Dynamic Security Rules for Legacy Systems

Rima Al-Ali
Charles University
Prague, Czech Republic
alali@d3s.mff.cuni.cz

Petr Hnetynka
Charles University
Prague, Czech Republic
hnetynka@d3s.mff.cuni.cz

Jiri Havlik
Institut of Microelectronic Applications
Prague, Czech Republic
havlik@ima.cz

Vlastimil Krivka
Institut of Microelectronic Applications
Prague, Czech Republic
krivka@ima.cz

Robert Heinrich
Karlsruhe Institute of Technology
Karlsruhe, Germany
robert.heinrich@kit.edu

Stephan Seifermann
Karlsruhe Institute of Technology
Karlsruhe, Germany
stephan.seifermann@kit.edu

Maximilian Walter
Karlsruhe Institute of Technology
Karlsruhe, Germany
maximilian.walter@kit.edu

Adrian Juan-Verdejo
CAS Software
Karlsruhe, Germany
adrian.juan@cas.de

ABSTRACT

Industry 4.0 tries to digitalize the production process further. The digitalization is achieved by connecting different entities (machines, worker) to data-exchange, which needs to be dynamic and to adapt to different changing situations and members in the process. However, just exchanging data might lead to confidentiality issues. The data-exchange needs to be protected to secure the confidentiality and trust in the system. Therefore, security rules need to adapt to these dynamic situations. One part of a possible solution might be dynamic access control rules. However in many cases, existing "legacy" systems are reused, which can in not handle dynamic access control rules. Due to this gap between the required and provided functionality, we propose an approach, which integrates dynamic access control based on the system-context into legacy systems. Our approach uses a security adaption controller, which dynamically adapts the access control rules to a new situation and integrates them into an existing legacy system. We discussed our approach with industrial practitioners and related our approach to their existing legacy system. In addition, we performed a scalability analysis to demonstrate the applicability of our approach in a realistic environment.

CCS CONCEPTS

• **Security and privacy** → *Domain-specific security and privacy architectures*; • **Computer systems organization** → *Self-organizing autonomic computing*.

SASI4 2019, September 9–13, 2019, Paris, France

ACM ISBN 978-1-4503-9999-9/00/00…$15.00
https://doi.org/10.1145/00000000000

KEYWORDS

autonomic systems, self-adaptive architecture, security rules, access control, legacy systems, Industry 4.0

ACM Reference Format:
Rima Al-Ali, Petr Hnetynka, Jiri Havlik, Vlastimil Krivka, Robert Heinrich, Stephan Seifermann, Maximilian Walter, and Adrian Juan-Verdejo. 2019. Dynamic Security Rules for Legacy Systems. In *Proceedings of First Workshop on Systems, Architectures, and Solutions for Industry 4.0 (SASI4 2019).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/00000000000

1 INTRODUCTION

Modern systems typically feature highly dynamic collaboration involving a high number of interconnected devices and other autonomous components. Such systems exhibit a high degree of dynamicity, meaning they constantly adjust their behavior and structure to the situation in their environment. Examples of such systems include smart buildings and cities, smart traffic and similar ones. Typically, these systems are composed of a number of sensors monitoring environment (temperature, level of traffic,…) and actuators (ventilation, traffic lights,…). Furthermore they typically encompass multiple components running in cloud, thus forming complex systems-of-systems that integrate cloud and edge-cloud services and embedded devices.

Security plays a very important role in these systems – be it the control of the physical access (e.g., monitoring position and controlling movement of persons by means of RFID readers and smart door locks) or the control of access to information held by smart system (e.g., regulating who has access to data about positions of other persons). In many cases, smart systems (as for instance in Industry 4.0, which we target in our projects and also in this paper) are built using of existing "legacy" systems which are extended and integrated to address the smart collaboration. Though this works well for the main business functionality, it fails to sufficiently address security (meaning access control in this paper), which is an inherently cross-cutting concern that a) spans through the integrated systems, and b) needs to address dynamically evolving situations.

Based on our experience from cooperation within industrial projects, the existing (legacy) systems mainly allow for static access control rules. For instance, in the case of physical access control, the doors are statically configured with a list of persons that may pass through and at which time window. However, for scenarios like Industry 4.0, this is not enough. Not only that the role of a company in a product chain can easily change, even the roles of individual persons can change during the day and thus different security rules may apply. Additionally, to reflect cases that arise when multiple teams are to work on a problem while at the same time there is a need to protect the intellectual property, there emerge rules that workers working on a product for one customer, may not be in the same workplace with workers working for another customer; or that workers of a third-party may get temporarily physical access and access to data in case they respond to some incident.

To solve these issues, we propose a self-adaptive architecture, where we introduce a dynamic security adaptation controller that adapts static security rules of existing legacy systems in order to adjust them to the current inter-system situation in hand. This controller is configured by dynamic security rules that are defined in via a DSL on the level of the whole smart system. This makes the rules explicit and avoids ad-hoc interdependencies of constituent legacy systems. The approach is based on our preliminary work [2, 3], where we have defined *security ensembles* that describe permitted interactions in the system and follow the system during its evolution. The approach presented in this paper brings it to the level of a self-adaptive architecture, introduces the concepts of the security adaptation controller and connects it to the legacy systems. Furthermore, this paper extends the semantics of security ensembles to allow fast evaluation of the dynamic security rules in order to make the approach scale to controlling accesses of tens of thousands of persons. We evaluate our approach on existing legacy systems, which are currently in use within multiple organizations.

The paper is organized as follows. Sect. 2 shows a running example and Sect. 3 presents our approach. In Sect. 4, the approach is evaluated and related work is discussed in Sect. 5. Sect. 6 concludes the paper.

2 RUNNING EXAMPLE

We model our running example to reflect real-life scenarios of our industrial partners. In the running example, we focus on physical access and access to personal data during a shift in a factory.

The factory has multiple working places (see the factory floor plan in Fig. 1). In each room, there is a team of workers led by a foreman. Each of these teams works on products for a different customer and thus workers from one team are not allowed to enter the room of another team. Plus, before the start of the shift, the workers have to take a headgear from the dispenser – otherwise they are not allowed to enter their workplace.

The work in the factory is organized to *shifts*, which are defined in the Human Resource Management system (HRM system). Each shift has assigned a workplace, time of start and end of the shift, a list of workers and a foreman. In addition to the workers, there is also a list of *standby* workers, which will be called in in case a worker assigned to the shift does not appear in time.

The scenario of the example progresses as follows: 1) Workers arrive at and pass through the main gate between 30-20 minutes before the start of the shift. The system grants them access to pass through the main gate. 2) Workers use the dispenser located at the main gate to obtain a protective headgear. The system grants them access to use the dispenser. 3) Then the workers proceed to the work place. The system again grants access to enter the workplace entrance to those workers who retrieved their protective gear from the dispenser.

The scenario has several dynamic situations: 1) If a worker does not arrive by 20 minutes before the start of the shift, the system notifies the foreman that a worker is likely to be late and grants the foreman temporary access to the worker's phone number and an estimate of distance to the factory. 2) If a worker does not arrive by 15 minutes before the start of the shift, his/her access is revoked and the system automatically selects a suitable stand-by worker and notifies the stand-by worker about being assigned to the shift and grants access to the stand-by worker to come to the shift.

The physical enforcement of access restrictions is performed by RFID readers and smart door locks (i.e., the doors to the particular workplace allows only workers with a headgear and from the particular shift).

This example is not a synthetic one but it is an actual situation obtained from an industrial partner in our projects. There is a high degree of dynamicity in the system; the doors in the factory cannot be statically configured with allowed workers. Even more, time-based configuration of the doors is not enough too, as the system has to deal with standby workers, i.e., once a standby worker is chosen for a shift, then the original replaced work must not be allowed for entering the workplace (even if he/she later arrives). Similar degree of dynamicity is present in the access to the personal data of a worker that is late. Here the HRM system is supposed to provide the data only if the worker does not pass through the main gate within a particular interval (which is knowledge present in the smart-lock system, not in the HRM system).

3 DYNAMIC SECURITY RULES

To allow for controlling security in a dynamic context of multiple interconnected systems (primarily legacy ones), we propose a variant of self-adaptive architecture. The main element of the architecture is a *dynamic security adaptation controller* (DSAC). It is responsible for dynamic adaptation of static security rules for legacy systems in order to configure them and adjust them to the current situation.

The whole approach employs the concept of autonomic ensembles [7]. An ensemble is a dynamically formed group of components that corresponds to a joint goal or a coordinated activity. Components to be part of an ensemble are selected at run-time, based on a membership condition of the ensemble. This condition is a predicate expressed over component types and their data.

In the case of using ensembles to control security, an ensemble determines a particular dynamically emerging situation, identifies components that take part in the situation and specifies access rules for the components. As the situation itself is dynamic, the access rules are also dynamic, i.e., they change in time as the components are admitted to and released from the ensemble and as the ensemble

comes to existence with the emergence of the situation and as the ensemble is dissolved when the situation passes off.

In contrast to the traditional notion of components, we do not control the components directly. The ensembles are used only to determine the situations among components and to determine the security rules. Since we do not assume the control of components, our approach works equally well for determining access of pure software components (e.g., HRM system), hardware devices (e.g., smart locks), places (e.g., factory hall) and even humans workers – all of them are uniformly treated as components. We only assume knowledge of some attributes of components (e.g., position, assignment of a person to a shift), which can be obtained from the existing legacy systems and can be attributed to a component.

In our approach, we distinguish between ensemble types and ensemble instances. An ensemble type can be thought of as a security rule describing how to identify a dynamic situation, components, and how to assign the access grants to the components. An ensemble instance is then created by DSAC for each instance of the given situation. The ensemble instance then comprises particular components instances. Naturally, as a similar situation may happen simultaneously at multiple places (e.g., when multiple shifts run at the same time), there may exist multiple simultaneous instances of the same ensemble type.

For instance, in our running example, one type of situation is when there is 30–20 minutes before the start of a shift (this is a situation determined by the temporal and spatial context – spatial because the shift takes place in a particular factory hall). The components that take part in this situation (i.e., in the corresponding security ensemble) are the workers assigned and the foreman to the shift. Within this situation, the corresponding security ensemble grants all workers access to pass through the factory gate (see the left group in Fig. 1).

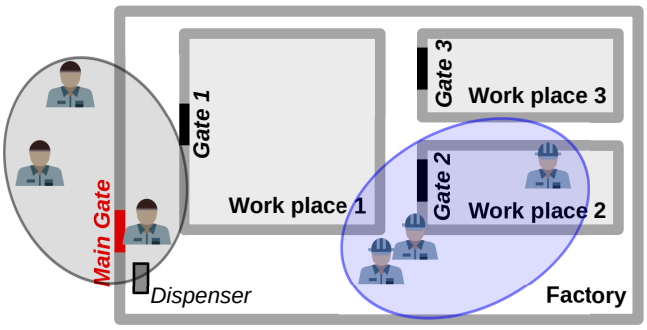


Figure 1: Ensembles in the running example

Another example is an ensemble that grants access to the work place to workers in the shift that retrieved their protective gear. This ensemble comes to exist 30 minutes before the shift till 30 minutes after the shift and it contains only workers with the protective gear (i.e., Worker components for which the smart lock access (SLoA) system registers that they have retrieved the gear). These workers are given access to pass through the door of the workplace where the shift takes place (the right group in Fig. 1).

In the rest of the section, we describe technical details of our approach. We start with the description of DSAC in Sect. 3.1, which, based on the knowledge of ensemble types, evaluates the situations and instantiates ensemble instances to determine the access grants. Then we present the domain specific language we use to describe ensembles and their semantics. We split the description to two parts – (a) to ensembles that describe how to grant access (Sect. 3.2), and (b) to ensembles that describe assertions about access grants (Sect. 3.3). Last, in Sect. 3.4, we overview how the privacy levels are composed when data are combined from different sources with different privacy levels and when the data are sufficiently minimized [19] (i.e., limiting data to achieve a required privacy level). Determining the privacy level of derived data then allows us to evaluate whether an assertion like "In no case, any ensemble can give a foreman access to privacy sensitive information about a worker" conflicts with giving a foreman ability to read an estimated distance to factory of a worker that is late.

3.1 Dynamic Security Adaptation Controller

In our approach, we follow the traditional adaptive architecture, which consists of a controller and the system(s) under control. The legacy systems and entities to be managed (humans, doors, locks, etc.) are *systems under control* and DSAC is their *controller*. Contrary to the traditional use of adaptive architecture, we do not adapt the functional aspects of the legacy systems under control, but rather their set of security rules/permissions. Furthermore, we control several systems at once, which allows us to reflect situations that comprise multiple systems and thus cannot be well addressed on the level of a single legacy system.

Within DSAC, we employ the standard MAPE-K loop [16] as follows: (A) *Monitoring:* DSAC collects data about the current situation. In our running example, DSAC monitors position of all the workers in the factory (obtained via the SLoA system) and obtains data about the shifts and assignment of workers to shifts (retrieved from the HRM system). (B) *Analysis:* DSAC instantiates ensembles to reflect the situations. Internally, the ensemble instances are determined by translating the specification of security ensembles to a constraint satisfaction problem (CSP). A CSP solver is then applied to find a model for the logical theory described by the CSP. (C) *Planning:* Each of the ensemble instances (created in the Analysis phase) specifies particular access grants. These access grants are collected and it is determined which permissions in the legacy systems (HRM, SLoA) have to be updated and how. (D) *Execution:* DSAC applies the permissions change set (determined in the Planning phase) to the legacy systems. As a result, these systems are reconfigured to comply with the current situation. Reconfiguration is performed atomically, i.e., the whole system remains in a consistent state.

3.2 Specifying security rules via ensembles

For easiness in prototyping, we have developed our specification as an internal DSL in Scala programming language[1]. Production oriented version of our framework will likely require a more non-expert friendly external DSL, however we take a liberty in this paper to explain the concepts on the Scala-based internal DSL. Though it is a bit more demanding for the reader, it allows us to showcase a specification that we have implemented and validated using our

[1] https://docs.scala-lang.org/

prototype solver for security ensembles[2]). In order to not require the reader to be familiar with Scala, we limited the use of Scala-specific constructs to the minimal extent possible. Nevertheless, we assume basic familiarity with modern object-oriented language concepts and with basics of functional programming.

The following listing shows an excerpt of the specification of the running example. The full version along with implementation of the analysis and planning part of DSAC can be found at GitHub[2].

```
1  class TestScenario(scenarioParams: TestScenarioSpec) extends Model with ModelGenerator {
2    ...
3    class Door(val id: String, val position: Position) extends Component
4    class Dispenser(val id: String, val position: Position) extends Component
5    class Worker(
6      val id: String, var position: Position,
7      val capabilities: Set[String], var hasHeadGear: Boolean
8    ) extends Component {
9      def isAt(room: Room) = room.positions.contains(position)
10   }
11   class WorkPlace(
12     id: String, positions: List[Position], entryDoor: Door
13   ) extends Room(id, positions, entryDoor) {
14     var factory: Factory = _
15   }
16   class Factory(
17     id: String, positions: List[Position], entryDoor: Door,
18     val dispenser: Dispenser, val workPlaces: List[WorkPlace]
19   ) extends Room(id, positions, entryDoor)
20   class Shift(
21     val id: String, val startTime: LocalDateTime,
22     val endTime: LocalDateTime, val workPlace: WorkPlace,
23     val foreman: Worker, val workers: List[Worker],
24     val standbys: List[Worker], val assignments: Map[Worker, String]
25   ) extends Component
26
27   class FactoryTeam(factory: Factory) extends RootEnsemble {
28     class ShiftTeam(shift: Shift) extends Ensemble {
29       val canceledWorkers = shift.workers.filter(wrk => wrk notified AssignmentCanceledNotification(shift))
30       val calledInStandbys = shift.standbys.filter(wrk => wrk notified CallStandbyNotification(shift))
31       val availableStandbys = shift.standbys diff calledInStandbys
32       val assignedWorkers = (shift.workers union calledInStandbys) diff canceledWorkers
33
34       object AccessToFactory extends Ensemble {
35         situation {
36           (now isAfter (shift.startTime minusMinutes 30)) &&
37             (now isBefore (shift.endTime plusMinutes 30))
38         }
39         allow(shift.foreman, "enter", shift.workPlace.factory)
40         allow(assignedWorkers, "enter", shift.workPlace.factory)
41       }
42
43       object AccessToDispenser extends Ensemble {
44         situation {
45           (now isAfter (shift.startTime minusMinutes 15)) &&
46             (now isBefore shift.endTime)
47         }
48         allow(assignedWorkers, "use", shift.workPlace.factory.dispenser)
49       }
50
51       object AccessToWorkplace extends Ensemble {
52         val workersWithHeadGear = (shift.foreman :: assignedWorkers).filter(wrk => wrk.hasHeadGear)
53         situation {
54           (now isAfter (shift.startTime minusMinutes 30)) &&
55             (now isBefore (shift.endTime plusMinutes 30))
56         }
57         allow(workersWithHeadGear, "enter", shift.workPlace)
58       }
59
60       object NotificationAboutWorkersThatArePotentiallyLate extends Ensemble {
61         val workersThatAreLate = assignedWorkers.filter(wrk => !(wrk isAt shift.workPlace.factory))
62         situation {
63           now isAfter (shift.startTime minusMinutes 20)
64         }
65         workersThatAreLate.foreach(wrk => notify(shift.foreman, WorkerPotentiallyLateNotification(shift,
                 wrk)))
66         allow(shift.foreman, "read.personalData.phoneNo", workersThatAreLate)
67         allow(shift.foreman, "read.distanceToWorkPlace", workersThatAreLate)
68       }
69
70       object CancellationOfWorkersThatAreLate extends Ensemble {
71         val workersThatAreLate = assignedWorkers.filter(wrk => !(wrk isAt shift.workPlace.factory))
72         situation {
73           now isAfter (shift.startTime minusMinutes 15)
74         }
75         notify(workersThatAreLate, AssignmentCanceledNotification(shift))
76       }
77
78       object AssignmentOfStandbys extends Ensemble {
79         class StandbyAssignment(canceledWorker: Worker) extends Ensemble {
80           val standby = oneOf(availableStandbys)
81           constraints {
82             standby.all(_.capabilities contains shift.assignments(canceledWorker))
83           }
84         }
85         val standbyAssignments = rules(canceledWorkersWithoutStandby.map(wrk => new
                 StandbyAssignment(wrk)))
86         val selectedStandbys = unionOf(standbyAssignments.map(_.standby))
87         situation {
88           (now isAfter (shift.startTime minusMinutes 15)) && (now isBefore shift.endTime)
89         }
90         constraints {
91           standbyAssignments.map(_.standby).allDisjoint
92         }
93         notify(selectedStandbys.selectedMembers, StandbyNotification(shift))
94         canceledWorkersWithoutStandby.foreach(wrk => notify(shift.foreman,
                 WorkerReplacedNotification(shift, wrk)))
95       }
96
97       object NoAccessToPersonalDataExceptForLateWorkers extends Ensemble {
98         val workersPotentiallyLate =
99           if ((now isAfter (shift.startTime minusMinutes 20)) && (now isBefore shift.startTime))
100            assignedWorkers.filter(wrk => !(wrk isAt shift.workPlace.factory))
101        else Nil
102        val workers = shift.workers diff workersPotentiallyLate
103        deny(shift.foreman, "read.personalData", workers, PrivacyLevel.ANY)
104        deny(shift.foreman, "read.personalData", workersPotentiallyLate, PrivacyLevel.SENSITIVE)
105      }
106      rules(
107        // Grants
108        AccessToFactory, AccessToDispenser, AccessToWorkplace,
109        NotificationAboutWorkersThatArePotentiallyLate,
110        CancellationOfWorkersThatAreLate, AssignmentOfStandbys,
111        // Assertions
112        NoAccessToPersonalDataExceptForLateWorkers
113      )
114    }
115
116    val shiftTeams = rules(shiftsMap.values.filter(shift => shift.workPlace.factory == factory).map(shift => new
             ShiftTeam(shift)))
117    constraints {
118      shiftTeams.map(shift => shift.AssignmentOfStandbys
119        .selectedStandbys).allDisjoint
120    }
121  }
122  val factoryTeams = factoriesMap.values.map(factory => root(new FactoryTeam(factory)))
123 }
```

[2]https://github.com/d3scomp/tcoof-security-ecsa

Both the components and ensembles are modeled as classes, i.e., they are types and can be instantiated multiple times. In our example, there are six components (lines 3–25), which represent the physical domain objects doors, dispensers, workers, work places and factories. All the components define their observable attributes (termed *knowledge*).

The security rules over these entities are represented as ensembles. A simple example of such a security ensemble is the AccessToFactory (starting at line 34). It states that the ensemble (meaning ensemble type) should be instantiated for every shift during the time interval 30 minutes before the shift till 30 minutes after the shift (line 35). This is the definition of a *situation*, which defines a spatial and temporal condition under which the ensemble is instantiated. The ensemble instance then grants the shift foreman and the workers assigned to the shift an "enter" access grant to the factory hall.

Technically, ensembles are represented as classes (i.e., denoted by the Scala keyword `class` and further instantiated by the keyword `new`) or as singleton objects (i.e., the keyword `object`). The ensembles can be hierarchically nested, i.e., an ensemble can contain other ensembles. This means that components, which are members of an ensemble, have to be also members of the parent ensemble. Thus, a top-level ensemble describes a goal of the system as a whole while the sub-ensembles decompose the system into sub-goals, which are easily manageable. A component can be a member of many ensemble instances at the same time (even of directly unrelated ensembles) reflecting a common requirement that a single component can be simultaneously in different situations.

In the running example, there is a top-level ensemble – FactoryTeam (starting at line 27). Inside it, there is the ShiftTeam ensemble nested (lines 28–114), which represents the security rules for the shift. Inside the ShiftTeam ensemble, there are ensembles defined that deal with particular situations in the shift. The FactoryTeam ensemble instantiates (line 116) all the individual ShiftTeam ensembles and

declares the global constraint for the system, i.e., that standby workers to be assigned to the shifts where required cannot be shared among several shifts (lines 117–120). The FactoryTeam ensemble is then instantiated for every factory in the scenario (line 122). The subensembles are registered to be part of the parent ensemble using the DSL's rules construct (lines 106–113).

The exact semantics is that an ensemble instance is only active if the condition defined in the situation block is true (if no situation condition is present, it is treated as implicitly true). When the ensemble instance is active, it tries to determine which component instances take part in the ensemble instance such that the constraints condition is true. After determining the component instances, it grants them permissions as per the allow statements and issues notifications (showcased later in the text) using the notify statements.

As multiple shifts are running simultaneously within a factory, the ShiftTeam ensemble is parameterized by the Shift component instance (which defines the shift). The ShiftTeam ensemble determines 5 groups of components (lines 29–32) to which the workers are stored during selection by sub-ensembles.

In general, the ShiftTeam's sub-ensembles can be divided into (i) ensembles, which assign permission to individual workers and (ii) which notifies workers about selection for or removal from a shift. The former ones are the ensembles AccessToFactory, AccessToDispenser, AccessToWorkplace, while the latter ones are CancellationOfWorkersThatAreLate and AssignmentOfStandbys, and the NotificationAboutWorkersThatArePotentiallyLate ensemble performs both functions. As already outlined above, all of them has the same structure which is as follows.

First, there is a situation definition. For the AccessToFactory, the current time has to be in the interval of start of the shift minus 30 minutes and end of the shift plus 30 minutes. Similarly, it is for AccessToDispenser but with a different time interval. In the case of the AccessToWorkplace, there is an extra condition that the workers must have a headgear from the dispenser expressed as a selection of the shift workers with the headgear (line 52). All these three ensembles assign (lines 39, 48 and 57) the particular permissions (to enter the factory, use the dispenser, enter the workplace) to the workers selected by the conditions.

The NotificationAboutWorkersThatArePotentiallyLate ensemble detects workers assigned to the shift but not present in the factory (line 61) 20 minutes before start of the shift (line 63). For these workers, the ensemble notifies the particular foreman that they are late and allows the foreman to see the workers' phone numbers (the foreman can call them to "hurry up" – line 65) and their distance from the factory (to see whether there is a chance to come in time yet – line 66). The CancellationOfWorkersThatAreLate ensemble is similar to the previous one, but it detects workers, which are late even 15 minutes before start of the shift (line 73), and notifies them that they are canceled from the shift (line 75).

Generally, the idea about combining allow grants and notifications is that if an entity received an access grant or the access grant has been revoked, the entity should learn about the fact. As such, we follow the design rule that changes in the access grants which cannot be anticipated, e.g., from the time schedule (as it is in the case of the AccessToFactory), need to be accompanied by the notifications. DSAC keeps a history of the notifications and makes it possible to use them in determining components of an ensemble. E.g., on line 29, where the list of workers canceled from the shift because they did not arrive on time is determined by checking whether the worker has been notified about being canceled.

The AssignmentOfStandbys ensemble is responsible for selecting and notifying the standby workers that replace the canceled ones. It has a bit special position with respect to the other ensembles in the specification in the sense that it defines a constraint optimization problem – namely the problem of assigning suitable replacements to workers that have been canceled from the shift. A suitable replacement here means that for each worker canceled from the shift, there is another worker from the list of standbys, such that a standby has capabilities to perform the work of the canceled worker. Furthermore, as the list of standbys is shared for all shifts within the same factory, the assignment has to be done in such a way that the same standby is not assigned to two positions simultaneously.

This constraint problem is defined by the sub-ensemble StandbyAssignment (line 79), which selects a suitable standby worker for a particular canceled worker. The fact that one standby worker is to be assigned is specified using the oneOf statement (line 80). The sub-ensemble is instantiated for each canceled worker (lines 85 and 86). The constraint (line 91) requires that a single standby worker is not used as a replacement for several workers. Within the given time interval (the situation at line 88), the ensemble notifies the selected workers to come (line 93) and notifies the foreman (line 94) of the particular shift about the replacement.

3.3 Security Assertions

Security assertions ensure that dynamic security policies do not violate fundamental security policies. The structure of the assertion is the same as for dynamic security rules but access can only be denied instead of granted. The assertion always overrides the decision of a policy in case of conflicting decisions. The conflict also denotes inconsistency of the specification, which is reported back to the designer of the specification. Because assertions are simpler than policies, they are less error prone. As such, they provide a safety net for users when formulating dynamic policies.

An example of a particular assertion in the running example is represented by the NoAccessToPersonalDataExceptForLateWorkers ensemble. It selects the potentially late workers (i.e., those that do not appear in time interval between 20 minutes till start of the shift and start of the shift – line 98) and other workers (line 102). For the potentially late workers, the ensemble states that the foreman cannot access the sensitive information (line 104) but can access the phone number. For all the other workers, the foreman cannot access any information (line 103).

3.4 Derivation of Privacy Levels

Security assertions (as shown in the previous section) include reasoning about privacy levels. In order to interpret such statements, DSAC requires the knowledge of a privacy level of a given piece of data. Often such data come from computations (such as the distanceToWorkplace, as used in the NotificationAboutWorkersThatArePotentiallyLate ensemble). In those cases, a reasoning mechanism is needed that determines the privacy level of the result given the

privacy levels of the inputs of the computation. For instance in the case of the distanceToWorkplace, the input needed to compute this value is the location, which itself is privacy sensitive. However, by transforming it to a Euclidean distance and minimizing it by value "too far" if more than 1 kilometer, the distanceToWorkplace becomes less privacy sensitive and can be shared with the foreman of the shift.

We shift the derivation of privacy levels to design time to speed up runtime analyses and free ensembles specifications from computation descriptions. The derivation takes place on a data-driven software architecture [23]. In this architecture, we annotate existing artifacts describing the software design on a high level with data flows. In our approach, we annotate design models formulated in the Palladio Component Model (PCM) [20] architectural description language (ADL). A detailed description of derivation is available in our previous work [23]

4 EVALUATION

The evaluation of our approach is twofold. First, we relate the approach to the state-of-the-art systems that are currently in use and second, we analyze scalability of the approach implementation on a realistic deployment.

Relation to state-of-the-art systems: To position our approach in the real industrial settings, we performed discussions with industrial practitioners that are responsible for deploying the smart-lock system (called *IMAporter*) in the production environments of big manufacturers. Here, we present the main points that we derived from the discussions. These points summarize the current state of the art without our approach and the benefits of introducing our approach.

Dynamicity: An essential difference between our approach and legacy system is the enormous dynamicity of changes of the security rules. The legacy systems typically consider only access rights linked to particular job position, which are usually unchanging for a long time period (e.g., until the job position expires). Exceptions are not frequent and manually managed on request by a personal department or in urgent cases (e.g., an access card lost) by a security department. As a partial dynamicity, most of the legacy systems allow for static rules "driven by a calendar" resulting in week-based or month-based rules scheduling. Integration with other third-party system (e.g., HRM or ERP ones) is either a manual or in some cases, semi-automatic that is controlled by a software service (a batch code) but still requiring manual intervention.

Privacy: A high risk of legacy systems is that everyone charged to enter security rules can access all the rules including personal data, suppliers' data, etc. With our approach, the legacy systems are not controlled directly but through our security controller and thus the risk is removed (or at least significantly reduced).

Trust: Arrival of Industry 4.0 completely changes processes within manufacturing companies. Individual workers within a company may not know, how the overall work is organized, what is produced in different workplaces or even allowed to talk with workers from a different team. Our approach allows for expressing such rules (which are with legacy system unreachable).

Effectiveness: Another advantage of Industry 4.0 is effectiveness of manufacturing, i.e., products are manufactures only if they are required and in the exactly required amount. Thus the company has to optimize manufacturing and dynamically increase/decrease amount of workers for particular jobs, even several times during a day. Such a process cannot be done manually. At least, it requires non-trivial integration of legacy systems, where the security rules are implicitly embedded in connectors between the legacy systems. This creates an architecture where security rules are not explicit and very difficult to maintain when the system evolves over time and grows. It is no exceptional that the system has to deal with 10000+ persons and 1000+ smart locks (doors/gates).

Our approach on the other hand deals with the security rules as first-class concepts and makes it easy to modify and evolve them as the whole system grows.

Scalability: To test the scalability of our approach, we have implemented the analysis and planning phase of the Dynamic Security Adaptation Controller (DSAC) and used it to interpret the specification shown in the paper. (Full sources are available at GitHub[3].) We have measured how long it takes to determine access rules for the scenario at different sizes and at two points of time: (1) 17 minutes before the shift and (2) 13 minutes before the shift.

We vary the number of workers in a shift (from 50 to 500) and the percentage of late workers that are canceled from the shift (from 5% to 20%). There are 3 shifts running at the same time. These shifts share a common pool of standby workers. The amount of these standby workers available is determined as $no\_of\_workers\_in\_shift * percentage\_of\_late\_workers * 5$.

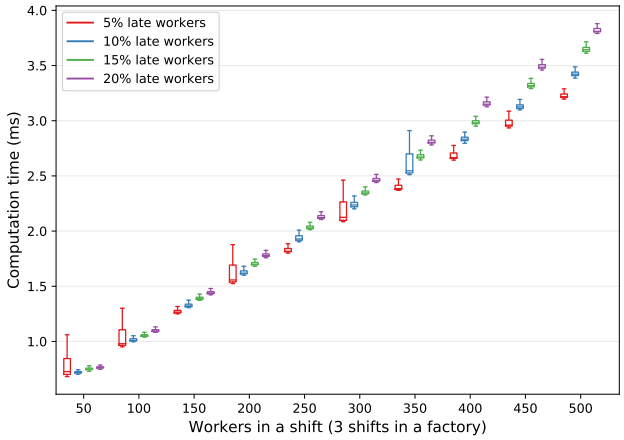
These two time points were chosen because they represent two major cases in the scenario. #1 represents the time when multiple ensembles are instantiated, but they do not require complex constraint optimization. #2 represents the time when complex constraint optimization is needed to determine suitable standby workers to replace the workers that have been canceled from the shift.

DSAC deals with both the time points uniformly – it translates the specification to the constraint solving problem (CSP) and uses a CSP solver to determine the assignment of components to ensemble instances and from them, it determines the access grants. The construction of CSP is however optimized in such a way that in case #1, the search space is significantly constrained and the solving process becomes linear in time complexity. In case #2, the exponential complexity of the solving process is unavoidable, nevertheless the search space grows only with the number of workers that have been canceled (and thus with the number of standby workers shared between the three shifts).

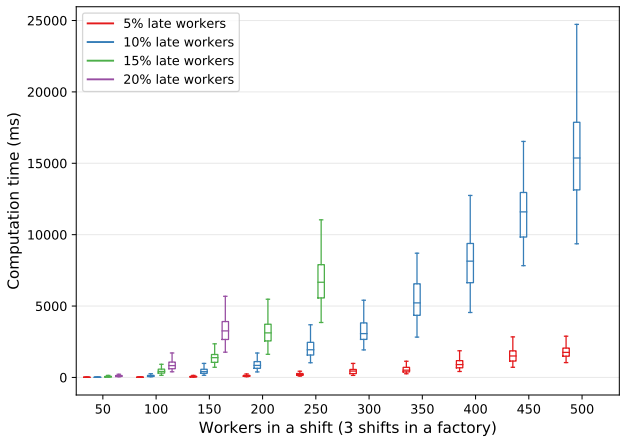
The results were computed on an Intel(R) Xeon(R) CPU E5-2660, running on 2.20GHz. In the case of #1, we performed a warmup of 1000 computations and collected 10000 measurements for each size of the scenario. In the case of #2, we performed a warmup of 10 computations and collected 100 measurements for each scenario size. We excluded computations which exceeded 60 seconds. The results are shown in Fig. 2a and Fig. 2b.

In case #1, the computation time scales linearily with the number of workers in a shift. Given the fact that 3 shifts are evaluated together, we can easily determine the access grants for 1500 workers in approx. 3.5 milliseconds. Case #2 scales exponentially with

[3] https://github.com/d3scomp/tcoof-security-ecsa



(a) 17 minutes to the shift



(b) 13 minutes to the shift

Figure 2: Evaluation results

the percentage of late workers. Nevertheless, even for 10% of late workers, we can assign access grants to 1500 persons in 15 seconds.

The same assignment of access grants happens in our scenario for each factory. As each factory has its own pool of shared standby workers, the access rules can be evaluated independently (as it is so in our test case) and in parallel. As such, the number of factories does not significantly influence the computation time and makes it possible to scale our use-case to arbitrarily large instance, provided that the size of a shift remains below 500 workers and the percentage of workers that do not come to the shift is below 10% (which is a realistic assumption).

5 RELATED WORK

Dealing with context is one of the most important aspects of our approach. The survey in [18] discusses context-based middlewares targeting systems like IoT. Security and privacy is dealt only by three middlewares out of eleven. E.g., in [21], the ontology models are employed but security aspects are handled only for user identification and authorization. From the security point of view, the most advanced is FlexRFID [10], which uses Role-Based Access Control (RBAC) model for security aspects.

Access Control: RBAC [11] is a classical access control approach that employs groups to gather access rights for similar users. Through this abstraction, the rules are comprehensible. However, the relationship from groups to rules is strictly static and does not fit dynamic situations.

Therefore dynamic access system like the Dynamic Role Based Access Control (DRABC) [28] or in [17] are introduced. The first one, DRAB, is an extension for RBAC and is used in the grid environments to assign dynamically resources. There are already dynamic parts included, but no horizontal integration of different companies, like it may happen in Industry 4.0. Another access control system is the Organisational Based Access Control (OrBAC) [15]. Here, contexts [8] describe the access to files. Initially it does not support horizontal integration, however lately there is research into this area [6]. However, it does not support the inclusion of architectural confidentiality analysis like in our approach.

A more generic access control approach than RBAC is Attribute Based Access Control (ABAC) [13] that manages access over attributes, which need to be satisfied for accessing data. In [4], an approach based on ABAC is described, which deals with dynamic situations but only abnormal behavior of users (that might represent an attack on a system) is targeted. Detection of abnormal user behavior is also targeted in [9], where a self-adaptive RBAC (saRBAC) is presented. This saRBAC method models a system via Markov chains and employs probabilistic model checking to discover abnormal behavior of a user (and if discovered, access permissions are modified to mitigate potential attack). Compared to our approach, only single source of dynamism is targeted. Plus, there is no global architecture of a system considered and rules are not hierarchical. Another approach for dynamic adaptation of access permission to mitigate potential attacks of users is presented in [26]. The approach works with design time specifications and updates them based on the runtime behavior of the system. For evaluation of potentially dangerous situations, a statistical approach is used.

An approach partially similar to our one is in [27], which targets policies generation primarily for dynamically established coalitions. Nevertheless, the coalitions are meant only as groups of people with the same goal but by itself are not dynamically described.

An approach very similar to our one is described in [5]. Here, the MAPE-K loop is used for the dynamicity management and the approach allows for optimizations, i.e., it can generate multiple solutions and based on their utility, the optimal one is chosen. Compared to our approach, we allow for unified modeling of both components under direct control and also beyond direct control ones (humans, etc.). Also, as ensembles are hierarchical, the security rules are hierarchical and thus, optimization might not be for a single rule only but it can reflect the overall goal of the system.

Privacy and Confidentiality Analysis The confidentiality analysis checks either whether a system or architecture complies with specific confidentiality rules or if there is a data-leak. One coarse-grained approach for checking security flaws (here for confidentiality) is Threat modelling [25]. However, changes which can happen often in a dynamic environment like in Industry 4.0 are difficult to model, because the security analysis and the system architecture use different models. One way to archive confidentiality is to secure

all transmission by encryption. This is shown in *Hoisl et al.* [12]. However, the approach does not support access control. Another similar approach is UMLsec [14]. It does support access control, however, the rules specification happens on the control flow, which is more complicated than on the dataflow. R-PRIS [22] is an approach, which investigates changes during runtime, that might lead to privacy issues. Therefore a runtime model is used, which is then checked against the privacy rules. In contrast to our approach, R-PRIS [22] only considers the location for privacy checks. There are approaches, which analyses directly the dataflow, like Joana [24] or code verification approaches like KeY [1]. Both support the detection of information leaks, however both approaches are restricted to a certain programming language (here Java). Our confidentiality approach, however, does not depend on one programming language, as it is based on the system architecture rather on the source code level.

6 CONCLUSION

In this paper, we have proposed a self-adaptive architecture with the Dynamic Security Adaptation Controller (DSAC), which adapts static rules for existing legacy systems. Thus, these legacy system can be still used in the dynamically changing context like in the domain of Industry 4.0 or smart manufacturing in general. For configuration of the dynamic security adaptation controller, we have defined an Scala-based internal DSL (available at https://github.com/d3scomp/tcoof-security-ecsa), which can express dynamic security rules that depend on spatial and temporal context. Thanks to the use of ensembles, the approach is able to describe complex dynamic situations with interdependencies across multiple legacy systems. The approach further provides means of runtime validation via assertions that additionally allow incorporating privacy constraints, which we determine based on specifications of data flows in Palladio.

The proposed approach has been discussed with industrial practitioners that are responsible for deploying a smart-lock system in real applications. The evaluation of scalability of the approach suggests that it is applicable for real-life size systems with several thousands of workers.

As an ongoing work, we further improve the performance and scalability of the approach (via optimization of constraints, which are prepared for internally used constraint solver). Also, we continue with cooperation with our industrial partners and are preparing an actual deployment of the proposed system.

ACKNOWLEDGMENTS

This work was supported by the German Federal Ministry of Education and Research (grant numbers 01IS17106A and 01IS17106B), the Technological Agency of the Czech Republic (project no. 2017TF04000064) and by Charles University institutional funding SVV 260451.

REFERENCES

[1] W. Ahrendt, B. Beckert, R. Bubel, R Hahnle, P. H. Schmitt, and M. Ulbrich. 2016. *Deductive Software Verification – The KeY Book.* Springer.

[2] R. Al Ali, T. Bures, P. Hnetynka, F. Krijt, F. Plasil, and J. Vinarek. 2018. Dynamic Security Specification through Autonomic Component Ensemble. In *Proceedings of ISoLA 2018, Limassol, Cyprus.* Springer.

[3] R. Al-Ali, R. Heinrich, P. Hnetynka, A. Juan-Verdejo, S. Seifermann, and M. Walter. 2018. Modeling of dynamic trust contracts for Industry 4.0 systems. In *Companion Proceedings of ECSA 2018, Madrid, Spain.* ACM Press.

[4] L. Argento, A. Margheri, F. Paci, V. Sassone, and N. Zannone. 2018. Towards Adaptive Access Control. In *Data and Applications Security and Privacy XXXII.* Springer.

[5] C. Bailey, D. W. Chadwick, and R. de Lemos. 2014. Self-adaptive federated authorization infrastructures. *J. Comput. System Sci.* 80, 5 (2014), 935–952.

[6] I. Ben Abdelkrim, A. Baina, C. Feltus, J. Aubert, M. Bellafkih, and D. Khadraoui. 2018. Coalition-OrBAC: An Agent-Based Access Control Model for Dynamic Coalitions. In *Trends and Advances in Information Systems and Technologies.* Springer, 1060–1070.

[7] T. Bures, F. Plasil, M. Kit, P. Tuma, and N. Hoch. 2016. Software Abstractions for Component Interaction in the Internet of Things. *Computer* 49, 12 (2016), 50–59.

[8] F. Cuppens and A. Miège. 2003. Modelling contexts in the Or-BAC model. In *Proceedings of ACSAC 2003, Las Vegas, USA.* IEEE, 416–425.

[9] C. E. da Silva, J. D. S. da Silva, C. Paterson, and R. Calinescu. 2017. Self-Adaptive Role-Based Access Control for Business Processes. In *Proceedings of SEAMS 2017, Buenos Aires, Argentina.* 193–203.

[10] M. A. El Khaddar, M. Chraibi, H. Harroud, M. Boulmalf, M. Elkoutbi, and A. Maach. 2015. A policy-based middleware for context-aware pervasive computing. *International Journal of Pervasive Computing and Communications* 11, 1 (2015), 43–68.

[11] D. Ferraiolo, J. Cugini, and D. Kuhn. 1995. Role-based access control (RBAC): Features and motivations. In *Proceedings of ACSAC 1995, New Orleans, USA.* 241–248.

[12] B. Hoisl, S. Sobernig, and M. Strembeck. 2014. Modeling and enforcing secure object flows in process-driven SOAs: an integrated model-driven approach. *Software & Systems Modeling* 13, 2 (2014), 513–548.

[13] V. Hu, D. Kuhn, and D. Ferraiolo. 2015. Attribute-Based Access Control. *Computer* 48, 2 (2015), 85–88.

[14] J. Jürjens. 2002. UMLsec: Extending UML for secure systems development. In *UML'02.* Springer, 412–425.

[15] A.A.E. Kalam, R.E. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurel, and G. Trouessin. 2003. Organization based access control. In *Proceedings POLICY 2003.*

[16] J. Kephart and D. Chess. 2003. The Vision of Autonomic Computing. *Computer* 36, 1 (2003), 41–50.

[17] K. Knorr. 2000. Dynamic access control through Petri net workflows. In *Proceedings of ACSAC 2000, New Orleans, USA.* 159–167.

[18] X. Li, M. Eckert, J.-F. Martínez, and G. Rubio. 2015. Context Aware Middleware Architectures: Survey and Challenges. *Sensors* 15, 8 (2015), 20570–20607.

[19] A. Pfitzmann and M. Hansen. 2010. A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management. https://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf

[20] R. Reussner, S. Becker, J. Happe, R. Heinrich, A. Koziolek, H. Koziolek, M. Kramer, and K. Krogmann. 2016. *Modeling and simulating software architectures: the Palladio approach.* MIT Press.

[21] N. D. Rodriguez. 2011. A Framework for Context-Aware Applications for Smart Spaces. In *Proceedings of SAINT 2011, Munich, Germany.* 218–221.

[22] E. Schmieders, A. Metzger, and K. Pohl. 2015. Runtime Model-Based Privacy Checks of Big Data Cloud Services. In *Proceedings of ICSOC 2015, Goa, India.* 71–86.

[23] S. Seifermann, R. Heinrich, and R. Reussner. 2019. Data-Driven Software Architecture for Analyzing Confidentiality. In *Proceedings of ICSA 2019, Hamburg, Germany.*

[24] G. Snelting, D. Giffhorn, J. Graf, C. Hammer, M. Hecker, M. Mohr, and D. Wasserrab. 2014. Checking probabilistic noninterference using JOANA. *it-Information Technology* 56, 6 (2014), 280–287.

[25] F. Swiderski and W. Snyder. 2004. *Threat Modeling.* Microsoft Press.

[26] T. T. Tun, M. Yang, A. K. Bandara, Y. Yu, A. Nhlabatsi, N. Khan, K. M. Khan, and B. Nuseibeh. 2018. Requirements and specifications for adaptive security: concepts and analysis. In *Proceedings of SEAMS 2018, Gothenburg, Sweden.* 161–171.

[27] D. Verma, S. Calo, S. Chakraborty, E. Bertino, C. Williams, J. Tucker, and B. Rivera. 2017. Generative policy model for autonomic management. In *Proceedings of IEEE SmartWorld 2017, San Francisco, USA.* IEEE.

[28] G. Zhang and M. Parashar. 2003. Dynamic Context-aware Access Control for Grid Applications. In *Proceedings of GRID 2003, Phoenix, USA.* 101–108.