A Recipe-based Algorithm for Access Control in Modular Automation Systems

Björn Leander[‡][†], Aida Čaušević[†] and Hans Hansson[†]
[‡] ABB Industrial Automation, Process Control Platform, [†] Mälardalen University,
Västerås, Sweden
{bjorn.leander, aida.causevic, hans.hansson}@mdh.se

***Abstract*—In the emerging trend towards modular automation, a need for adaptive, strict access control between interacting components has been identified as a key challenge. In this article we discuss the need for such a functionality, and propose a workflow-driven method for automatic access control policies generation within a modular automation system.**

The solution is based on recipes, formulated using Sequential Function Charts (SFC). The generated policies are expressed using Next Generation Access Control (NGAC), an Attribute Based Access Control (ABAC) standard developed by NIST. We provide (1) a definition of required policies for device-to-device interactions within a modular automation system, (2) an algorithm for automatic generation of access policies, (3) a formal proof of the correctness of this algorithm, and (4) an illustration of its use.

I. Introduction

Modular Automation (MA) [1] is an emerging technology within the process automation industry that promises to enable profitable operations, reduced time-to-market and shortened product life cycles [2]. Even though the technology is in its infancy, a number of pilot projects have been already carried out[1], along with a number of control system vendor implementations specifically targeting MA[2]. Within the chemical, pharmaceutical, and energy sectors there is an estimated 2030 market potential of approximately 12 billion euros for modular process automation equipment [3].

The technology suggested to be used in MA exhibits similar characteristics as solutions provided in the Industry 4.0 paradigm, namely interconnected service oriented devices, utilizing different connectivity capabilities, including wireless communication [4], [5]. The different entities within the systems are assumed to be highly heterogeneous and dynamic, and the architecture is expected to be modular, with different modules able to autonomously fulfill specific tasks, requiring only high level engineering to combine and re-combine modules to execute the complete production scheme. This allows a high level of customization and re-use of modules provided and possibly maintained by specialized vendors.

In these dynamic and flexible systems where communication paths are not pre-defined, and production schemes are ever-changing, it becomes difficult to detect malicious behaviour, at least between devices seen as legitimate. At the same time, the attack surface and complexity of the system is increasing, raising the risk of a legitimate device being compromised.

A compromised device, controlled by a malicious actor, may cause a significant economic damage for the factory owner, as well physical damage on e.g., humans, machinery or the environment. The impact may be direct, e.g., the opening of a valve may overfill a tank or turning on heating in an empty reactor may cause a fire. Impact could also be indirect, e.g., changing ratios of materials used to produce a medicine may render it harmful. The direct causes are usually mitigated by implementations of secondary safety measures, while indirect causes may be more difficult to detect and mitigate.

During the last years, there has been a steady trend of increasing amounts of cyber-attacks on industrial control systems [6]. When analyzing who is attacking and why attacks occur against different targets, there is a number of standard categories [7], [8] used: Hobby hacker, Insider, Cyber-criminal, Hacktivist, Terrorist and Nation state. For attacks against industrial control systems, the two main categories with knowledge and capacity to perform targeted attacks are the Insider and the Nation State. However, any of the other categories can use an Insider to gain initial foothold, e.g., by social engineering, bribery or extortion. An Insider can hold deep knowledge of the system, credentials, as well as physical access to the system.

Applying strict and fine-grained access control according to the principle of *least-privilege* [9] is one of the major mechanisms able to protect against the threat from Insider attacks, by allowing access to operations or data only to privileged entities. It also increases the visibility of the malicious actor, as denied access control requests are typically monitored e.g., using a Security Information and Event Monitoring (SIEM) system [10]. However, using a strict access control at the lower layers in an automation system is quite uncommon. Historically, industrial automation systems have been built up using proprietary communication protocols, hard-wiring between controllers and IO, and the notion of an air-gapped network, i.e., no communication between the control network and the outside world. These assumptions on the technical solutions have meant that the pragmatic solution is to allow any legitimate device on the network to perform any action. With

[1]new.abb.com/life-sciences/references/modular-automation-solution-for-life-science-company-bayer-ag

[2]new.abb.com/news/detail/31671/plant-orchestration-and-pilot-application, www.festo.com/us/en/e/automation/industries/water-technology/modular-automation-id_4801/

the advent of MA and Industry 4.0 none of these assumptions hold anymore, and therefore the practice of including a strict access control between devices in automation systems is of increasing importance.

Two of the main hurdles to introducing access control for machine-to-machine interactions in a MA system are the difficulty to express policy rules matching the dynamic behavior of the system, and the management effort required to uphold the policies in a timely and efficient manner. In relation to that, the following research questions are stated:

RQ1 How can access control policies be expressed to fulfill the principle of least-privilege for device to device interactions within a MA system?

RQ2 How can the effort related to access control policy management be minimized in a MA scenario?

In this paper we propose an approach providing answers to both these questions, by introducing a model-based method for generating access control policies from formalized recipe descriptions. We present a definition on required access control rules for recipe orchestration, and provide a formal proof showing that the algorithm produces rules in accordance with that definition. Moreover, we apply the algorithm on a simple example.

The remainder of this paper is structured as follows. Definitions are given in Section II, including a formal definition of the requirements on privileges required during the recipe orchestration. In Section III, an algorithm for access policy generation is described followed by an illustrative example in Section IV. Furthermore, we discuss the proposed solution and results in Section V, compare it to relevant related work in Section VI, before making a few concluding remarks in Section VII.

II. Preliminaries

A. A Recipe definition using an SFC

The execution of a workflow in MA is described by a *recipe* with different processing steps, each containing a set of operations that one or more modules shall perform. A common format used to describe a recipe is through a Sequential Function Chart (SFC), which is currently used e.g., for batch processing in traditional process automation. Execution of a recipe is driven by a central unit, following the concept of orchestration of autonomous services [11]. SFC is a high-level Programmable Logic Controllers (PLC) language, defined within the IEC 61131 standard [12].

Let us consider an example of a simple MA setup described within the DIMA project [2]. In order to produce a specific product, different modules are combined and a process recipe is being formulated as follows:

1) A reactor is filled with three different materials in a specific ratio.
2) The reactor module mixes and heats the mixture, and maintains a fixed temperature for a specified amount of time.
3) The resulting mixture is distilled by a distilling module.

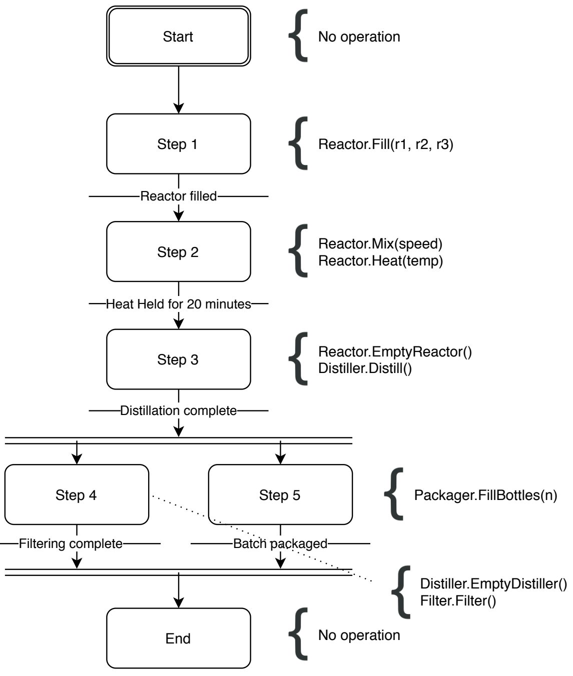

Fig. 1: An example of a recipe expressed as an SFC

4) The distillate is further purified by a filtration module.
5) The product is packed into a container by a filling module.

The example can be formulated as a recipe using an SFC, as illustrated in Fig. 1. We assume that filtration and packaging can be executed in parallel, i.e., the packaging can start as soon as there is a sufficient amount of the final product available.

An SFC consists of steps and transitions. Each step in the recipe describes the operations relevant to perform in that step. Moreover, each step contains zero or more outward directed transitions (arcs) describing the conditions for continuing to the next step(s), i.e., a transition point to one or more subsequent steps. In the case of more than one step, the following steps are executed in parallel as soon as the condition annotated on the transition enabling that step is fulfilled. To join a parallel execution, two (or more) edges point to the same step. In such join-cases, conditions for all edges pointing at the same step must be fulfilled for it to be triggered. Moreover an SFC may contain loops (not included in Fig.1).

In general, operations described for a step contain code describing operations detailing the control logic of a step. The standard allows nested SFCs, so that a step can be described by another SFC. However, in MA recipe declaration, this description will most likely be vastly simplified, as the modules are expected to perform the low level control logic by themselves, based on high-level instructions executed by the orchestrator. For our description of an SFC formulating a modular automation recipe, the important aspect is that one step contains zero or more module-related operations.

A recipe R is a pair (id, s_0), where id is a unique recipe

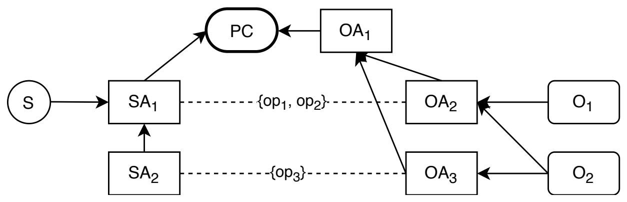

Fig. 2: An example of an NGAC graph

identification, and s_0 is the initial step of an SFC $F = (S, s_0)$. $F.S$ is the set of all steps contained by the SFC F. A *step* is defined as a triplet $step = (id, OP, T)$ where OP is a set of operations, T is set of transitions and id is a unique identifier for the step. Moreover, $op \in OP$ is described by a pair $op = (id, target)$, where id is a unique operation identification, and $target$ identifies the target module. A transition $t \in T$ is described by a pair $t = (c, steps)$ where c is a Boolean condition that must hold for the transition to be fired, and $steps$ is a set of one or more (parallel) steps to be activated by the transition.

For the approach of a policy generation algorithm presented in this article, the condition of a transition is not used. However, in future versions we envision using the conditions to more closely make the policy rules match the workflow of the recipe.

B. An NGAC graph definition

Access control is the practice of granting or denying a legitimate subject privileges to a requested resource [13]. An Access Control Model is a model for formally describing access control policies. Next Generation Access Control (NGAC) [14] is a NIST standardized access control model, based on the paradigm of Attribute Based Access Control (ABAC). In ABAC, attributes of the subject, resource and the environment are used to express the policies, as opposed to traditional models that are usually based mainly on the identity or role of the subject [15].

In the following, we provide a simplified description of NGAC, based on the work by Ferraiolo et al. [16], focusing to describe only those parts of the mechanism important for the purpose of this article. We exclude the details regarding prohibitions, while obligations will be briefly discussed later on in this article.

In NGAC, attribute assignments and privilege associations are described using a graph G. Subjects s, objects o, policy-classes pc and attributes a are modeled as vertices in the graph. Assignments of attributes to subjects, objects or policy-classes are modeled as directed edges ending at the assignment target. Assignments are also allowed between attributes, so that hierarchies of attributes can be formed. The assignment operation should be interpreted as containment, e.g, $o \rightarrow a$ means an object o is contained by an attribute a. Privileges to execute operations are modeled as associations between subject and object attributes (sa and oa, respectively) and described as a triplet: (sa, ops, oa) and visualized in the graph as an un-directional dashed line between the subject and object attributes, where ops is a set of operations.

Let us consider an example depicted in Fig. 2. It describes an NGAC-graph, where a subject S is assigned to attribute SA_1; objects O_1, O_2 are assigned to attribute OA_2. Between OA_2 and SA_1 there exists a privilege association for operations $\{op1, op2\}$. Furthermore, an object O_2 is assigned to attribute OA_3, and there exists a subject attribute SA_2 associated with OA_3 for operation $\{op_3\}$. Object attributes OA_2 and OA_3 are assigned to OA_1. OA_1 and SA_1 are both contained in policy class PC. Using the privilege association between SA_1 and OA_2 the operations op_1 and op_2 on objects O_1 and O_2 are granted to the subject S. However, an operation op_3 on O_2 is not granted, since no association can be made between the subject S and the object O_2 for the given operation. If, on the other hand, S would have been contained in attribute SA_2, then operations op_1, op_2, op_3 would have been allowed on O_2.

For operations on an NGAC graph, we will use a number of definitions from [14]. A short summary of these definitions and their meaning is provided in Table I. In NGAC, the term *user* denotes the same entity as we denote *subject*, therefore e.g., ua represents "user attribute", while we write "subject attribute" to maintain a consistent nomenclature within the article.

For operation op on object o executed by subject s, we say that (s, op, o) is a privilege using the following definition:

$$\textsc{Privilege}(s, op, o) = \begin{cases} true \text{ if } \begin{cases} \forall pc \in \text{PC} : \textsc{Assign}^+(o, pc) \wedge \\ (\exists (sa, ops, oa) \in \\ \textsc{Associations} : op \in ops) \wedge \\ \textsc{Assign}^+(s, sa) \ \wedge \ \textsc{Assign}^+(o, oa) \ \wedge \\ \textsc{Assign}^+(s, pc) \ \wedge \ \textsc{Assign}^+(oa, pc)) \end{cases} \\ false \text{ otherwise} \end{cases} \tag{1}$$

Intuitively, this means that for for the privilege of s executing operation op on target object o to be granted for a policy-class pc containing o, there must exist an association between a subject attribute sa and an object attribute oa containing operation op, where s is assigned to attribute sa and o is assigned to attribute oa, and both s and oa are assigned to the policy class pc.

C. Definition of privileges required by a recipe orchestrator

Using the definitions introduced in Sections II-A and II-B, we are able to define which access control privileges are required by a recipe orchestrator when a recipe formalized as an SFC is executed, following the principle of least privilege.

An orchestrator $subj$ is allowed to execute a step $step \in F.S$ for an SFC $F = (S, s_0)$ where the access control policies are described by an NGAC graph by the following definition:

$$\textsc{Priv}_{\textsc{step}}(subj, step) =$$
$$\begin{cases} true \text{ if } \forall op \in step.OP : \textsc{Privilege}(subj, op.id, op.target) \\ false \text{ otherwise} \end{cases}$$

Name	**Description**
PC	The set of all policy-classes in the graph.
$\textsc{Associations}$	The set of all associations in the graph, one association being defined by a triple (sa, ops, oa).
$\textsc{Assign}^+(x, y)$	There exists a series of assignments from x to y. Note that in the NIST standard [14], the notation used is $(x, y) \in \textsc{Assign}^+$.
$\textsc{CreateOAinPC}(oa, pc)$	Create an object attribute with id oa and assign it to policy class pc. A call of this function implies that $\textsc{Assign}^+(oa, pc)$ is fulfilled.
$\textsc{CreateUAinPC}(ua, pc)$	Create an subject attribute with id ua and assign it to policy class pc. A call of this function implies that $\textsc{Assign}^+(ua, pc)$ is fulfilled.
$\textsc{CreateOAinOA}(oa_1, oa_2)$	Create an object attribute with id oa_1 and assign it to object attribute oa_2. A call of this function implies that $\textsc{Assign}^+(oa_1, oa_2)$ is fulfilled.
$\textsc{CreateUAinUA}(ua_1, ua_2)$	Create a subject attribute with id ua_1 and assign it to subject attribute ua_2. A call of this function implies that $\textsc{Assign}^+(ua_1, ua_2)$ is fulfilled.
$\textsc{CreateAssoc}(sa, ops, oa)$	Creates an association between subject attribute sa and object attribute oa with operations ops, i.e. $\textsc{Associations} = \textsc{Associations} \cup \{(sa, ops, oa)\}$. We assume that consecutive calls using the same combination of sa and oa will update the set of operations for the association using a set-union function.

TABLE I: NGAC-operations

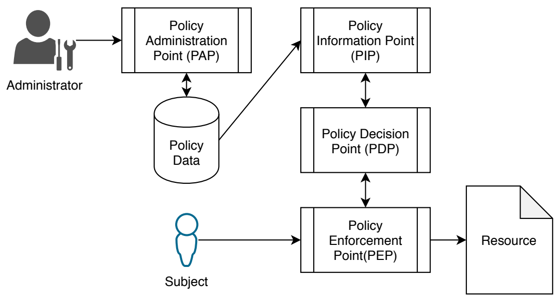
Administrator
Policy Administration Point (PAP)
Policy Information Point (PIP)
Policy Data
Policy Decision Point (PDP)
Policy Enforcement Point(PEP)
Resource
Subject

Fig. 3: Access Control Architecture

Subsequently, for the orchestrator $subj$ to execute a recipe $R = (id, s_0)$, where SFC $F = (S, s_o)$, $\textsc{Priv}_{\textsc{step}}(subj, step)$ must be fulfilled for all steps in the SFC, i.e.,

$$\textsc{Priv}_{\textsc{recipe}}(subj, R) = \begin{cases} true \text{ if } \forall\ step \in F.S : \textsc{Priv}_{\textsc{step}}(subj, step) \\ false \text{ otherwise} \end{cases} \quad (2)$$

D. Access Control Architecture prerequisites

Policy enforcement is an important characteristic of an access control mechanism. Fig. 3 depicts a typical architecture that describes the entities involved in an access control enforcement architecture [16], [17], [18]. For the mechanism to work, there can be no other way for a subject to access a resource than through a Policy Enforcement Point (PEP). Therefore PEP must be kept close to the resource, typically running on the same device as the resource. After a privilege request is initiated, a PEP must ask a Policy Decision Point (PDP) for a decision defining whether the request shall be granted or not. To answer the request, a PDP must be able to query policy data through a Policy Information Point (PIP). Policy data is administered through the Policy Administration Point (PAP). The actual placement and implementation of these policy interaction points will be of great importance, as it influences how well the access control mechanism functions and scales. In this article, we assume that an appropriate such architecture is in place. Another prerequisite for a secure access control is that identities of all involved entities can be trusted.

Secure authentication of entities can be achieved using a number of methods, including public key certificates. In this article we will assume that authenticity of identities are proven using some trusted mechanism.

III. Generating Access Control rules in NGAC using an SFC Recipe

As a recipe is activated and assigned to modules and an orchestrator, the access control policies prescribing which operations that a specific orchestrator is able to perform within the system shall also be updated. Similarly, deactivation of a recipe shall remove privileges exclusively granted through that recipe.

In this article we are focusing on the interactions between orchestrators and modules in an MA system. To define these interactions, we propose to use the recipe as a basis to formalize and automate access control rule generation. Based on these rules, it is possible to grant only those privileges prescribed by the processing needs. We propose to use the recipe as a model, further used to derive detailed policy rules expressed as ABAC policies according to the NGAC specification.

In this section we introduce an algorithm to enable automatic generation of access control policies. The algorithm takes a formalized SFC model as input and as result updates an NGAC graph with access privilege information.

For each step in an SFC, zero or more operations are allowed to be executed by the orchestrator on the target modules. Therefore it is logical to, in terms of NGAC attributes, think of the SFC steps as subject attributes. Based on them, privileges will be granted to the orchestrator by associations to a respective target module specific attribute.

Algorithm 1 POLICYGENERATION(R, pc)

```
function GENERATESTEPPOLICIES($step, R_{id}$)
    if $step.OP \neq \{\}$ then
        $mod := \text{MOD\_ID}(R_{id})$
        $orch := \text{ORCH\_ID}(R_{id})$
        CREATEUAINUA($orch, step.id$)
        for all $op \in step.OP$ do
            $targ := \text{TARGET\_ID}(op.target, R_{id})$
            CREATEOAINOA($targ, mod$)
            CREATEASSOC($step.id, op.id, targ$)
        end for
    end if
end function

function VISITSTEP($step, R_{id}$)
    if $\neg$VISITED($step$) then
        VISIT($step$)
        GENERATESTEPPOLICIES($step, R_{id}$)
        for all $t \in step.T$ do
            for all $sub\_step \in t.steps$ do
                VISITSTEP($sub\_step, R_{id}$)
            end for
        end for
    end if
end function

begin algorithm
    $orch := \text{ORCH\_ID}(R.id)$
    $mod := \text{MOD\_ID}(R.id)$
    CREATEOAINPC($mod, pc$)
    CREATEUAINPC($orch, pc$)
    VISITSTEP($R.s_0, R.id$)
end algorithm
```

In the algorithm, we use the functions described for an NGAC graph in Table I, together with the following functions:

- MOD_ID(R_{id}) - returns a unique attribute id for all modules being orchestrated by the recipe, based on the recipe id.
- ORCH_ID(R_{id}) - returns a unique attribute id for the orchestrator of the recipe based on the recipe id.
- TARGET_ID$(target, R_{id})$ - returns a unique attribute id for a specific module, based on the recipe id and the target id as used in the recipe.

Algorithm 1, POLICYGENERATION, is called using a recipe R, and a policy class pc as arguments. The policy class pc must be predefined, and could e.g., be used to keep together all the policies related to control of modules. Unique attributes for module mod and orchestrator $orch$ are generated, based on the recipe identification $R.id$. The attribute mod will be common for all modules related to the recipe $R.id$, and the attribute $orch$ will be used for the orchestrator of the recipe $R.id$. As can be seen, ASSIGN$^+(mod, pc)$ and ASSIGN$^+(orch, pc)$ are the major result of the algorithm. Finally, function VISITSTEP is called using the initial step of the recipe, $R.s_0$, as input.

In VISITSTEP, the function VISITED(S) and method VISIT(S) are used to be able to determine if policies are already generated for the specific step. If the step has not been previously visited, function GENERATESTEPPOLICIES is called. For all the transitions $t \in step.T$, all $sub\_step \in t.steps$ are used as arguments for calls to VISITSTEP, to ensure policy generation for steps following a transition from the input parameter step.

In GENERATESTEPPOLICIES, if there are any operations related to the step, then (1) a subject attribute representing the step in the SFC is created based on the unique identification of the step, (2) $orch$ is assigned to it, i.e., $orch \rightarrow step.id$, (3) for all operations $(target, op)$ in the step, an attribute $targ$ is created for the target module unique within the recipe, such that $targ \rightarrow mod$, and (4) an association is created between attributes $step.id$ and $targ$ such that $\exists(step.id, ops, targ) \in$ ASSOCIATIONS : $op \in ops$.

A. A proof of algorithm correctness

In the following we provide a proof that by induction shows that the algorithm will create access control policies fulfilling the relationship as defined in Section II-C, i.e., that PRIV$_{\text{RECIPE}}(subj, R)$ is fulfilled. The proof is divided into three lemmas and a proof of the main theorem based on the lemmas.

In the proof we rely on the transitive property of the ASSIGN$^+$ relation, i.e.,:

$$\text{ASSIGN}^+(a, b) \wedge \text{ASSIGN}^+(b, c) \implies \text{ASSIGN}^+(a, c) \quad (3)$$

Theorem 1. *Algorithm 1 will create policies fulfilling definition* PRIV$_{\text{RECIPE}}(subj, r)$ *for a recipe* $R = (id, s_0)$*, an orchestrator subj, and a set of target modules* T_m*, using an NGAC graph containing the policy class pc, under the following assumptions:*

$$\text{ASSIGN}^+(subj, \text{ORCH\_ID}(R.id)) \quad (4)$$

$$\forall t \in T_m : \text{ASSIGN}^+(t, \text{TARGET\_ID}(t, R.id)) \quad (5)$$

$$\exists! pc \in \text{PC} : \forall t \in T_m : \text{ASSIGN}^+(t, pc) \quad (6)$$

Intuitively, the theorem states that Algorithm 1 provides access control policies fulfilling the principle of least privilege with regards to recipe orchestration.

The first assumption described (Eq. 4) states that the orchestrator $subj$ will need to be assigned to attribute ORCH_ID$(R.id)$. The second assumption (Eq. 5) states that the modules being used in recipe orchestration will have to be assigned to a unique attribute for the combination of the recipe and target id. Both of these assumptions should be fulfilled during recipe activation, as part of the operation engineering phase. Therefore these assumptions are necessary and valid.

The third assumption (Eq.6) states that there is exactly one policy class for privileges related to the target modules being orchestrated. As the purpose of a policy-class is to organize and distinguish between distinct types of policies [14], it is reasonable to make this assumption. This is indicated by the first part of the privilege definition (Eq. 1): $\forall pc \in \text{PC}$: ASSIGN$^+(o, pc)$. It follows that for a multi-policy scenario where one object is contained by more than one policy-class, for any privilege to be granted in relation to that object, associations must be present in all of the containing policy-classes.

Lemma 1 (Policy generation for a single SFC step)**.** *Function* GENERATESTEPPOLICIES *(Alg. 1, Ln. 1) generates access control policies fulfilling* $\text{PRIV}_{\text{STEP}}(subj, step)$ *for any step in an SFC used as parameter, given that:*

$$\text{ASSIGN}^+(subj, pc) \tag{7}$$

$$\text{ASSIGN}^+(\text{MOD\_ID}(R_{id}), pc) \tag{8}$$

Proof. By induction on $op \in step.OP$.
Base case: For a SFC step $step$ with $step.OP = \{\}$, a call to GENERATESTEPPOLICIES will fulfill $\text{PRIV}_{\text{STEP}}(subj, step)$.

In this case the proof is trivial. No policy elements will be created. Hence, there are no operation in $step.OP$, and $\text{PRIV}_{\text{STEP}}(subj, step)$ is vacuously true.
Induction hypothesis: Assume that for a step $step$, GENERATESTEPPOLICIES will grant privileges fulfilling $\text{PRIV}_{\text{STEP}}(subj, step)$.
Induction: Let $step'$ contain operations $step'.OP = step.OP \cup \{(op', t')\}$.

As $step'.OP \neq \{\}$, attribute assignments and associations will be provided according to the following:

1) From Alg. 1, Ln. 5

$$\text{CREATEUAINUA}(orch, step'.id) \implies \text{ASSIGN}^+(orch, step'.id) \implies \text{ASSIGN}^+(subj, step'.id) \tag{9}$$

since $\text{ASSIGN}^+(subj, \text{ORCH\_ID}(R.id))$ is in our initial assumption (Eq. 4), and $orch \equiv \text{ORCH\_ID}(R.id)$.

2) For the additional operation (op', t') (Alg. 1, Ln. 8):

$$\text{CREATEOAINOA}(targ, mod) \implies \text{ASSIGN}^+(targ, mod) \implies \text{ASSIGN}^+(targ, pc) \tag{10}$$

due to $\text{ASSIGN}^+(\text{MOD\_ID}(R_{id}), pc)$ in the assumptions of this lemma, and $mod \equiv \text{MOD\_ID}(R_{id})$. Furthermore, $targ \equiv \text{TARGET\_ID}(t', R_{id})$ according to the initial assumptions (Eq. 5), and therefore $\text{ASSIGN}^+(t', targ)$ is fulfilled.

3) From Alg. 1, Ln. 9:

$$\text{CREATEASSOC}(step'.id, op', targ) \equiv \exists(step'.id, ops, targ) \in \text{ASSOCIATIONS} : op' \in ops \tag{11}$$

It then follows by stated assumptions (Eq. 6, 7):

$$\exists(step'.id, ops, targ) \in \text{ASSOCIATIONS} : op' \in ops \wedge \text{ASSIGN}^+(subj, step'.id) \wedge \text{ASSIGN}^+(t', targ) \wedge \text{ASSIGN}^+(subj, pc) \wedge \text{ASSIGN}^+(targ, pc) \tag{12}$$

As we assume that there exists exactly one pc for operations related to target modules (Eq. 6), $\text{PRIVILEGE}(subj, op', t')$ is true according to Eq. 1. Since $\text{PRIV}_{\text{STEP}}(subj, step)$ is satisfied according to the inductive assumption and $step'.OP = step.OP \cup \{(op', t')\}$, we have shown that also $\text{PRIV}_{\text{STEP}}(subj, step')$ is true.

Base case + induction shows that for any SFC step $step$ where GENERATESTEPPOLICIES$(step, ...)$ is called, $\text{PRIV}_{\text{STEP}}(subj, step)$ will be fulfilled, under given assumptions. ■

Lemma 2 (Policy generation for Visited steps)**.** *A step* p *visited by procedure* VISITSTEP$(p, ...)$*, will imply that* $\text{PRIV}_{\text{STEP}}(subj, p)$ *is fulfilled.*

Proof. $\text{VISIT}(p)$ will set the $\text{VISITED}(p)$. Furthermore, from Alg. 1, Ln. 17

$$\text{GENERATESTEPPOLICIES}(p, R_{id}) \implies \text{PRIV}_{\text{STEP}}(subj, p) \tag{13}$$

according to Lemma 1. ■

Lemma 3 (Policy generation for an SFC)**.** *For a recipe* $R = (R_{id}, s_0)$ *where SFC* $F = (S, s_0)$*, a call to function* VISITSTEP$(s_0, ...)$ *will generate policies such that* $\text{PRIV}_{\text{RECIPE}}(subj, R)$ *is fulfilled.*

Proof. By induction on $step \in F.S$
Base case: For a recipe $R = (R_{id}, s_0)$ where SFC $F = (S, s_0)$ with $F.S = \{s_0\}$, a call to VISITSTEP$(s_0, ...)$ will generate policies such that $\text{PRIV}_{\text{RECIPE}}(subj, R)$ is fulfilled. By Lemma 2 it follows that $\text{PRIV}_{\text{STEP}}(subj, s_0)$ is fulfilled. Given that $F.S = \{s_0\}$ Eq. 2 is also fulfilled, which proves the base case.
Induction hypothesis: Assume that for a Recipe $R = (R_{id}, s_0)$ where SFC $F = (S, s_0)$ and contains step $step_i \in S$, procedure VISITSTEP using s_0 as parameter will produce policies fulfilling the definition in Eq. 2.
Induction: Let recipe $R' = (R_{id}, s_0)$ where SFC F' being F extended with one additional step $step_{i+1} \neq s_0$ such that $F'.S = F.S \cup \{step_{i+1}\} \wedge \exists trans \in step_i.T : step_{i+1} \in trans.steps$.

For $step_i$, VISITSTEP(sub_step, G, R_{id}) implies that VISITSTEP will be called for $step_{i+1}$, since $\exists\ trans \in step_i.T : step_{i+1} \in trans.steps$. According to Lemma 2, this implies that $\text{PRIV}_{\text{STEP}}(subj, step_{i+1})$ is true. For F, we have that $\forall step \in F.S : \text{PRIV}_{\text{STEP}}(subj, step)$. As $F'.S = F.S \cup \{step_{i+1}\}$, the following holds:

$$\text{PRIV}_{\text{STEP}}(subj, step_{i+1}) \wedge \forall step \in F.S : \text{PRIV}_{\text{STEP}}(subj, step) \implies \forall step \in F'.S : \text{PRIV}_{\text{STEP}}(subj, step) \tag{14}$$

which is according to $Eq.$ 2 is equivalent to $\text{PRIV}_{\text{RECIPE}}(subj, R')$.

Base case + induction proves that for any recipe $R = (R_{id}, s_0)$ where SFC $F = (S, s_0)$, a call to function VIS-

ITSTEP will fulfill the definition in Eq. 2, if the assumptions in Lemma 1 holds. ■

Recalling Theorem 1, we will now show that for any recipe $R = (R_{id}, s_0)$, Algorithm 1 will generate policy elements fulfilling $\textsc{Priv}_{\textsc{recipe}}(subj, R)$

Proof of Theorem 1.

$$\textsc{CreateOAinPC}(mod, pc) \implies \textsc{Assign}^+(\textsc{mod\_id}(R_{id}), pc) \quad (15)$$

and, from the initial assumption (Eq.4),

$$\textsc{CreateUAinPC}(orch, pc) \implies \textsc{Assign}^+(orch, pc), \textsc{Assign}^+(subj, \textsc{orch\_id}(R.id)) \wedge \textsc{Assign}^+(orch, pc) \implies \textsc{Assign}^+(subj, pc) \quad (16)$$

Thereby, both stated assumptions in Lemma 1 are fulfilled. Furthermore, $\textsc{VisitStep}(R.s_0, R.id)$. is called. Together, this imply that Eq. 2 is fulfilled according to Lemma 3.

Consequently, we have proved that the initial theorem is correct. The proposed algorithm will generate access control policies fulfilling the definition in Eq. 2, required for an orchestrator $subj$ to execute a recipe $R = (R_{id}, s_0)$, i.e., $\textsc{PolicyGeneration}(R, pc) \implies \textsc{Priv}_{\textsc{recipe}}(subj, R)$. ■

IV. Proposed algorithm exemplified

Let us consider using the proposed algorithm on the example of the recipe described by the SFC in Fig. 1. For readability reasons, in this example we use a string representations for attribute and entity IDs. In reality these will most likely be numeric IDs. As an input to the algorithm we use the SFC and a policy class, which is assumed to already be existing in the NGAC-graph. We annotate them as "Module Control Policies" as ID for the policy class.

In the main part of the algorithm, firstly two unique attributes will be generated, one for the orchestrator ("Recipe ID Orchestrator") and one for the modules ("Recipe ID Module"), see lines 26-30 in ALGORITHM 1. After that the function VISITSTEP is called using the step Start of the recipe as input, along with the id of the recipe.

In function VISITSTEP the step Start is marked as visited (lines 15-16 in ALGORITHM 1) and then the GENERATESTEPPOLICIES function is called, using the step as input parameter. As the starting step contains no operations, nothing happens in this first call to GENERATESTEPPOLICIES (condition on line 2 in ALGORITHM 1 is false). On line 20 "Step 1" is used as an input to a recursive call to VISITSTEP, which leads to a call to GENERATESTEPPOLICIES using "Step 1" as an input.

Since there is an operation related to Step 1, a subject attribute related to the step is created ("Recipe ID Step 1") on line 5 such that "Recipe ID Orchestrator"→"Recipe ID Step 1", followed by lines 7-8 where an object attribute is created for the module (in this case "Recipe ID Reactor"), such that "Recipe ID Reactor"→"Recipe ID Module". On line 9, an association between the attribute "Recipe ID Step 1" and "Recipe ID Reactor" is created, containing the operation "Fill". In this way all the steps of the SFC are iterated, thus creating the NGAC sub-graph related to this specific recipe.

The illustration in Fig. 4 depicts the NGAC sub-graph related to this policy, after recipe activation. The gray area in the graph represents the results of executing our algorithm. The module and orchestrator assignments to the respective attributes are part of a recipe activation, and the policy class is assumed to be existing prior to the execution of the algorithm. We omit the details regarding the assignment of modules and orchestrator to the respective attribute in the proposed algorithm. The physicals modules that are a part of the manufacturing scheme must be selected by the operational engineer upon recipe activation. We assume that there will be a simple way to match the physical module ID with the representative ID used in the recipe.

As can be seen, each of the steps from the original SFC are represented by subject attributes in the NGAC-graph, and all the individual modules utilized in the SFC are given as object attributes. The privileges are described as associations between the step-, and module-attributes, e.g., for Step 3, there is one association to the Reactor-attribute, granting an operation "EmptyReactor", and one association to the Distiller-attribute, granting an operation "Distill".

V. Discussion

The suggested approach of restricting access control policies based on the recipe description would effectively prevent any entity to perform actions on modules outside of active recipes, mitigating the effects of a compromised or faulty device with regards to execution of operations. Depending on the implementation of the authorization enforcement layer, it could also improve the resilience against a Denial of Service (DoS) attack against a module or the orchestrator, as processing of unauthorized requests could be minimized, i.e., processing of malicious requests can be skipped if it can easily be determined that they are unauthorized. Furthermore, failed authorization is usually captured by audit logging and can be visualized in a SIEM system, increasing the visibility of the attacker. The approach would also provide an access control model supporting the concept of module reuse, which is one of the main objectives of MA. By automatically generating the access control policies from already existing engineering data, the management effort related to sustaining the rules in accordance with the least privilege principle is minimized.

A difference between NGAC and other ABAC implementations, e.g., eXtensible Access Control Markup Language (XACML) [17], lies in the fact that an attribute does not represent a named property that can hold different typed values. This results in creation of several "synthetic" attributes, i.e., attributes that are not naturally associated with a subject or object. For example, as a result of the policy generation algorithm, the need for unique attributes for each combination of recipe and module yields a large number of attributes that can only be used in the context of execution of a specific recipe. A more natural concept would be to have a general

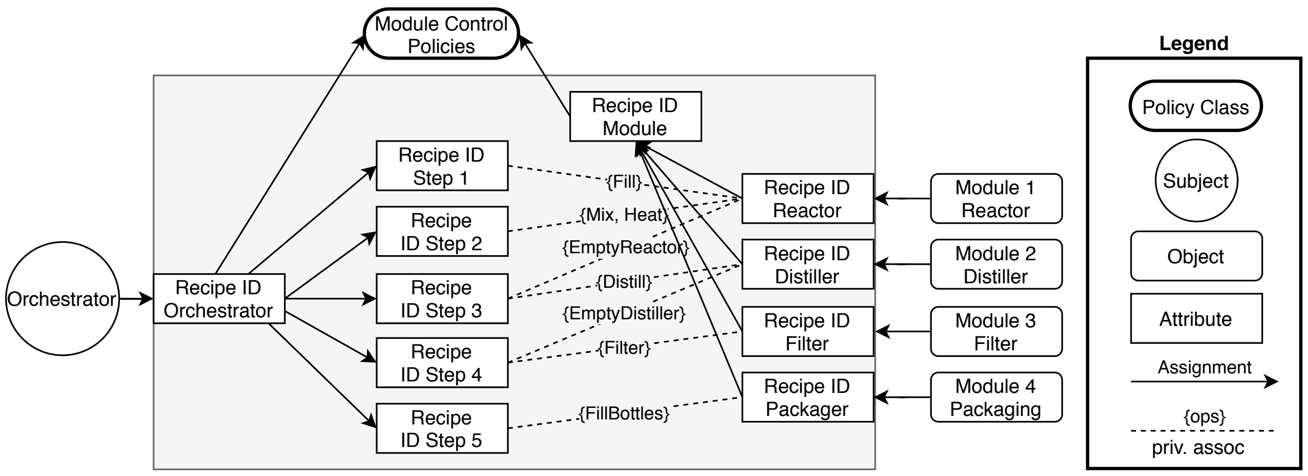

Fig. 4: Example of NGAC policy and attribute setup for the recipe described in Fig. 1.

attribute for all modules representing the recipe that the module currently is assigned to. An evaluation of the execution cost for such a growing number of attributes in NGAC should be performed, if considering using this approach in a scaled up scenario with real time requirements.

There are aspects of SFC recipes that cannot be captured by the suggested method for policy generation, e.g., related to the difficulty to express transitions between steps. Another aspect is the actual logic within one step of the SFC. There may be IF-conditions or loops that surrounds the module operations with additional logic, something not captured by the access control logic. A third aspect are parameters used for operations. A parameter set using a malicious value in an otherwise valid function call could have a harmful effect on the system.

Our suggested approach uses only positive grant policies, since they provide a natural way of describing the execution of an SFC. We do, however, not claim that this is always the best way of describing all kinds of recipe orchestration policies. There could be scenarios where combinations of grants and prohibitions provide a better solution, e.g., if a specific system state should prohibit an execution. For scenarios where policy evaluation leads to conflicting results, NGAC will always use the most restrictive outcome.

Despite these shortcomings, we see our approach as a potential mitigation against compromised devices in automation systems. Fine-grained access controls between devices is a useful additional layer of security which is not present at the moment, neither in traditional systems for process automation, nor in the current frameworks describing MA architectures.

A. On recipe activation

As mentioned, the module and orchestrator attribute assignments are omitted from the algorithm. In the following we provide some rationale behind that decision. First, as there may be considerable delay between the moment of completed integration engineering, in which recipe formulation is one part, and the start of production, there is a need for the graph to be created without granting any privileges. If an orchestrator attribute is assigned already at this point, the principle of least privilege would not be followed. Second, the generated graph can be used as a template. When e.g., increasing the production by adding additional production lines, there will be no need to generate new access control policies, instead the policies generated for the recipe can be reused as a template. Third, this approach allows for integration engineering using MTPs without the presence of the physical modules in the system, i.e., recipe formulation could be completed before a module procurement.

An alternative approach would be to not generate any part of the access control graph at all until the recipe is activated. This approach could be beneficial as the matching to modules and subject could be done with a minimal extension of the algorithm, and the total NGAC access control graphs would not be burdened with "unused" parts related to recipes not actively in use. However, there might be potential issues related to who has the privileges to perform the administrative operations on access control policies. An attribute assignment and provisioning is usually done locally, while policy administration is done centrally, in the same way as recipe activation and supervision is done by an operation engineer, while recipe formulation is done by an integration engineer.

B. On recipe de-activation / decommissioning

When a recipe should no longer be used in the MA system, the question arises about the best way to dispose it, such that the privileges granted under the recipe are no longer active, but the actions performed during the recipe life-time still can be explainable on review. One approach could be to remove all the attributes and associations related to the recipe from the NGAC graph. Another approach would be to keep the entire generated part of the graph, and only remove the attribute associations from the orchestrator and modules. The best choice depends on the expected life-cycle of a recipe.

C. On temporal policies and obligations

In the presented NGAC policy generation algorithm, the task transitions as described by the SFC are not at all considered. This is a violation against the principle of least privilege, since the orchestrator will be allowed to perform any of the operations prescribed by the SFC at any point in time, regardless of the current working step in the recipe. The use of *obligations* may be a way around this shortcoming.

Obligations in NGAC are described by a tuple (ep, r), informally expressed as "**when** ep **do** r", where ep is an event pattern and r is a response, containing one or more administrative operations. One example of an obligation in this context could be:
when *Orchestrator succesfully has performed Fill on Reactor* **do** *remove assignment of Orchestrator to Step 1, assign Orchestrator to Step 2.*
However, the obligations in NGAC are limited to describing policy-related events, i.e., there is no way of telling that the Orchestrator actually performed the Fill-operation on Reactor, only that the Orchestrator requested and has been granted (or denied) the right to perform the operation. Therefore, based on the current knowledge, the workflow characteristics of modular automation cannot be modeled using obligations in NGAC.

An alternative way of driving the workflow model would be to have an external entity assigning and de-assigning the recipe step attributes to the orchestrator following the SFC state model. Such a scheme would fulfill the least-privilege principle, but would defeat the purpose of having an orchestrator being responsible for driving the state model for the recipe. This kind of solution is however common in e.g., safety controllers, where a secondary safety module receive the same input as the primary controller, performs the same logic and compares the resulting output, forcing the controller to a safe state on deviating results.

VI. Related Work

Workflows as a basis for access control is discussed in several publications related to business process modeling and Process-Aware Information Systems (PAIS). A review of security related to PAIS is provided by Leitner et al. [19]. Knorr [20] discusses the use of workflows modeled as Petri-nets in an access control enforcement engine. Domingos et al. [21] suggest an access control model for adaptive work-flows, based on RBAC. These works relate to our approach in the use of formalized workflow models as a basis for authorization, while the difference lies in the application domain, where the PAIS typically is implemented as a part of a business process system, e.g., for document handling or similar, whereas our approach aims at industrial control systems.

Task-based authorization control (TBAC), by Thomas et al. [22], is Access Control model aiming at achieving similar objectives as the approach presented in this paper, i.e., limit access control to a just-in-time and need-to-do basis, following task descriptions. Also in this field, the target applications are, e.g., for transaction management- and information management-systems. Furthermore, this field of research has not materialized in any generally accepted standards, and there are no well established reference implementations available.

Ruland et al. [23] describe an access control system for smart energy grids and similar IACS. The system works in two stages, the first one is based on a limited set of policies expressed in XACML, the second stage uses knowledge about behavior of the system to prevent actions outside defined boundaries, to maintain safety properties of the system. This approach is similar to the one we suggest, as the expected behavior of the system is used as basis to formulate the secondary stage policies. However, the supported use cases are rather static, and there is no effort toward automation of policy formulation. Nevertheless, the idea of separating the privilege inference in several stages could be interesting, especially for real-time sensitive applications.

In the field of Model-Driven Security (MDS), originating from Model Driven Architecture, there is a body of research related to the design of secure systems, with regards to modeling, analysis as well as model transformation. Basin et al. [24], summarizes a lot of that work. The focus of MDS is mainly on the design phase for including security specific models when realizing a system architecture, by e.g., defining modeling languages for access control rules [25]. Most of MDS research is, with regards to access control, focused on the RBAC-model, there are however some examples utilizing attribute based access control; including Alam et al. [26] that describe a MDS approach for SOA, with XACML as policy expression language, and Lang et al. [18] that present a proximity-based access control model originating from the ABAC model, where the low-level policies are generated based on high-level policies described in natural language. An important argument from [18] is that: for ABAC in general, MDS is a requirement, as the low-level policy descriptions are so complex they cannot be managed without some amount of automation. To the best of our knowledge, there are no examples of MDS applied to policy automation in systems having properties similar to the ones of MA. In particular, we are not aware of any work covering a system where the policies needs to change dynamically, as required by the orchestration of modules in a MA system.

VII. Conclusions

In this work, a method for automated access policy generation in the context of MA is presented. The policies are generated using recipes expressed in SFCs, which is an industry standard for PLC programming in the 1131 family. The resulting policies are described in the format of an NGAC sub-graph. With this work we have shown that efficient policy generation is possible in an MA system without any additional work being performed by engineering personnel. Using this algorithm in an industrial system would increase the system overall security by decreasing the maneuverability and increasing the visibility of a compromised device.

Recalling the initially stated research questions: RQ1 is related to how to express policies. As an answer we have provided a definition applicable to policies expressed using the NGAC model. RQ2 relates to minimizing the management effort related to access policy formulation in an MA system. The presented algorithm is one answer to that, describing how to use an available workflow model to automate the policy generation without the need for additional engineering efforts.

As future work we envision creation of an experimental setup allowing simulation of an MA system, including both integration and operational engineering, which will contain a full access control enforcement architecture using NGAC as policy engine. This would be one way to further confirm the results in this article, with regards to scalability. We also plan to further investigate mechanisms to support more fine-grained workflow-related characteristics of MA.

Using XACML instead of NGAC to express policy rules is another natural continuation of this work, trying to evaluate if policy generation is feasible in that framework. As XACML allows for valued and typed attributes, the policy generation may in that context not need the same amount of synthetic attributes.

Moreover, automated access-policy generation is also of interest in wider domains than MA, e.g., in smart manufacturing and other dynamic and flexible systems requiring fine grained access control policies. Extending our results into these domains are possible directions for further research.

Acknowledgements

This work is supported by the industrial postgraduate school Automation Region Research Academy (ARRAY, funded by the Knowledge Foundation) and ABB. The authors would like to acknowledge Tomas Lindström for his valuable feedback.

References

[1] ZVEI—German Electrical and Electronic Manufacturers' Association, "Module-based production in the process industry—effects on automation in the "industrie 4.0" environment," White Paper, Frankfurt, 2015.

[2] J. Ladiges, A. Fay, T. Holm, U. Hempen, L. Urbas, M. Obst, and T. Albers, "Integration of modular process units into process control systems," *IEEE Transactions on Industry Applications*, vol. 54, pp. 1870–1880, March 2018.

[3] ZVEI—German Electrical and Electronic Manufacturers' Association, "Process INDUSTRIE 4.0: the age of modular production," White Paper, Frankfurt, 2019.

[4] K.-d. Thoben, S. Wiesner, and T. Wuest, ""Industrie 4.0" and Smart Manufacturing – A Review of Research Issues and Application Examples," *International Journal of Automation Technology*, no. January, 2017.

[5] J. Wan, S. Tang, Z. Shu, D. Li, S. Wang, M. Imran, and A. V. Vasilakos, "Software-Defined Industrial Internet of Things in the Context of Industry 4.0," *IEEE Sensors Journal*, vol. 16, no. 20, pp. 7373–7380, 2016.

[6] J. Slowik, "Evolution of ICS Attacks and the Prospects for Future Disruptive Events," tech. rep., 2017.

[7] M. Rocchetto and N. O. Tippenhauer, "On attacker models and profiles for cyber-physical systems," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9879 LNCS, pp. 427–449, 2016.

[8] E. D. Knapp and J. T. Langill, *Industrial Network Security: Securing critical infrastructure networks for smart grid, SCADA, and other Industrial Control Systems*. Syngress, 2014.

[9] J. Saltzer and M. Schroeder, "The Protection of Information in Computer Systems," in *proceedings of the IEEE*, vol. 63, pp. 1278–1308, September 1975.

[10] "IEC 62443 security for industrial automation and control systems," standard, Internation Electrotechnical Commission, Geneva, CH, 2009-2018.

[11] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, pp. 46–52, Oct 2003.

[12] "IEC 61131-3:2013 Programmable Controllers - Part 3: Programming Languages," standard, IEC, 2013.

[13] R. S. Sandhu and P. Samarati, "Access control: Principles and Practice," *IEEE Communications Magazine*, vol. 32, no. September, pp. 40–48, 1994.

[14] D. Ferraiolo, S. Gavrila, and W. Janse, "Policy Machine: Features, Architecture and Specification," white paper, NIST, October 2015.

[15] E. Yuan and J. Tong, "Attributed Based Access Control (ABAC) for web services," in *Proceedings - 2005 IEEE International Conference on Web Services, ICWS 2005*, vol. 2005, pp. 561–569, 2005.

[16] D. Ferraiolo, R. Chandramouli, R. Kuhn, and V. Hu, "Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC)," pp. 13–24, 2016.

[17] "eXtensible Access Control Markup Language (XACML) version 3.0 plus errata 01," standard, OASIS, 2017.

[18] U. Lang and R. Schreiner, "Proximity-based access control (pbac) using model-driven security," in *ISSE 2015* (H. Reimer, N. Pohlmann, and W. Schneider, eds.), (Wiesbaden), pp. 157–170, Springer Fachmedien Wiesbaden, 2015.

[19] M. Leitner and S. Rinderle-Ma, "A systematic review on security in process-aware information systems – constitution, challenges, and future directions," *Information and Software Technology*, vol. 56, no. 3, pp. 273 – 293, 2014.

[20] K. Knorr, "Dynamic access control through petri net workflows," in *Proceedings 16th Annual Computer Security Applications Conference (ACSAC'00)*, pp. 159–167, IEEE, 2000.

[21] D. Domingos, A. Rito-Silva, and P. Veiga, "Authorization and access control in adaptive workflows," in *European Symposium on Research in Computer Security*, pp. 23–38, Springer, 2003.

[22] R. K. Thomas and R. S. Sandhu, "Task-based authorization controls (tbac): A family of models for active and enterprise-oriented authorization management," in *Database Security XI*, pp. 166–181, Springer, 1998.

[23] C. Ruland and J. Sassmannshausen, "Access Control in Safety Critical Environments," in *Proceedings - 12th International Conference on Reliability, Maintainability, and Safety, ICRMS 2018*, pp. 223–229, IEEE, 2018.

[24] D. Basin, M. Clavel, and M. Egea, "A decade of model-driven security," in *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies*, SACMAT '11, (New York, NY, USA), p. 1–10, Association for Computing Machinery, 2011.

[25] T. Lodderstedt, D. Basin, and J. Doser, "SecureUML: A UML-based modeling Language for model-driven security," in *International conference on model engineering, concepts and tools*, 2002.

[26] M. Alam, R. Breu, and M. Hafner, "Model-driven security engineering for trust management in SECTET," *Journal of Software*, vol. 2, no. 1, pp. 47–59, 2007.