWorkersThatAreLate
      )
    }
    val shiftTeams = rules(shifts.filter(shift => shift.workPlace.factoryBuilding.factory ==
          factory).map(shift => new ShiftTeam(shift)))
    val gateAccessRules = rules(gates.map(gate => new GateAccess(gate)))
  }
  val factoryTeam = root(new FactorySystem(factory))
}
```

Listing 1: Original security specification

Below, we overview the parts of the specification important to this paper. Details about the syntax and semantics of DSL for security ensemble specifications are available at [24].

Components are used to represent entities in the system that (a) can be assigned access control, (b) are subject of access control, or (c) can determine the access control by controlling formation of security ensembles (i.e., acting as the Adaptation Manager in the middle layer).

The components are listed in lines 2-4. These represent components for the Gate, Factory, Workplace, Central−Access−System (CAS), etc. but also components that cannot be directly controlled by the system, but still are relevant to access control—like Workman, Repairman, etc.

Then, the ensembles representing security specifications are defined. The ensembles are hierarchical, which allows for more simple definition thanks to decomposition. The FactorySystem ensemble (line 5) represent the whole system. The GateAccess (line 6) ensemble controls access through the gate to the factory. This ensemble is initiated (i.e., its instantiation is controlled by) the CAS component and it has a single further subensemble TransportAccessThroughGate (line 9), which is instantiated for each transport coming to the factory. Here, if the transport satisfies the condition defined in the situation (line 10), it is allowed to enter through the gate via the allow rule (line 15). Similarly to the GateAccess ensemble, the ShiftTeam ensemble (line 20) controls access of workers (and other persons) to and within the factory. It is also decomposed to several subensmbles controlling access to individual elements of the factory, i.e., the AccessToFactory ensemble controlling access to the factory, the AccessToDispenser controlling access to the dispenser for headgear, and so on.

In addition to the allow rules, the specification also lists deny rules. The semantics is allow-deny, meaning that a deny rule overrides any allow rules. The deny rules are used in the specification to express cross-cutting policies—e.g., that no external repairman should get access to sensitive data. We assume that all the security to be primarily specified via the allow rules. The deny rules thus act more as assertions to detect inconsistencies in the specification.

As mentioned above, the H-adaptation manager monitors the system for unanticipated situations and introduces new ensembles and rules to the security specification of the system. Particularly for our use-case, if the H-adaptation manager detects a situation corresponding to the *Example 1* (the broken dispenser, i.e., the **Pattern 1a**), it updates the AccessToDispenser ensemble with the additional allow rule and thus, the ensemble rules will look as follows:

```
object AccessToDispenser extends Ensemble {
  /* ... */
  allow(shift.foreman, Use, shift.workPlace.factoryBuilding.dispenser)
  allow(shift.foreman, Open, shift.workPlace.factoryBuilding.dispenser) // ADDED
  allow(assignedWorkers, Use, shift.workPlace.factoryBuilding.dispenser)
}
```

Listing 2: Updated specification based on Pattern 1a

Similarly, if the H-manager detects a situation corresponding to the *Example 3* (the broken machine, i.e., the **Pattern 2b**), it removes the deny rule at line 45 (in Listing 1) disallowing the repairman to read sensitive data.

If the H-manager detects a situation corresponding to the *Example 2* (the disconnected gate, i.e., the **Pattern 3**), it updates the GateAccess ensemble so it is initiated not by the CAS component but by the security personnel, i.e., it starts as follows:

```
class GateAccess(gate: Gate) extends Ensemble {
  initiatedBy(gateSecurityHeads(gate)) // instead of initiatedBy(CAS)
```

Listing 3: Updated specification based on Pattern 3

6 Conclusion

In this paper, we have presented access-control related situational patterns, which serve as meta-adaptation strategies in cases when an unanticipated situation is encountered in the system. The patterns primarily target the domain of Industry 4.0, however they are applicable to other similar domains of modern smart cyber-physical system. They are based on our experience gained from participating in industrial projects. The patterns represent a first step for self-adaptation of security management.

Currently, we are continuing with the implementation of the adaptation framework and incorporating the patterns there. As an ongoing work, we investigate further industrial-based examples and update the patterns correspondingly.

Acknowledgment

This work has been funded by the DFG (German Research Foundation) – project number 432576552, HE8596/1-1 (FluidTrust), supported by the Czech Science Foundation project 20-24814J, and also partially supported by Charles University institutional funding SVV 260451.

References

1. Abreu, V., Santin, A.O., Viegas, E.K., Stihler, M.: A multi-domain role activation model. In: Proc. of of ICC 2017, Paris, France. pp. 1–6. IEEE, Paris, France (2017)
2. Ahmad, M., Gnaho, C., Bruel, J.M., Laleau, R.: Towards a Requirements Engineering Approach for Capturing Uncertainty in Cyber-Physical Systems Environment. In: New Trends in Model and Data Engineering, CCIS, vol. 929, pp. 115–129 (2018)
3. Al-Ali, R., Bures, T., Hnetynka, P., Krijt, F., Plasil, F., Vinarek, J.: Dynamic Security Specification Through Autonomic Component Ensembles. In: Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems. p. 172–185 (2018)
4. Al Ali, R., Bures, T., Hnetynka, P., Matejek, J., Plasil, F., Vinarek, J.: Toward autonomically composable and context-dependent access control specification through ensembles. International Journal on Software Tools for Technology Transfer (2020)
5. Al-Ali, R., Hnetynka, P., Havlik, J., Krivka, V., Heinrich, R., Seifermann, S., Walter, M., Juan-Verdejo, A.: Dynamic security rules for legacy systems. In: Proc. of ECSA 2019 (vol 2), Paris, France. pp. 277–284. ACM (2019)
6. Anaya, I.D.P., Simko, V., Bourcier, J., Plouzeau, N., Jézéquel, J.m.: A Prediction-Driven Adaptation Approach for Self-Adaptive Sensor Networks. In: Proc. of SEAMS 2014, Hyderabad, India. pp. 145–154 (2014)

7. Argento, L., Margheri, A., Paci, F., Sassone, V., Zannone, N.: Towards Adaptive Access Control. In: Data and Applications Security and Privacy XXXII, vol. 10980, pp. 99–109 (2018)
8. Baudry, G., Macharis, C., Vallée, T.: Range-based Multi-Actor Multi-Criteria Analysis: A combined method of Multi-Actor Multi-Criteria Analysis and Monte Carlo simulation to support participatory decision making under uncertainty. European Journal of Operational Research **264**(1), 257–269 (2018)
9. Ben Abdelkrim, I., Baina, A., Feltus, C., Aubert, J., Bellafkih, M., Khadraoui, D.: Coalition-OrBAC: An Agent-Based Access Control Model for Dynamic Coalitions. In: Trends and Advances in Information Systems and Technologies, vol. 745, pp. 1060–1070 (2018)
10. Bures, T., Plasil, F., Kit, M., Tuma, P., Hoch, N.: Software Abstractions for Component Interaction in the Internet of Things. Computer **49**(12), 50–59 (2016)
11. Bures, T., Weyns, D., Schmer, B., Fitzgerald, J.: Software Engineering for Smart Cyber-Physical Systems: Models, System-Environment Boundary, and Social Aspects. ACM SIGSOFT Software Engineering Notes **43**(4), 42–44 (2019)
12. Buschmann, F. (ed.): Pattern-oriented software architecture: a system of patterns. Wiley (1996)
13. Cheng, P.C., Rohatgi, P., Keser, C., Karger, P.A., Wagner, G.M., Reninger, A.S.: Fuzzy Multi-Level Security: An experiment on quantified risk-adaptive access control. In: Proc. of SP '07, Berkeley, USA. pp. 222–227 (2007)
14. Cheng, S.W., Garlan, D.: Stitch: A language for architecture-based self-adaptation. Journal of Systems and Software **85**(12), 2860–2875 (2012)
15. Cotrini, C., Weghorn, T., Basin, D.: Mining ABAC Rules from Sparse Logs. In: Proc. of EURO S&P 2018, London, UK. pp. 31–46 (2018)
16. Cámara, J., Garlan, D., Kang, W.G., Peng, W., Schmerl, B.R.: Uncertainty in Self-Adaptive Systems Categories , Management , and Perspectives. Report CMU-ISR-17-110, Institute for Software Research School of Computer Science Carnegie Mellon University, Pittsburgh, PA 15213 (2017)
17. De Capitani di Vimercati, S., Samarati, P.: Mandatory Access Control Policy (MAC) BT - Encyclopedia of Cryptography and Security. p. 758. Springer (2011)
18. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley Professional (1994)
19. Esfahani, N., Malek, S.: Uncertainty in Self-Adaptive Software Systems. In: Software Engineering for Self-Adaptive Systems II. Springer (2013)
20. Fowler, M.: Writing Software Patterns (2006), `https://www.martinfowler.com/articles/writingPatterns.html`
21. Gerostathopoulos, I., Bures, T., Hnetynka, P., Hujecek, A., Plasil, F., Skoda, D.: Strengthening Adaptation in Cyber-Physical Systems via Meta-Adaptation Strategies. ACM Transactions on Cyber-Physical Systems **1**(3), 1–25 (2017)
22. Gerostathopoulos, I., Škoda, D., Plášil, F., Bureš, T., Knauss, A.: Tuning Self-Adaptation in Cyber-Physical Systems through Architectural Homeostasis. Journal of Systems and Software **148**, 37–55 (2019)
23. Heinrich, R.: Architectural runtime models for integrating runtime observations and component-based models. Journal of Systems and Software **169** (2020)
24. Hnetynka, P., Bures, T., Gerostathopoulos, I., Pacovsky, J.: Using Component Ensembles for Modeling Autonomic Component Collaboration in Smart Farming. In: Proc. of SEAMS 2020, Seoul, Republic of Korea (2020)
25. Hu, V.C., Kuhn, D.R., Ferraiolo, D.F.: Attribute-Based Access Control. Computer **48**(2), 85–88 (2015)

26. Kruijff, G., et al.: Designing, developing, and deploying systems to support human–robot teams in disaster response. Advanced Robotics **28**(23), 1547–1570 (2014)
27. Krupitzer, C., Roth, F.M., VanSyckel, S., Schiele, G., Becker, C.: A survey on engineering approaches for self-adaptive systems. Pervasive and Mobile Computing **17**, 184–206 (2015)
28. Latella, D., Loreti, M., Massink, M., Senni, V.: Stochastically timed predicate-based communication primitives for autonomic computing. Electronic Proceedings in Theoretical Computer Science **154**, 1–16 (2014)
29. Li, X., Eckert, M., Martinez, J.F., Rubio, G.: Context Aware Middleware Architectures: Survey and Challenges. Sensors **15**(8), 20570–20607 (2015)
30. Lu, Y.: Industry 4.0: A survey on technologies, applications and open research issues. Journal of Industrial Information Integration **6**, 1–10 (2017)
31. Mahdavi-Hezavehi, S., Avgeriou, P., Weyns, D.: A Classification Framework of Uncertainty in Architecture-Based Self-Adaptive Systems With Multiple Quality Requirements. In: Managing Trade-Offs in Adaptable Software Architectures, pp. 45–77. Elsevier (2017)
32. Perez-Palacin, D., Mirandola, R.: Uncertainties in the Modeling of Self-adaptive Systems: A Taxonomy and an Example of Availability Evaluation. In: Proc. of ICPE 2014, Dublin, Ireland. pp. 3–14 (2014)
33. Peruma, A., Krutz, D.E.: Security: a critical quality attribute in self-adaptive systems. In: Proc. of SEAMS 2018, Gothenburg, Sweden. pp. 188–189 (2018)
34. Ramirez, A.J., Jensen, A.C., Cheng, B.H.C.: A Taxonomy of Uncertainty for Dynamically Adaptive Systems. In: Proc. of SEAMS 2012, Zurich, Switzerland. pp. 99–108 (2012)
35. Reijsbergen, D.: Probabilistic Modelling of Station Locations in Bicycle-Sharing Systems. In: Software Technologies: Applications and Foundations. pp. 83–97 (2016)
36. Sartoli, S., Namin, A.S.: Modeling adaptive access control policies using answer set programming. Journal of Information Security and Applications **44**, 49–63 (2019)
37. Sharif, M., Alesheikh, A.A.: Context-aware movement analytics: implications, taxonomy, and design framework: Context-aware movement analytics. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **8**(1) (2018)
38. Somenzi, F., Touri, B., Trivedi, A.: Almost-Sure Reachability in Stochastic Multi-Mode System. arXiv:1610.05412 (2016)
39. Verma, D., Calo, S., Chakraborty, S., Bertino, E., Williams, C., Tucker, J., Rivera, B.: Generative policy model for autonomic management. In: Proc. of IEEE SmartWorld 2017, San Francisco, USA. pp. 1–6 (2017)
40. di Vimercati, S.D.C.: Discretionary Access Control Policies (DAC), pp. 356–358. Springer US, Boston, MA (2011)
41. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H., Bruel, J.M.: RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. In: Proc. of RE 2009, Atlanta, USA. pp. 79–88 (2009)
42. Yang, Y., Zheng, X., Guo, W., Liu, X., Chang, V.: Privacy-preserving smart IoT-based healthcare big data storage and self-adaptive access control system. Information Sciences **479**, 567–592 (2019)