




# Securing Emergent IoT Applications

Prabhakaran Kasinathan<sup>1,2</sup>(✉)  and Jorge Cuellar<sup>1,2</sup>

<sup>1</sup> Siemens AG, CT, IT Security, Munich, Germany  
{prabhakaran.kasinathan,jorge.cuellar}@siemens.com  
<sup>2</sup> University of Passau, Passau, Germany

**Abstract.** Attacks on IoT, Cyber-Physical-Systems (CPS), and other computing systems are evolving rapidly. As a result, IoT devices used in critical infrastructures such as energy, health-care, and water supply systems are vulnerable to attacks. A successful attack on such safety-critical infrastructures may have life-threatening consequences. On the other hand, existing security mechanisms are not enough to protect constrained IoT devices. Therefore, we need better security mechanisms and tools to manage and protect IoT devices from malicious use.

In emerging paradigms like Internet-of-Things (IoT) platforms, Industry 4.0, collaborative portals, and many others, we deal with a multi-tenant architecture. In a multi-tenant architecture, the owners want to secure their own integrity, confidentiality, and functionality goals without being concerned about the goals of other entities. In this paper, we present a framework to negotiate, compromise, and inter-operate between different services or platforms to fulfill a *purpose*. Furthermore, to ensure correct and safe operation of IoT systems, we must assure that the integrity of the underlying systems and processes is properly executed as intended i.e., the processes cannot be changed in an unauthorized way.

In this paper, we present our Petri Net based workflow specification and enforcement framework to realize *workflow-aware access control* and to protect the *process integrity* of IoT applications. The Petri Net models are amenable to formal verification. The resulting workflows have other properties such as the ability to recover from error conditions. In addition, we present a method to achieve distributed access control and accountability integrated with our framework. We allow practitioner-friendly tools to collect requirements and goals to design secure IoT systems and processes. Finally, we present a guide to implement our framework with existing development environments and validate the methodology using concrete use case scenarios.

## 1 Introduction

The EU Research Cluster on IoT (IERC) [68] defines Internet of Things (IoT) as “an infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual *things* have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network”.

Constrained IoT devices are categorized by their ability to process and store data, energy consumption, and communication capabilities (see [12]). *Class 0* devices are really constrained sensor-like motes with less than 10 KB of RAM and 100 KB of flash memory. *Class 1* devices are quite constrained, and cannot use standard Internet Protocol stack; however, class 1 devices support protocols designed for constrained devices. *Class 2* devices are less constrained, can support some security functionalities specifically designed for constrained devices. Finally, the devices with capabilities beyond class 2 support most of the traditional Internet and security protocols like HTTP and TLS; however, they can still be constrained by limited energy supply. Generally, IoT devices use both long and short-range communication technologies such as Zigbee, Bluetooth, LTE, etc. combined with constrained communication protocols such as Constrained Application Protocol (CoAP) for Internet connectivity. Constrained IoT devices are cheap, compact, easy to deploy, and consume less energy. Recently, organizations use data collected from IoT devices to get insights, predict, and to optimize their services with the help of Artificial Intelligence (AI) technologies. This approach is used in various applications such as smart manufacturing, industrial control systems, financial services, retail, intelligent logistics, transportation, medical and healthcare applications, smart grid, intelligent traffic, environmental monitoring, smart home, assisted living, agriculture, and many more.

Constrained IoT devices are vulnerable to attacks because existing state-of-the-art security mechanisms do not fit within the constrained devices and often they can be accessed physically by an attacker. For example, modern remote attestation technique is difficult to achieve in constrained IoT devices because of the lack of space and processing power [13, 61]. Implementing secure key generation and key storage in constrained IoT devices are hard (see guidelines from Trusted Computing Group [71]) because these devices lack sufficient entropy to generate random numbers and are prone to side-channel attacks. Several attacks on industrial IoT devices are presented in [61]. Due to the vulnerabilities, hackers frequently target IoT devices to escalate attacks on valuable assets. The 21st century has seen a sudden rise of insecure IoT devices in an unexpected scale which require immediate attention i.e., we must secure those emergent IoT devices. Now, since IoT devices are used in critical infrastructures, it is evident that we need better security mechanisms and tools to protect them. Researchers in academia and industry are working together to secure emergent IoT devices, protocols, and applications. Furthermore, IoT devices collect personal information or data that can be used to infer private activities or habits without proper consent. As a consequence, the European Union’s privacy regulation GDPR (see [22]) has enforced strict regulations for handling private information of users.

In emergent IoT applications, the multi-tenant architecture is more prominent. In such applications, different entities provide and consume services from one another and each entities might want to enforce their own integrity, confidentiality or functionality goals on other entities consuming his service. The main problem with multi-tenant systems and architecture is the “trust problem”: in

order to achieve his goals, each party requires that the others behave in a particular way. Ideally, the party would like to specify a “contract” that declares his assumptions about the behavior of other entities as well as the guarantees that he offers to them about his actions. But how can he trust other parties to behave according to the contract? How can he be sure that they do not “cheat”? In this paper, we investigate a way to automatically enforce a contract.

This is the purpose of a “smart contract”: it declares what happens if some of the parties misbehave and what will happen in case of other error or unexpected conditions. Each party imposes his rules on entities while they interact with his services. In the case of electronic money, this is easier to enforce: the party that cheats lose money. In the case where money is not available directly, it may be difficult to penalize a party that is not complying with the stipulated rules. In general, those sequences of interactions defined by “smart contracts” may be seen as a set of allowed actions, or in other words, a workflow. Clearly, there is a need to negotiate, compromise, and inter-operate the tasks to be completed by the different entities within such a system.

To enforce such tasks to be executed in a particular order we need a workflow specification and enforcement method. It is important to notice that securing the assumption-commitment semantics of a smart contract is also the key for its verification. The smart contracts are given as a refinement of Petri Nets, which are subject to verification, see [59].

More specifically, we use the Petri Net based Workflow Specification and Enforcement method presented in [39,40] to write such smart contracts which guarantee the integrity of processes. The method also supports dynamic workflows that adapt to error conditions by allowing services to create on the fly sub-workflows. Furthermore, the framework provides accountability and transparency without assuming a central authority.

## 1.1 Security and Privacy Challenges in IoT

Security and Privacy challenges in IoT and Industrial Internet of Things (IIoT) are discussed in [61,66,76] where the authors discuss technical, financial and legal issues involved in IoT and existing solutions. In this paper, we discuss the technical aspects of security and privacy challenges in IoT/IIoT. The OWASP (Open Web Application Security Project) presented the Top 10 IoT vulnerabilities and attack surfaces (see [47]), we discuss the topics relevant to this paper here:

- *Authentication and Authorization*: the goal of an authentication system is to verify that entities are correctly identified [11]. After authenticating an entity, the security mechanism of verifying whether the entity is allowed to perform certain actions is known as authorization. Existing state-of-the-art authentication and authorization mechanisms do not fit in constrained IoT devices; academic and industrial researchers are working towards addressing them (see IETF ACE working group [32]). Since IoT devices are cheap, they do not have interactive interfaces to implement traditional authentication

mechanisms such as a display to present security info to the user, or a keypad to enter passwords. Sometimes, even when proper security mechanisms are in place, users do not use them properly. For example, the default password for many IoT devices is not changed by their users because of its complexity i.e., the user needs to connect the device to the local network and login into it via a web interface using default credentials to change it. For instance, hackers have used this vulnerability to mount denial-of-service attacks on popular websites by sending remote commands to billions of IoT devices - see Mirai botnet attack [4]. On the other hand, most IoT devices implement single-factor authentication such as the username and password, and authorization does not consider the context of activities involved like tasks in a workflow.

- *Confidentiality and privacy*: IoT devices can collect sensitive information, including personal data. Therefore, the data subjects want their information to be confidential. Constrained IoT devices cannot use standard encryption mechanisms, such as Transport Layer Security (TLS). Light-weight protocols, such as Datagram Transport Layer Security (DTLS) over CoAP (See [24]) have been designed to support the confidentiality and integrity of transported data. One of the challenges is that for instance, class 1 devices cannot properly support DTLS, and therefore, packet losses will result in retransmission of messages, affecting the performance of battery powered devices. Compromised devices holding private data will expose information about the private life of the data subjects. This demands the need for privacy-preserving (enhancing) and confidentiality mechanisms integrated with the IoT device communication [76].
- *Integrity*: there are at least three aspects of integrity. First, we have data integrity – the assurance that the data transferred from one entity to another has not been altered or tampered with. Second, we have the integrity of data stored in memory – this includes, firmware, key material, data, or programs stored in memory – is not altered. Third, we have *process integrity*. A business/technical process specified must be executed as it is specified i.e., no one is able to change, add additional steps or skip steps defined in the process. This property is called “process integrity”, and it is discussed in detail below. Security mechanisms such as Message Authenticate Code (MAC) exist to ensure data integrity, and hardware or software attestation techniques exist to ensure the integrity of firmware or application code. But achieving process integrity is difficult, and no solutions exist to enforce it. One of the main goals of this paper is to specify a process and ensure that it is properly enforced on the entities executing it.
- *Interoperability*: IoT devices are heterogeneous in terms of processing power, memory capacity, and communication technologies. Some IoT devices may or may not operate with each other because of non-interoperable standards. Different organizations collaborate to create interoperable standards such as the Alliance for the Internet of Things Innovation (AIOTI) (see [3]). Also, the research community such as ACE (see [32]) is working towards standardizing security protocols to make IoT devices interoperable and secure. In particular,

we need interoperable security mechanisms that can be implemented on the majority of IoT devices.

- *Self-Configuration and Multi-Tenancy*: is evident that IoT devices are getting powerful, cheaper, (see Moore’s law [63]) and energy-efficient day-by-day. Installing and configuring such advanced IoT devices with existing IoT applications should not require too much human involvement. The IoT devices should have self-configuring features i.e., backward compatibility, resilient to connection losses and device failures, etc. In such error cases, the IoT system must re-adapt to the changes and work normally. Multi-tenancy refers to the fact that devices or services belong to different owners with different or competing goals. Those parties prefer to cooperate by exchanging information with each other such that both parties will profit from information or activities exchanged. IoT devices need to support such kind of multi-tenant features without losing the security requirements of parties involved.

**Protecting the Process Integrity of IoT Applications.** A *process* is a set of interrelated activities or tasks that must be carried out to accomplish a goal [11]. A business/technical process is also called a workflow, but we use the two words as synonymous. Different owners/stakeholders of devices or services will probably try optimizing their own results and to secure their own integrity, confidentiality or functionality goals, without really being concerned about the goals of other entities. We call this property as Multi-Tenancy. We need a method to protect the integrity of business processes of each owner/stakeholder without compromising the integrity of the process of other involved entities.

A workflow can be defined as a pattern of activities or tasks to be completed in a particular partial order by the involved entities, following predefined rules, in order to accomplish a specific goal or subgoal. A workflow must be executed as it is specified i.e., ensuring *process-integrity*. During the execution of the workflow, the participants may exchange with each other documents, information, see [77]. Confidentiality is not as important as the availability and integrity of the cybersecurity processes, which is mission-critical. Achieving process integrity of different owners/stakeholders collaborating with each other is the main focus of the paper.

We describe a small case study to gather the requirements, study the challenges, and to formulate the goals of our work. Let us consider the following use case scenario (UC1): a manufacturing company requires continuous monitoring and maintenance of equipment in its factory. For example, IoT devices are used to monitor temperature, smoke, and fire, etc. IoT devices can also be used as actuators to control access to doors, equipment, and emergency exits. The provenance of IoT devices, quality, and maintenance of the manufacturing plant are strictly enforced by predefined processes (workflows) defined by the manufacturing owner. The integrity of such processes must be enforced to ensure quality products being produced in the plant. Usually, a manufacturing plant consists of different equipment or systems from different manufacturers, each will have their own maintenance processes. If the production stops because of an equipment malfunction or a supply chain problem or a worker who failed to follow

the predefined rules, etc., then the problem must be identified and addressed as soon as possible. To ensure the integrity of the processes strict access control methods must be used. With this use case, we will formulate the requirements, challenges, and goals of our work.

## 1.2 Goals of Our Framework

We want our framework to have a *workflow-driven access control* in contrast to the commonly used mandatory (MAC), discretionary (DAC), or role-based access control (RBAC), which have been well-studied in the literature, see [62]. Thus, the goals of our framework are:

- To provide a generic, interoperable, and distributed workflow-aware access control method that restricts the entities to execute tasks in a predefined order defined in the workflow. By doing this, we can guarantee the process integrity of that particular workflow.
- Our Petri Net based workflow specification and enforcement method should be interoperable i.e., it should support existing authorization standards such as OAuth.
- Our method should support dynamic workflows that adapt to error conditions i.e., allowing services to interact with each other and create on the fly sub-workflows without changing the objective of the main workflow.
- Our framework should be extendable and support the integration of practitioner-friendly tools.
- Our framework should support distributed accountability i.e., when necessary, we can prove the actions of entities executing the workflow.

In this paper, we extend our Petri Net based workflow specification and enforcement framework presented in [39,40] to present a comprehensive access control security framework for the Internet of Things (IoT); however, this approach can be applied to any generic computing system. The main contributions of this paper are: first, we summarize our Petri Net extensions such as timeout transitions contracts and open Petri Net places; second, we extend our framework to support requirement elicitation methods with practitioner-friendly tools, distributed accountability, and generation of Petri Net based smart contracts for Blockchain; third, we use our framework to solve three use case scenarios; finally, we present a high-level guide to implement our framework with existing systems.

To summarize, we present a method:

- To specify processes as workflows that can be created in a stepwise manner using standard software engineering processes and tools. Such workflows specified as Petri Nets are amenable to formal verification.
- To constrain an entity using an application/services to obey a prescribed workflow with fine-grained authorization constraints based on *least privilege* and *need to access* principle.
- That allows entities participating in a workflow to have a choice, for example, to accept (or reject) “contracts” or conditions.

- That allows services and entities executing a workflow to handle error conditions by supporting the creation of dynamic workflows, and that provides accountability without assuming a central authority.
- To exchange authorization tokens in a secure and privacy-enhanced way. Note: this method can also be used to transfer other tokens (such as money, information, etc.) not just authorization tokens.
- To support distributed accountability while executing the workflow i.e., actions executed by entities executing the workflow is recorded in an immutable database.
- To support the generation of Petri Net based smart contracts to be deployed in a Blockchain.

**The Rest of the Paper is Organized as Follows:** Section 2 describes security and privacy requirements of IoT and motivates the need for advanced security mechanisms such as workflow-aware access control methods for emergent IoT applications; Sect. 3 describes the evolution and background of Petri Nets; Sect. 4 describes the existing background work published in the literature; Sect. 5 presents the contributions of our work; Sect. 6 describes three different use case scenarios where we apply our method and solve them; Sect. 7 describes a high-level summary of our method and a guide to implement our method with existing systems; finally, we present limitations of our approach in Sect. 8 and conclusion in Sect. 9.

## 2 Security and Privacy Requirements for IoT Applications

The technical challenges of securing emerging IoT applications were described in Sect. 1.1. Now, we discuss the relevant security and privacy *requirements* for securing the emergent IoT applications. In particular, we refer to the maintenance of manufacturing plant use case scenario UC1 to formulate the following requirements.

### 2.1 Requirements Elicitation

The requirements engineering process can be divided into four tasks namely the elicitation, negotiation, specification/documentation, and verification/validation of requirements [55]. When we want to solve a problem, first, we need to gather more information about the problem i.e., elicitation of the requirements, needs, and constraints about the system. Often, information about the problem (or system) is distributed among many stakeholders i.e., the knowledge is not available from one source (user or customer). Therefore, the identification of the relevant sources during the elicitation task is crucial. Modern tools such as Unified Modeling Language (UML) or Systems Modeling Language (SysML) allow us to collect requirements, use cases, draw activity diagrams, and finally to validate requirements of a complete system. In particular, SysML provides tight

integration of both software and hardware components. Thus, requirement elicitation is important to understand the problem and to gather requirements. For example, in UC1, we need to understand which processes are critical and the actors involved in the manufacturing plant. A detailed interview with managers and workers handling the production equipment and IoT devices will give the required information to define a workflow.

## 2.2 Distributed Authorization

Distributed authorization mechanisms are important to support a growing number of IoT devices. Authorization in distributed systems is complex to achieve [25] as the resources are spread across a network of devices under different domains, multi-tenant systems, and they might know each other or not. As described earlier in the introduction, this is a trust problem. A smart lock installed in a smart home opens or closes the door based on the access control (AC) policies defined by the owner. The owner may use his smartphone to present his credentials to the smart lock. The smart lock may use OAuth based mechanism to verify authorization tokens and update its AC policies periodically. From the perspective of an IoT device (i.e., smart lock), whenever a request from a Client (i.e., in this case, the owner's smartphone) arrives, the IoT device evaluates the authorization token attached with the request and sends an appropriate response. This standard approach (for instance, IETF ACE [32]) ensures interoperability. In our work, we introduce changes only to the clients and to the authorization service, but not to the IoT devices. For example, in UC1 there could be several scenarios where we need distributed authorization. For example, scenario 1: a worker wants to update some software in an IoT device; for this purpose, the administrator authorizes the worker. Scenario 2: an IoT device needs to authenticate, present authorization credentials to a secured server to write some data; for this purpose, the IoT device needs to get an authorization token from an authorization server. The role of the client and resource server from the context of OAuth ACE protocol changes depending on the use case but clearly we need distributed authorization. More information about this topic is presented in Sect. 4.1.

## 2.3 Device Commissioning and Secure Software Updates

Often, IoT devices are deployed in large scale. To protect that infrastructure, it is important to deploy devices with unique authentication credentials. Secure device commissioning i.e., key-provisioning, device hardening, etc. helps to protect the device from attacks, and also perform secure software updates. Software updates are often required to fix the software bugs or vulnerabilities in any computer software. In particular, firmware updates can patch vulnerable IoT devices, but an update from an untrusted source can install a Trojan or malware into the device. Various commercial software update solutions exist, but they are not interoperable and may not work with constrained devices. The IETF working



group - ‘software updates for IoT’ [72] is working towards creating an interoperable and secure software update solution for IoT devices (class 1 or above) with approximately 100 KB of flash memory. Commissioning a large number of IoT devices is still a challenge, we need automated tools, protocols for secure device commissioning (see Enrollment over Secure Transport (EST) is used as a certificate provisioning protocol over HTTPS [67]). For example, in UC1, secure device commissioning is crucial to ensure that deployed manufacturing equipment and IoT devices are malware free, credentials provisioned are safe, etc. After deploying the equipment or IoT device, it is important to have the ability to provide updates i.e., for introducing new features, roll back to the previous stable state, or apply security-patches for the existing software, etc.

## 2.4 Attack Escalation Resilience

Compromising one IoT device means that the attacker can escalate the attack on other IoT devices or systems connected to the same network. Attack escalation is a serious problem, and we need resilience mechanisms. Authorization coupled with the context of task execution workflow stops the attack escalation problem. In this work, we describe a workflow-aware access control method which prevents attack escalation to an extent. On the other hand, when multicast security is used i.e., a group key is used for controlling a set of IoT devices. The IETF RFC [26] specifies requirements and security considerations for generic group key management protocols. The IETF draft [74] specifies a secure group communication for IoT devices that use the Constrained Application Protocol (CoAP). In this work, we do not focus on multicast security. For example, in UC1, let us assume that one of the IoT devices is physically accessible at the perimeter of the manufacturing plant and the IoT device is compromised by an attacker (how it is compromised is out of scope). For instance, the attacker may plan to escalate the attack by accessing other devices via the network. Therefore, we need proper security mechanisms to restrict the attacker from compromising other devices or equipment via a weak compromised device. Let us assume that there exists a workflow for initializing software update or updating the configuration of devices inside the manufacturing plant, then the attacker cannot perform his attack unless he was able to execute that workflow and reach the state which allows him to perform a software update. Note: at the first place, we should have proper access control and authorization mechanisms for initializing and executing the workflow.

## 2.5 Fine-Grained Access Control

In common Access Control (AC) methods such as role-based access control (RBAC) see [62], access control and authorization is given to an entity based on a Role. A role like *admin* is very powerful and has (almost all) permissions such as to change, add, and delete features of a system. If such an entity (with admin role) is compromised, then the attacker can do a lot of damage. Therefore, we want to limit the set of permissions (fine-grained) given to an entity based on

a workflow i.e., an entity can complete/execute a legitimate set of actions/tasks in a particular order defined in the workflow. This motivates the need for a fine-grained access control model such as the *workflow-aware access control*. Such access control methods can protect the assets to an extent even if an entity is compromised i.e., the entity should be executing the workflow in order to access a particular service. For example, in UC1, it is a bad idea to give access to all equipment and IoT devices to one single administrator account, because if that admin credential is stolen or misused, then the attacker is able to access entire system associated with the credential.

To achieve this, we need a *least privilege* principle for task authorizations within each workflow. The least privilege principle is a security concept where every computer module (such as a process, user, or program, depending on the subject) may be able to access only the information and resources that are necessary for its legitimate purpose. As a particular case, the principle “Need to Know” is a confidentiality policy which states that no subject should be able to read objects unless reading them is necessary for that subject to perform its functions [11]. What we need is a similar policy, but regarding integrity. We call this principle “Need to Access”: it states that no subject should be able to *write or change* objects unless it is necessary to complete the required task of a process or workflow at that particular state. By enforcing the need to access principle, an entity can get privileges to execute a task only at the required step of the workflow. This provides workflow-driven (workflow-aware) access control.

The workflow-aware access control needs an error-free workflow (free from deadlocks) and a device to execute it. A powerful computing device like a smartphone is used to execute the workflow, not a constrained IoT device. Any generic application logic or process that we want to enforce is represented as one or more workflows. We elaborate further the requirements of the workflow-aware access control below.

**Verification of Workflows.** Formal methods refer to mathematically rigorous techniques and tools for specification, design, verification of software and hardware systems. Formal verification is the act of proving or disproving the correctness of a system with respect to a certain formal specification or property [82]. A verified system may satisfy safety and liveness properties such as no deadlocks, mutual exclusion is satisfied, each request will have a response, freedom from starvation, etc. Therefore, we need a modeling language with which we can verify some properties such as deadlocks in a workflow. A survey of formal verification of business process modeling is presented in [49].

**Adapt and Recover from Error Situations.** An error-resilient IoT application should be capable of recovering from unforeseen error situations to an extent. Therefore, it is important to allow human interaction to solve a problem that cannot be fixed by the system itself. A workflow may allow the owner of the services to create on the fly sub-workflows without changing the main workflow to recover from error conditions. This requirement is necessary to build a usable security in an IoT application.

## 2.6 Distributed Accountability

Accountability is a fact or condition where an entity is accountable for actions committed directly or indirectly. To enforce accountability in a system, we must record (e.g., log) all important actions/interactions of an entity with the system, including solicitation and execution. Logging is a standard feature in many computing systems, it records system activities, process executions, user interactions, etc. with relevant information such as timestamps and user identifier. Thus, logging helps to achieve accountability. An accountability system needs more than just logging i.e., it should satisfy integrity requirements of logs generated and stored by all processes. For example, the logs cannot be tampered or destroyed in case of an attack i.e., mirroring logs on different servers or backup solutions is necessary. Such accountability information is commonly used to perform various analysis such as auditing and forensic security analysis. Auditing is an independent analysis of accounting records i.e., in a computer system, it can be a program trace, log information, etc. Forensic security analysis is performed to investigate a computer attack i.e., to find bugs in software processes, irregularities, and frauds committed by people, malware, etc. For example, in UC1, in case of an attack or system/equipment failure, the production plant auditors must have the capability to find the root cause of the incident. For this purpose, we need proper accountability mechanisms by default. An accountability system records every major decision (e.g., change logs, etc.) taken by the administrators or workers.

## 3 Evolution of Petri Nets

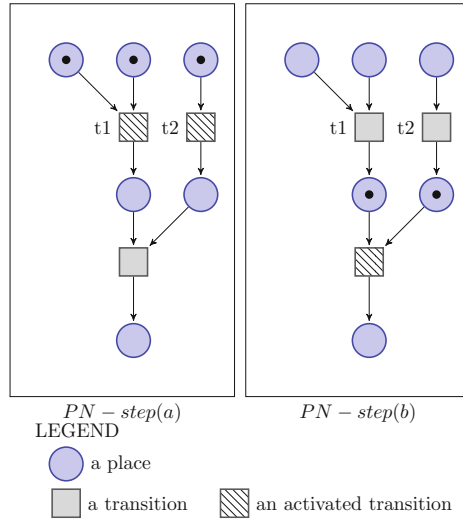
In this section, we introduce Petri Nets and evolution over the decades since their inception on the year 1966. In traditional Petri Nets (PN) (see [54]), there are places, tokens, and transitions.

A place in a traditional Petri Net can hold one or more tokens (*markings*) of the same type. A transition may have one or more input and output places. A transition fires if its input places have sufficient tokens and as a result, it produces tokens in output places. We recall the classical definition of a Petri Net (P/T net) from [60, 78]:

A Petri Net is a triple  $(P, T, F)$ , where

- $P$  is a finite set of places,
- $T$  is a finite set of transitions ( $P \cap T = \emptyset$ )
- $F \subseteq (P \times T) \cup (T \times P)$  is a finite set of arcs (the flow relation)

A transition  $t$  has input and output places. A place  $p$  is input or output for transition  $t$  based on the directed arc from  $p$  to  $t$  or from  $t$  to  $p$ . A place can contain zero or more tokens. A token is represented by a black dot  $\bullet$ . The global state of a Petri Net, also called a marking, is the distribution of tokens over places. Formally, a state or marking  $M$  is a function  $M : P \rightarrow \mathbb{N}$  that assigns to every place  $p$  the number of tokens  $M(p)$  that reside in  $p$ . We use the notation



**Fig. 1.** PN-Step(a) shows the initial state of a Petri Net and PN-Step(b) shows the state of the Petri Net after transitions  $t1$  and  $t2$  have fired.

$\bullet t$  to denote the set of input places for a transition  $t$ ; similarly,  $t\bullet$ ,  $\bullet p$  and  $p\bullet$ . Figure 1 shows a simple Petri Net in two steps: first, in step(a) transitions  $t1$  and  $t2$  are activated because  $\bullet t1$  and  $\bullet t2$  have sufficient tokens; second, in step(b)  $t1$  and  $t2$  fire to produce tokens in output places of  $t1\bullet$  and  $t2\bullet$ .

Several extensions of Petri Nets such as Time Petri Nets and Colored Petri Nets have enabled us to model different constraints such as time and types of tokens, and so on. Thus, Petri Nets were widely used in various application areas to verify network protocols, supply chain, etc. For a deeper understanding of Petri Nets, we recommend the book of Reisig [60] to the readers. We briefly present the most important extensions of Petri Net relevant for our work below.

**Time Petri Nets (TPN)** is used to model and simulate real systems as it is often important to describe the temporal behavior of the system, i.e., we need a way to model duration and delays (time) of transition firing (see [46]). The classical Petri Net is not capable of handling this.

**Colored Petri Nets (CPN)** is an extension of Petri Nets where different types of tokens can exist in the same place (see [34,35]). In a colored Petri Net, each token is represented by specific colors (types). CPN have the same kind of concurrency properties as Place/Transition Nets. Different tools such as CPN-Tools [37] are available to model and validate concurrent systems.

**High-level Petri Nets** simplify the process of creating complex workflows by breaking them into smaller partial workflows. At a high-level, it provides an overall description of the process without considering all details. As we navigate to a lower level, it provides in-depth description that particular component. The extension of Petri Net with *color*, *time* and *hierarchy* allows us to model complex

industrial systems with several layers of hierarchy without losing the details (see [2, 36]).

**Workflow Nets (WF-net)** are used to model a typical business process workflow using Petri Nets. Research advancements in the area of workflow nets contributed to our research. Most of the research discusses about mapping workflow concepts such as task execution, synchronization (split and join) actions, etc. into Petri Nets (see [1, 78]). Workflow Nets showed that Petri Nets can be used to design and model complex workflows. In addition, Petri Net tools can be used to verify traditional Petri Net properties such as liveness, etc. in Workflow Nets.

**Open Petri Nets** provide interfaces that enable two or more workflows to exchange information in the form of tokens. Open Petri Nets provide entry and exit points via Open Petri Net places to exchange information between workflows (see [27]). One of the goals of this work is to support multi-tenancy, i.e., to support activities, tasks from different organizations. *Composition* is a common approach in software engineering i.e., to assemble small systems into larger ones. Reisig in [60] describes the composition of nets using *interfaces* that can be used for asynchronous and directed communication between Petri Nets.

Petri Nets and its applications are well studied in the literature. Petri Nets enable us to create verified workflows with properties like guaranteed termination, separation-of-duties, reachability, liveness (deadlock-free), and coverability [1, 19, 51]. In this section, we presented the important extensions of Petri Net that help us to specify and verify workflows. By enforcing verified workflows with fine-grained access control, we achieve workflow-aware access control.

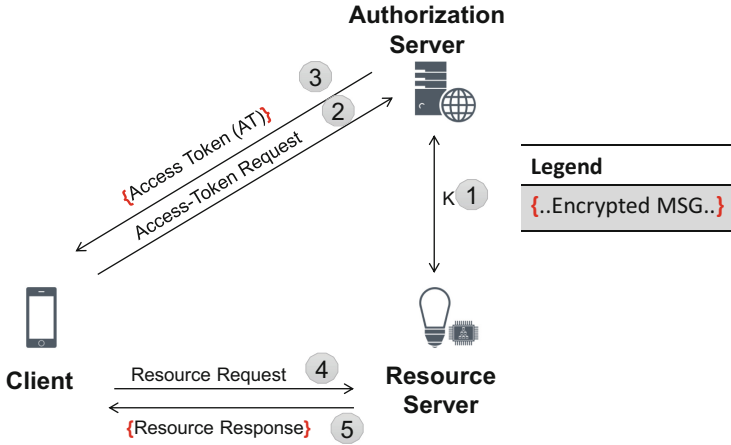
## 4 Background Work

In this section, we present relevant background and existing work on the three topics we focus in this paper.

### 4.1 Authorization for Constrained IoT Devices

Authorization mechanisms are important to restrict or allow an entity to access a resource in an IoT device. One of the important goals of our workflow-aware access control is to use appropriate authorization tokens within the workflow. Therefore, we present the state-of-the-art authorization methods for IoT in this section.

The OAuth 2.0 was developed for the web to create and transfer authorization tokens to an authenticated entity that wants to access a resource from the server. For instance, a browser is typically the client and a resource in OAuth 2.0 can be a restricted web-page (that needs special access rights) hosted on a server. The IETF working group (WG) Authentication and Authorization for constrained devices (ACE) [32] is specifying a framework for authentication and authorization in IoT environments called “ACE-OAuth” [65].



**Fig. 2.** An example ACE-OAuth scenario and actors involved. The numbers explain the sequence of an authorization process and resource request between three actors. Notations:  $K$  is a shared secret and {encrypted message}.

ACE-OAuth is based on OAuth 2.0 and CoAP. The motivation of ACE-OAuth is to create an authorization solution suitable for IoT devices. To describe the ACE-OAuth actors, let us consider an example use case. John owns a smartwatch (a typical IoT consumer device), and with that he wants to track, store his steps, heartbeat, etc. John wants complete control over his data i.e., deleting information stored on the device or in the cloud. John uses his smartphone to access or modify information stored on his smartwatch. For special access i.e., deleting information or changing the owner information on his smartwatch, John needs an access token from the cloud service provided by the smartwatch manufacturer. Thus, we can map the use case actors with the ACE-OAuth actors: the smartphone is a client (C), the smartwatch is the resource server (RS), the cloud service is the authorization server (AS), and John is the resource owner (RO). Below, we describe the simple ACE-OAuth messages exchanges to create the access token required by the client to access a resource on a resource server.

In Fig. 2, we show three important actors of ACE scenario. ACE-OAuth uses the term *Resource Server (RS)* to represent an IoT Device with several resources, i.e., typically sensors such as temperature, heartbeat recognizing sensor, gyroscope, etc. A smart lock, smart bulb, or a building automation device is a typical example of a resource server. The term *Client (C)* is used to represent the device that the resource owner (RO) uses to access the resource on an IoT device. Sometimes, simple client functionalities are embedded into the IoT device itself. For example, a user can access or modify certain functions on his smartwatch via the on-board display. Typically, an *authorization server (AS)* creates an access token and transfers it to the client. Now, we describe a particular ACE scenario as shown in Fig. 2: to access a resource on a Resource Server (RS), a Client (C) should request an access-token (AT) from AS, either directly or using its Client

Authorization Server (CAS). For the sake of simplicity, we do not consider introspective calls between the resource server and the authorization server or client authorization server.

Based on the above described scenario, a simple ACE OAuth message flow as shown in Fig. 2 can be described as follows:

- A C may perform a resource-request to RS without a valid access-token, then RS will reject, and it may provide AS information to the C in the response. Such that, the C may go to the AS to get a valid access-token. The Resource Owner (RO) may define access control policies on the Authorization Server (AS) describing who can access the resources on a RS.
- (1) A common secret ( $k$ ) is shared between the AS and RS while device commissioning. We assume that RS stays offline after deployment and cannot perform introspective calls to AS to verify the access token presented by the C.
- (2) The C performs an Access-Request to AS to ask for an access token (AT) that allows accessing the required resource (R) on RS. The AS checks if C can access the resource (R) on RS or not, based on permissions assigned by the RO.
- (3) If C has sufficient permissions, then AS generates an Access-Token (AT) plus a proof-of-possession (PoP) key bounded to the access-token and the secret ( $k$ ). AS sends both the AT and the PoP key to C via a secure encrypted channel.
- (4) After receiving AT and PoP key, C performs a resource-request to RS by ACE-OAuth token construction method defined in one of the ACE profiles. For example, the client may use privacy enhanced token construction method as described below.
- (5) The RS can reconstruct the PoP key from the AT and verifies the received AT. If it is valid, RS encrypts the response with the PoP key.

In the ACE working group, several other proposals with different profiles exist to solve specific problems. One of the proposed profile is Privacy-Enhanced Authorization token (PAT) profile. Note: at the time of writing this paper PAT profile was expired.

**Privacy-Enhanced Authorization tokens (PAT)** is a profile specified for ACE-OAuth [16] with a special focus on creating privacy-enhanced unlinkable authorization tokens. The PAT profile for ACE-OAuth provides unlinkability features even when a client performs non-encrypted authorization requests (i.e., sending request without network or transport layer encryption such as DTLS). PAT was designed such that the Resource Server (RS) is able to verify the access tokens without performing the introspective call to the Authorization server (AS) to verify and validate the client authorization token.

**History based Capability systems for IoT (HCAP)** proposes a history-based capability system for enforcing permission sequencing constraints in a distributed authorization environment [70]. The authors formally establish the security guarantees of HCAP, and empirically evaluate its performance. In their

work, permission sequencing constraints are encoded as a Security Automaton and embedded in a capability.

## 4.2 Modeling Workflows for Access Control Systems

In the literature, we can find extensive work on the specification and enforcement of workflows; in particular, Bertino et al. [10] studied how to model and enforce workflow authorization constraints such as separation-of-duties in workflows, but using a centralized workflow management system. Workflow based access control is also well-known (Knorr [41] calls them “Dynamic access control”), but this requires a centralized WF enforcement engine. Basin et al. [9] model the business process activities as workflows with a special focus on optimizing the authorizations permissions.

Petri Nets [54] provide a graphical modeling tool used to describe processes performing an orchestrated activity, or in other words, a workflow [1,78]. Petri Nets have the advantage that many properties such as liveness (deadlock-freeness), reachability are easy to verify [19,51,58]. Atluri et al. [5,6] studied how to model workflows using Petri Nets, but did not describe the implementation details. Huang et al. [28] presented a web-enabled workflow management system, and Compagna et al. [15] presented an automatic enforcement of security policies based on workflow-driven web application, but both work presented a centralized architecture. Heckel [27] showed how open Petri Nets are suitable for modeling workflows spanning different enterprises. No existing work discusses about how to handle error conditions during workflow execution, support or integrate practitioner-friendly design and specification tools, enforcing cross-organizational agreements or commitments (i.e., process integrity) and to enforce them to achieve workflow-aware access control with a special focus on modern IoT systems.

Wolter et al. [79] showed a model-driven transformation approach from modeled security goals in the context of business process models into concrete security implementation. Their work focuses on service-oriented architecture. The security annotated business processes are transformed into platform specific security access control or policy languages such as XACML; in particular, they considered security goals such as confidentiality, authentication, and data integrity. Basin et al. [44] presented SecureUML, an UML based modeling language for model-driven security, their approach is based on role-based access control with additional support for specifying authorization constraints. Similarly, Jürjens [38] presented UMLsec (an extension of UML) for secure software development.

Mortensen [50] presented a method for automatic implementation of systems based on Colored Petri Nets (CP-nets or CPN) models. The paper does not describe the algorithms and data structures used to implement the code generation tool, but rather the context of the tool. The paper shows that the method introduced reduces the development time and cost compared with prevailing system development methods where system implementation is accomplished manually by evaluating it on a real-world access control system. We refer to the



concepts presented in this work for generating smart contract code from our Petri Net workflows.

Linhares et al. in [43] presented an empirical evaluation of OMG SysML's to model an industrial automation unit using the open source modeling tool Modelio [48] but not in the context of modeling workflows for access control.

### 4.3 Distributed Accountability and Smart Contracts

To achieve accountability in a system, we need to record all system activities and store them in a database with data properties such as availability, integrity, persistence, and consistency. Distributed database management systems (DDBS) provide data consistency, reliability, and availability (see [53]). In addition, with strong access control systems integrated with a DDBS, we could enforce who can access (read and write) the database. Just integrating access control is not enough to provide accountability in a system i.e., a person with access to the database may insert/update/delete malicious data into the database. Such that the person with access to the DDBS could tamper the data without being noticed by other entities.

The Blockchain technology provides availability, data integrity, non-repudiation (if public-key signatures are used), and persistence properties i.e., once a data block is added by a user and becomes a valid block of the Blockchain, it is impossible to update/delete it without being noticed by others participating in the Blockchain. There are two main types of Blockchain: permissioned and permissionless. A permissioned Blockchain includes an access control layer that can enforce who can read, publish, or approve transactions in a block chain (see IBM Hyperledger [29]). A classic example of permissionless Blockchain is bitcoin [52] i.e., anyone can participate (publish and verify transactions) in the Blockchain. To approve a transaction or a block consisting of many transactions different consensus methods exist such as proof-of-work, but it is not the focus of the paper.

Smart Contracts, introduced in [69], have become popular with the advancements in Blockchain technology. Smart contracts are often written to ensure fairness between participating entities even when one entity may attempt to cheat the other entity in arbitrary ways (see [17]). Smart contracts (SC) deployed in a Blockchain can be seen as arbitrary code expressing one or more business logic, and they are automatically triggered if some preconditions defined in the SC match. A smart contract is executed, the results are verified by the nodes participating in the Blockchain. In [14] and [7] an example of an IoT application using Smart Contracts and Blockchains is presented. The Bitcoin blockchain has a simple stack language to express the rules and conditions for a successful transaction and how new coins are produced and consumed. Ethereum, which has popularized the use of smart contracts, uses a Turing complete language to specify them. In [45], the authors have studied the security of running smart contracts based on Ethereum, and presented some problems in Ethereum's smart contract language solidity; they also show some ways to enhance the operational semantics of Ethereum to make smart contracts less vulnerable.

## 5 Contributions

In this paper, we present a security framework that addresses the following security requirements of constrained IoT environment described in Sect. 2: distributed authorization, requirements elicitation, fine-grained access control, secure software updates, attack escalation resilience, and distributed accountability.

We present a security framework to design, specify, verify, and enforce IoT processes or workflows using Petri Nets. Our framework adapts to error conditions during workflow execution, supports the integration of practitioner-friendly design and specification tools, and enforces cross-organizational agreements or commitments (i.e., the process integrity) as workflows. Thereby, we achieve workflow-aware access control for multi-tenant IoT systems.

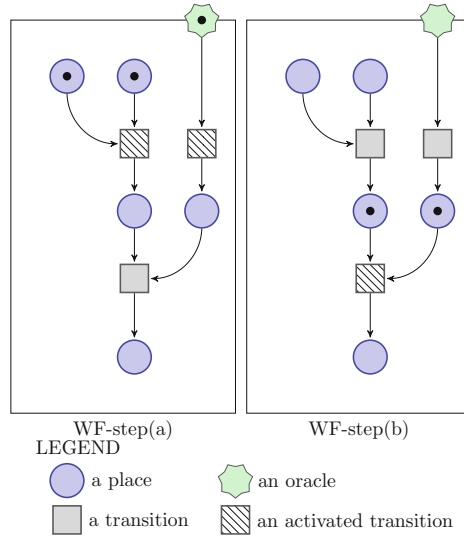
We presented our *Petri Net based Workflow Specification and Enforcement* framework earlier in [39,40]. In this paper, besides summarizing the basic ideas, we extend our framework to support the generation of blockchain-based smart contracts from Petri Nets and to achieve distributed accountability. Furthermore, we demonstrate the applicability of the method by solving three use cases. We also present a high-level guide to implement the framework with practitioner-friendly tool and development systems.

### 5.1 Petri Nets for Workflow Specification

We use Petri Nets and its extensions for specifying workflows. Existing solutions and methods for modeling workflows are described in Sect. 4.2. Petri Nets were chosen to specify workflows for the following advantages and properties. Petri Nets (PN) provide the formal semantics for designing workflows such that PN workflows are amenable to verification of certain properties such as being deadlock free. The expressiveness of Petri Nets and the state-transition model of Petri Nets support all primitives needed to model a workflow process precisely. Extensions of Petri Nets enable us to specify and model complex workflows by solving different workflow issues including concurrent task execution and separation-of-duties between different processes interacting with each other. Petri Nets are a graphical language and as a result, it is simple to design workflows using graphical tools. Also, other practitioner-friendly tools that collect requirements and create activity diagrams can be integrated to generate Petri Net workflows. Petri Nets workflows are technology or platform-independent, therefore, it can be used to implement and integrate platform or technology dependent multi-tenant processes. Overall, it satisfies all requirements that we need to achieve the integrity of the process. Thus, we use Petri nets to enforcing workflow-aware access control (Fig. 3).

In addition to the classical and existing Petri Net extensions, we introduce additional concepts in our Petri Net model:

- Permissions, endorsements, money (crypto coins), signature, or any information that is required for the workflow execution can be represented as *tokens*



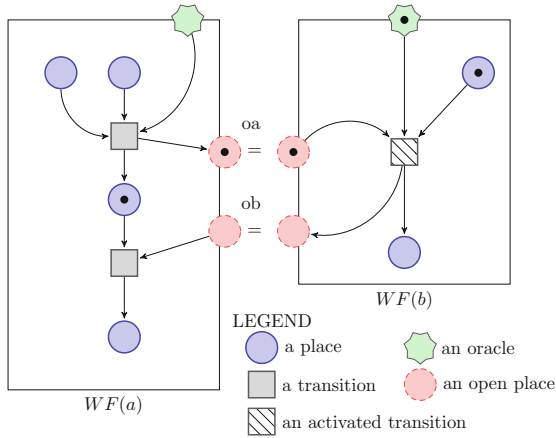
**Fig. 3.** WF-step(a) shows the initial state of a Petri Net workflow specification with an *Oracle*. WF-step(b) shows the state of the workflow after the first two activated transitions have fired.

within the Petri Net. Thanks to CPN, different types of tokens can be used in the same Petri Net to model workflows where entities exchange different information between them. In particular, OAuth tokens are used to enforce access control in a stepwise manner as specified in the workflow.

- An *Oracle* is a type of place, represented in star shape that can receive tokens (as described earlier) from an external source. In classical Petri Nets, places are represented as circles and always receive tokens from a transition. An oracle is drawn on the boundary of a Petri Net to represent that it receives information from an external source. Note: the term oracle is used in different computer science fields including cryptography, blockchain, and smart contracts, etc. Our concept of an Oracle is similar to the Oracles introduced in Ethereum blockchain, i.e., it is used to receive external information into a blockchain smart contract. The difference is: an Oracle in our method need not be a contract that is accessed by other contracts to pull information as described in [8,83]. If blockchain is implemented in an IoT application as a back end distributed database, then an external service can push some information into the blockchain. The published information in the blockchain can be accessed by the Oracle via a predefined URL. Note: it is critical to enforce strict access control that restricts who can publish such information in the blockchain.

Our PN workflows are designed to solve use cases that include interaction with real world IoT devices and actors. In such cases, a workflow should handle error conditions or unexpected situations to an extent. We introduce *dynamic*

*Workflows* to handle such special situation with authorized user decisions and so on. Note: such dynamic workflows must also be verified together with the main workflow (at least during its creation) i.e., without changing the goal or purpose of the main workflow. Protecting the integrity of the processes and allowing dynamic workflows may be competing goals, but, it must be assured that only “authorized” entity can create dynamic workflow and any misuse must be penalized. Therefore, we also need a system to provide accountability of actions performed by the participants executing the workflow.



**Fig. 4.** Two different workflows WF (a) and (b) exchange information using Open Petri Net places ( $oa$  and  $ob$ )

Thanks to Open Petri Nets (see [27]), we apply this concept to create an entry and exit points i.e., Open Petri Net places to exchange information between Petri Net workflows. Exchanging information in the form of tokens simplifies the integrating of two or more PN workflows. Open Petri Nets enable to satisfy one of our goal i.e., interaction between different stakeholders’ processes. For instance, Fig. 4 shows two different workflows WF (a) and WF (b) exchanging tokens via the open place ( $oa$  and  $ob$ ). An open place exists on the boundary of the workflow, and the equivalence (=) sign identifies the entry and exit places between two workflows. The open place ( $oa$ ) is an exit place for WF (a) and entry place for WF (b). The main difference between an oracle and an open place is: an oracle can receive information from external sources whereas, the open places are mainly used to exchange tokens between workflows. Open Petri net places are particularly useful when creating a dynamic workflow to exchange information with the main workflow.

We showed how workflows can be specified using Petri Nets, but we need a mechanism to enforce them on entities executing it. For this purpose, we use small *smart contracts* written in the transitions of Petri Net. A brief introduction of Smart Contracts is presented in the previous Sect. 4.3. For our requirements

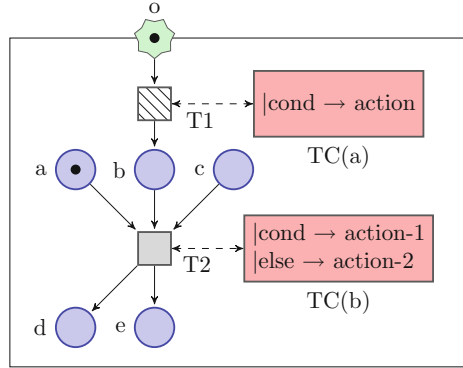


Fig. 5. Petri Net with transition contracts  $t1$  and  $t2$ .

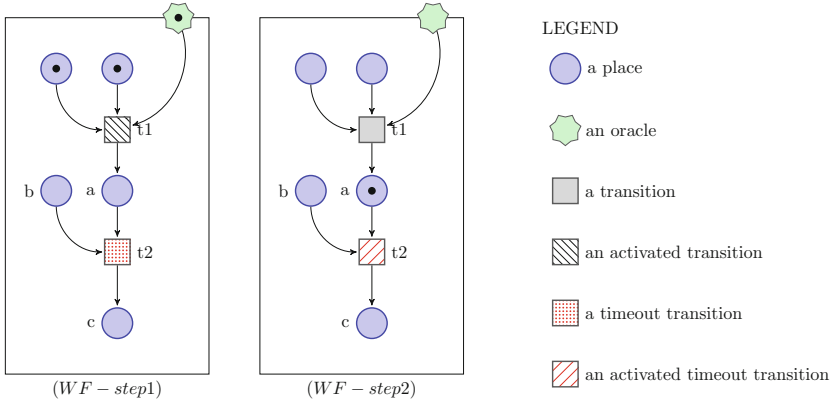
both Bitcoin and Ethereum languages are not suitable. Bitcoin’s stack language is not flexible therefore, we cannot express workflow conditions on it. Ethereum’s solidity language could be vulnerable (see [45]), and we cannot verify such contracts. Therefore, a smart contract language that is flexible to specify conditions and at the same time verifiable is required. To clarify, a complete Petri Net workflow can be seen as a big smart contract comparable to a blockchain based smart contract. The conditions that are written in the transitions of Petri Nets workflows are called *transition contract*.

### 5.2 Transition Contracts

To implement a workflow-driven access control system in Petri Nets, the transitions should be able to verify conditions and evaluate information encoded in the tokens. The conditions written on a single transition using a simple smart contract language is called a *transition contract*. We use a simple guarded command (a conditionally executed statement) language (similar to [18]) to write transition contracts.

Figure 5 shows a simple Petri Net where two transitions ( $T1$  and  $T2$ ) have a pointer to the transition contracts ( $TC(a)$  and  $TC(b)$ ) respectively. Note: smart contracts do not always have to run on blockchain, they can also be implemented between two or more parties without blockchain technology.

The properties (or rules) for each transition can be seen as small smart contracts that restrict the choices of the participants of the workflow for this step, or they impose additional conditions. The combination of a few transition contracts allows us to create *multi-step smart contracts*: say, the first transition creates a token based on some conditions (which may verify authentication or authorization status of participants), and then the second transition produces an OAuth token that can only be used in a subsequent transition in a particular way. The allowed actions, permissions of workflow participants are determined by the Petri Net and the next transition contracts. We use the combination



**Fig. 6.** Timeout transitions in Petri Net workflows

of Petri Nets and transition contracts to specify, enforce sequences of atomic transitions (transactions), and properties that must be satisfied in a workflow.

A transition performs three steps before firing:

- First, it takes tokens from the input places (could be a normal place, open place, or an oracle).
- Next, it verifies the validity, properties of input tokens.
- Finally, it evaluates the conditions described (as guarded commands) in the transition contract and produces the output tokens in output places (could be a normal place, open place, or an oracle).

An output produced by the transition contract can be a token representing information or a workflow for one or more entities. When our workflow-aware access control method is used, compromising one device may not compromise other devices. To explain, let us consider a workflow that is defined by a company for updating Firmware on its IoT devices. Assume that the devices could be triggered to update its Firmware Over-the-Air (OTA) whenever a new Firmware is available. Assume that an attacker compromises one device (how he compromises is not relevant here) and updates a malicious firmware on it. The attacker broadcast the new (malicious) firmware to other legitimate devices such that he could take control over other devices too. This attack is mitigated because the corresponding firmware update workflow as specified by the company must be initiated and a legitimate service person needs to do several steps (for example, provide authorization credentials) before the devices may get into the state where it will accept firmware via the broadcasts channel.

By default, the Petri Net transitions fire when the input places have enough tokens. In many real-world use cases, it is important to have the notion of time required for a task completion. Some tasks in the real-world might require just 10 min, and others might need some hours. If a transition is waiting for a token to arrive in one of its input places, probably it does not want to wait indefinitely.

**Timeout Transitions** are required to stop transitions from waiting indefinitely. Sometimes, a user or an entity may fail to complete a task in a workflow that is expected to be completed within a certain time. That transition may wait forever to get a token in one of its input places. To solve this, we introduce timeout transitions i.e., after a predefined time expires, the timeout transition executes set of predefined timeout conditions (in contrast to the regular conditions) and fires a timeout token in its output place. These timeout tokens may contain or invoke the dynamic workflows. It is important to specify when the timeout timer should start and stop in the timeout transition. If all the input tokens are available before the timeout occurs, then conditions of regular transition contract is executed to produce tokens. Therefore, every timeout transition has two instructions: first, a timeout instruction (timeout contract) is enforced when timeout occurs and some of the input tokens are not available; second, a regular instruction (transition contract) is enforced when all the input tokens are available before timeout.

The example workflow is shown in Fig. 6 explains a simple use case of a timeout transition. Consider that the task  $t2$  must be completed within some time ( $x$  minutes) after the task  $t1$  is completed. When task  $t1$  is completed, then transition  $t1$  produces a token in place (a). A token in place (a) triggers the timer to start in transition  $t2$ . Now, the timeout transition  $t2$  executes one of the three possible cases:

- Case 1: the timer expires after  $x$  minutes (timeout) and place (b) has no token then, the timeout transition contract is executed. A timeout transition contract is similar to a traditional contract but is used only to defined what happens after a timeout.
- Case 2: the timer has not expired and place (b) has a token then, the regular transition contract is executed.
- Case 3: place (b) has already a token before task  $t1$  is completed then, the transition  $t2$  waits until task  $t1$  is completed. When both the input tokens (a and b) are available, the regular transition contract is executed.

### 5.3 Systems Modeling Language (SysML) - Activity Diagram

We investigated how a practitioner (a software developer or engineer) could use our method with existing and familiar tools. It could be complex to design and model a multi-organizational, human interactive process that includes different software and hardware components using Petri Net tools only. Therefore, an existing practitioner-friendly tool is used to model a high-level activity diagram of complex processes and systems. Later, this activity diagram is translated into Petri Net workflows.

Software developers, engineers, and similar practitioners are familiar with UML, since, SysML is an extension to UML, it is easy to understand and learn SysML's notations. The generally accepted method is to refine the specification in a stepwise manner using software engineering tools such as the object

management group (OMG) system modeling language's (SysML) activity diagram presented in [73]. The Object Management Group's OMG SysML [73] is a general-purpose graphical modeling language that supports the specification, design, analysis, and verification of systems that may include different software and hardware components, people, tasks, and other entities. SysML supports the practice of model-based systems engineering (MBSE) and is an extension of Unified Modeling Language (UML) version 2.

SysML is used to develop system solutions to solve technologically challenging problems. One of the challenges is interconnectivity among systems. Therefore, systems can no longer be treated as stand-alone, but behave as part of a larger ecosystem including humans. Such complex systems are known as the system of systems (SoS) [23].

SysML can represent different aspects of systems, components, and other entities [23] such as:

- Structural composition, interconnection, and classification.
- Function-based, message-based, and state-based behavior.
- Constraints on the physical and performance properties.
- Allocations between behavior, structure, and constraints.
- Requirements and their relationship to other requirements, design elements, and test cases.

SysML uses nine diagrams including the *Activity diagram* to represent the relationships between entities in a complex SoS. In particular, the SysML Activity diagram (modified from UML) represents the business/technical process in a defined order i.e., a sequence of actions to be executed based on the availability of their inputs, outputs, and control. Moreover, SysML's activity diagram describes how the actions transform the inputs to outputs. As this is a standardized approach, it is easy for practitioners to use SysML Activity to describe complex systems and processes (both technical and business).

Furthermore, SysML activities are based on token-flow semantics related to Petri-Nets [59]. Thus, SysML provides a semantic foundation for modeling system requirements, and the SysML's activity diagram can be transformed intuitively into a Petri Nets model. The Petri Net tokens hold the values of inputs, outputs, and controls that flow from one action to another. Therefore, it is easy to transfer the SysML activity diagram into Petri Net workflows. For our purposes, we use only the SysML's activity diagram to model the process or workflow.

We use the open source modeling tool known as "Modelio" [48] to draw SysML activity diagrams. Modelio implements all SysML features according to the OMG's specification, and it can also be used to model BPMN and UML diagrams. An example screenshot of the Modelio tool is presented in Fig. 10.

The requirements and SysML activity diagrams lack mathematical semantics to check for inconsistencies, but the SysML activity diagrams can be converted into Petri Nets (for example, colored) and then can be verified using model checking tools [33, 57].



## 5.4 Petri Net Execution Engine

We use the open source Python library called “Snakes” [56] to implement basic Petri Net functions. We extended the Petri Net library to represent different types of tokens, places, and conditions. Furthermore, we present future requirements to extend the standard Petri Net Markup Language (PNML) exchange format. Similarly, there are several Petri Net libraries available for other programming languages such as Java, C, etc.

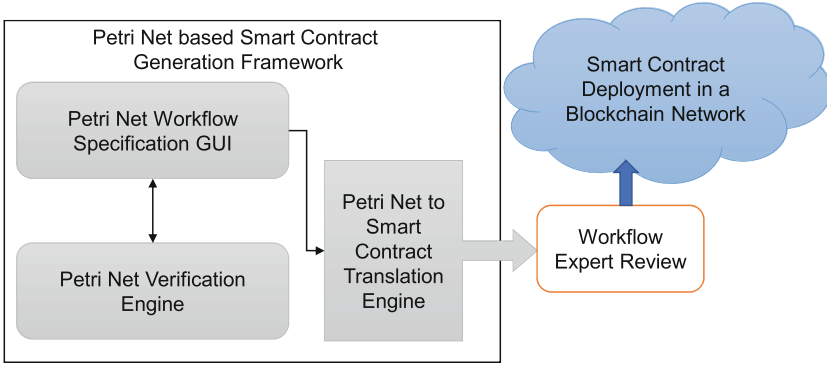
We implemented and evaluated the core part of the above simple use case scenario application using Snakes and other Python library. For doing this, we have extended the Snakes library to realize additional functions and modules that can recognize our new types of tokens, places, and conditions (guarded expressions). The Snakes library is extended to support features such as oracles, open places, timeout transitions, and different types of tokens. The Petri Net workflow evaluates transitions with conditions – for example, validates security tokens from an oracle –, and if necessary, produce tokens in a specific format that will be required for subsequent transitions. The prototype implementation was developed with Ubuntu operating system and Python libraries for implementing REST services, and Petri Net functions. In our current implementation, the transition contracts are expressed with limited features of Snakes library’s arc notations, expressions. Note: extreme caution must be taken to avoid side effects – by calling native Python functions to evaluate input tokens and produce required tokens. Further implementation work is required to realize a smart phone application with an integrated Petri Net execution engine.

We need additional XML tags to represent workflow and its rules i.e., expressions and conditions written in a Transition, token types, open Petri Net places, and how they could interact or interface with dynamic or sub-workflows. We implemented a part of building automation use case presented in Sect. 5.6. Our future work is to extend the standard PNML with additional tags for exchanging Petri Net workflows between different entities and users of any platform.

## 5.5 Petri Net Based Smart Generation Framework

In our next investigation, we looked at various problems in traditional Blockchain based Smart Contracts. We noticed that we could use our method to create *safe and understandable* Smart Contracts (SC). In this section, we introduce a framework that can create Blockchain based smart contracts from Petri Net workflows.

Blockchain-based applications use open source blockchain implementations such as IBM’s Hyperledger Fabric [30], and Ethereum [21]. The corresponding business logic is written using Smart Contracts (SC) in their respective languages i.e., Chaincode [31], and Solidity [20]. Solidity is a Turing-complete computer programming language specifically designed to write Smart Contracts (i.e., to write the business logic). Chaincode is (used synonymous with Smart Contracts) also used to write smart contracts for IBM’s Hyperledger Fabric. But, Chaincode can be written using popular Turing complete languages such as GO, Java, etc.



**Fig. 7.** Components of Blockchain based Smart Contract generation framework

Turing complete languages are known to have problems such as undecidability [17].

With Petri Net workflows (which can be seen as high level Smart Contracts), it is possible to check the properties such as deadlock, etc. Therefore, if a Petri Net is verified (properties are checked), then translating the verified Petri Net into a solidity code is also safe. We present the software prototype architecture below.

The requirements of a smart contract (SC) should be as follows:

- A SC should be easy to understand and write.
- A SC should be amenable to verification of process integrity i.e., it should only allow what it is specified to do.
- If necessary, the SC should support human interaction for example, to approve or reject conditions specified in a SC. Also, the smart contracts should allow recovering from error conditions by allowing dynamic workflows.

The Smart Contract can be a standalone contract, or a part of a big contract consisting of many small SCs. Our proposed Blockchain based SC generation framework consists of three main modules: Petri Net workflow specification GUI, Petri Net verification engine and the Petri Net translation engine into Smart Contract translation engine. Figure 7 shows the main components of the proposed Framework.

In this paper, we provide a brief overview of our proposed framework. In our forthcoming paper, we will describe the specifics of implementation, user interfaces, etc. in detail.

**Petri Net Workflow Specification GUI** provides the user with a simplified GUI interface to the practitioners. The GUI interface consists of places, transitions, and arcs to connect places and transitions. PNML is the standard and recognized format for exchanging Petri Nets.

**Petri Net verification engine** simulates and evaluates whether the Petri Net satisfies the properties such as no deadlocks, etc. We propose to use any standard Petri Net tool or library to implement this functionality.

**Petri Net to Smart Contract Translation Engine** works by mapping places and transitions from the specified Petri Net into blockchain based smart contracts. We are currently working on a prototype that can translate a PN workflow into an Ethereum's based Solidity code - details will be discussed in our forthcoming paper. Nevertheless, the translation engine can be extended to translate the Petri Net smart contracts into other types of blockchain executable smart contract code (executable byte-code = compiled smart contract) such as IBM's Hyperledger fabric's chain code.

Once the Petri Net is translated into a Smart Contract (SC), a workflow expert reviews the generated SC code and published it in the blockchain.

## 5.6 Distributed Accountability and Access Control

Our framework uses a distributed blockchain network for achieve accountability and transparency. A private blockchain is used to set access control restrictions i.e., who can participate in the blockchain. For instance, the user publishes the status of every task when he/she is executing the workflow – i.e., the state of the Petri Net workflow – in the private blockchain. The stakeholders will verify and approve the transactions in the blockchain, and this provides transparency and accountability in an immutable database without assuming a trusted centralized entity.

Distributed access control is achieved by enforcing token validation on the handhelds. Usually, a PN workflow is executed by one or more entities with the help of a handheld or more powerful device capable of executing a Petri Net workflow. We use a trusted application installed on entity's handheld enforcing the validity of the tokens generated and received. Sometimes, the handhelds may also delegate some tasks to a cloud service, for example, to check the blockchain for updates, or, to pull information tokens from an oracle, etc.

Distributed access control is generally used in web technologies. Typically, a browser is a *client* accessing a service hosted on (cloud based) web servers. For instance, in an IoT scenario, the authorization server (AS) evaluates (or delegates evaluation of) the client credentials – the user submits the credentials to AS via a handheld device – and if those client credentials are valid, then the AS presents the client with an authorization token to access the IoT device (or its services).

Our method introduces *Workflow aware access control*, and it is enforced by restricting the users to perform tasks as specified (in an order) in the workflow. Each user uses a handheld device to execute the workflow. The user executing the workflow needs to authenticate to the App (i.e., to prove that the user can execute the workflow). The handheld uses an App that binds a *secret* with a workflow – note: we assume that the client is not able to extract this secret from the handheld or the workflow. The IETF draft “Privacy Enhanced Tokens” a

profile for OAuth 2.0 for constrained devices [16] provides an example of how these proof-of-possession tokens can be generated using the secret. Some actions or tasks that the user needs to perform are enforced on the resource servers. The resource server can verify the tokens without having to communicate with the authorization server.

**Publishing and distributing the Workflow** or Smart contracts through a contract store (i.e., a distributed database) similar to existing smartphone app store or browser add-on/extensions store. The users can download preferred Petri Net workflows and contracts from the Petri Net smart contract store – we use a single contract store based on one distributed database technology. The contract store enforcing a strict process that analyses and validates the contract before publishing it. The distributed database similar to a blockchain can be used to store the Petri Net workflows based smart contracts. We propose to use a single blockchain for publishing contracts. If necessary, access to these contracts may also be restricted by using a permissioned blockchain. For example, IBM’s Hyperledger can be deployed as a permissioned blockchain where entities require permissions to access and publish information in the blockchain.

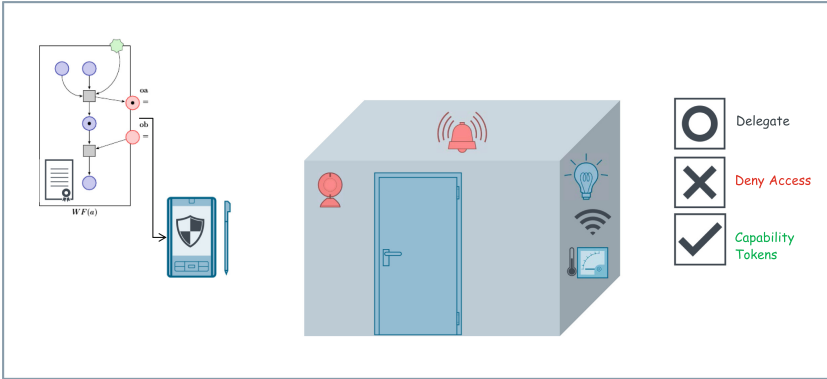
*Verification of Petri Net Workflow:* we use the term verification in terms of verifying the properties of the workflow by simulation, model checking, theorem proving, etc. Verification of Petri Nets must not be confused with validity checking (=validation) of validity tokens as described in Sect. 5.6.

The author of the Petri Net workflow is responsible for verifying the correctness of the workflow’s application or the process itself. The Petri Net (PN) engine assists the authors while creating the Workflow in terms of simulating and verifying Petri Net properties. The PN engine simulates the workflow after saving and provides a comprehensive report to the author about potential problems such as deadlocks, etc. via a notification panel. This feature minimizes the errors while creating the workflow and provides a detailed analysis when the workflow is completed.

*Workflow expert:* the author requests to publish the PN workflow through a process. The objective of the workflow expert is to have “/Quality Control/”. A trusted entity (a workflow expert) checks whether the workflow is designed properly and represents the process defined. Additionally, the workflow expert may use automated tools to check whether the contract follows standard guidelines or not.

Even when the properties of the Petri Nets satisfy, the workflow could perform unnecessary steps not related to the goal of the process. So far, the best process to solve this human problem is to use the *four-eyes* principle [80, 81]. The four-eyes principle means that a certain activity, i.e., a decision, transaction, etc., must be approved by at least two people with expertise. Therefore, before publishing the contract, a workflow expert analyses the process or activity requirements, and verifies whether the designed workflow does the same as described.

**Enforcing AC, validating tokens and conditions by delegation** is a validity checking process that includes checking the validity of an access token, vali-



**Fig. 8.** Building automation - Petri Net workflow enforcement - access denied or granted based on the workflow specification.

dating the signature, integrity checks, etc. By enforcing proper validity checking we enforce access control. Some IoT applications perform this process by delegating validation tasks to trusted (more powerful) devices. We call those devices *handhelds*. Handhelds are more powerful in terms of connectivity, power supply and processing capacity than constrained IoT devices.

Consider a simple use case where a building owner delegates installation or maintenance work to a contracting company. The RFC 7744 [64], provides a summary of authorization problems that emerge during the device life-cycle (commissioning, maintenance, re-commissioning, decommissioning). In addition to the authorization problems, the building owners may wish to ensure that only products with a certain provenance or quality are installed, and that the process complies to standard operating procedures. The building owner may also wish that the contractor obeys other conditions written on a contract. This use case is described in detail in Sect. 6.

The workflow (WF) is created and signed by the building owner. Next, the WF is provided to the contractor. The contractor uses his handheld device as shown in Fig. 8 to execute the WF. The workflow contains a secret material with which the authorization tokens are constructed, please refer to [16] for more details on token construction. We assume that the secret cannot be extracted by the contractor. The building automation devices use the standard ACE-OAuth [32] protocol to validate the token that it receives, and if the tokens are valid, then access to resource is granted otherwise not. If the IoT device receives a request that it is unable to process, it may also delegate this request to an authorization server or other trusted entity. All these three types of response are shown in Fig. 8. The IoT devices can evaluate the validity of the proof-of-possession tokens (i.e., whether this token is constructed based on the shared secret or not) and can respond appropriately to the client device.

**Enforcing Accountability using Blockchain** is possible with our method. When some tasks of a workflow are executed, all information related to that

task including who is executing the task, when it started, when it stopped, and what were the outcomes of the tasks must be logged for future reference. It is important that only authorized persons can write into the log, and no one can tamper with the logging information. For this purpose, we propose to enable the workflow execution application to append relevant logging information into the blockchain.

## 6 Use Cases

### 6.1 Connected Mobility Lab (CML)

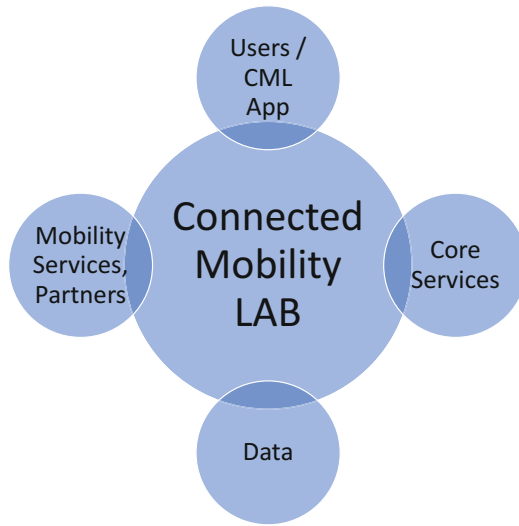
The Connected Mobility Lab (CML) is a public funded project that integrates the services from different stakeholders – such as mobility, financial, and IT services – to provide a comprehensive mobility solution by seamlessly exchanging data and analytics (see [42]). The CML has core services such as IT security, accounting, data management, and identity management that integrate data and processes from different mobility providers. The CML mobile application (CML App) assists users (i.e., travelers) to experience the CML mobility solution with an intuitive user interface. A complete overview of CML is shown in Fig. 9.

The users of CML can be private persons or employees of a company that has a service agreement with the CML. A user may want to use different mobility services to complete one single journey. In CML, different mobility service providers have different specifications and implement “equivalent” tasks differently. For example, validating a ticket or a payment is done differently by each mobility service provider. It is important to guarantee the process integrity of such processes defined by each service provider therefore, we need a workflow-driven access control and a high-level workflow specification language to express those processes.

Consider a simple use case: a user might use a car sharing service from his home to the main train station, then park the car in one of the available parking lots and take a train to reach the final destination. During the trip, the user must obey the rules and conditions specified by that particular mobility provider. The CML mobility service enforces a global workflow specified using our method.

Now, let us consider a more complex business mobility use case scenario: two companies A and B decide to use the mobility services offered by CML to enforce some public funded project-specific travel restrictions on its employees. The use case requirements are:

- Every business travel must be approved by the respective managers of participating companies, and in special cases, the public funding project manager approval is also required.
- Special conditions whenever necessary could be inserted by authorized persons (i.e., the Managers)
- Travelers/Users using CML should be able to recover from error conditions, for instance, if a train or flight is canceled then rebooking should be possible.
- Reimbursement of travel cost after a successful trip should be automated.



**Fig. 9.** The Connected Mobility Lab (CML) offers a comprehensive mobility service by integrating different mobility service providers, partners using its core services and CML App.

- Actions executed by the users/travelers must be recorded in a distributed immutable database for accountability.

As the first step, the requirement engineering experts perform the elicitation process i.e., to collect information from the involved stakeholders. The business mobility process and conditions are defined after consulting with participating companies (A and B) and the public funding project manager. The collected use case requirements are used to create the OMG SysML’s activity diagram. The open-source modeling tool “Modelio”, for example, can be used to create a SysML activity diagram. Figure 10 shows the SysML activity diagram of the above mentioned CML business mobility use case. An employee (e) is able to make a travel request which can be approved or rejected by his manager (mA). In case of a special request, the public funded project manager (mP) must also approve. The CML calendar service provides information about the meeting such as location, time, etc. If the trip is approved, then the employee (i.e., the traveler) may choose the transportation type (for example, public transport, car sharing, and so on) and get the tickets from the CML App. Finally, when the trip comes to an end, the reimbursement process is initiated. Later, the workflow expert transforms the SysML activity diagram into a Petri Net workflow specification as shown in Fig. 11. Finally, the Petri Net workflow is executed by the employee using the CML App.

Let us assume the following:

- The CML App has access to CML core services including the CML calendar service.

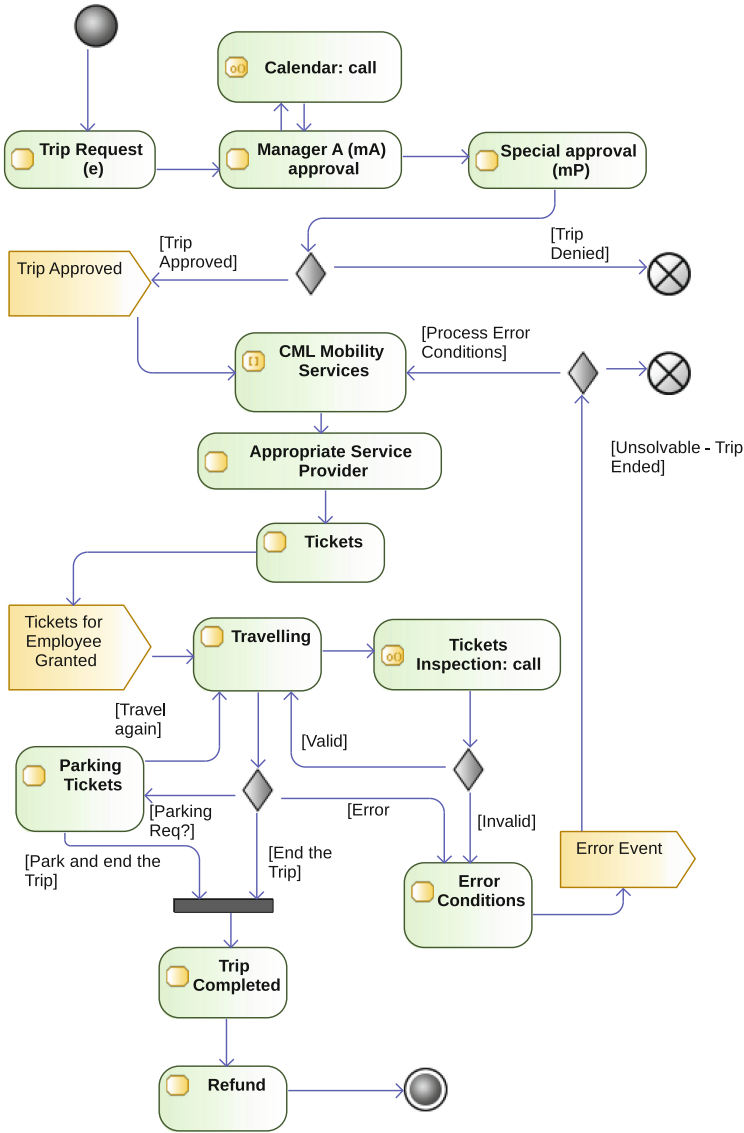
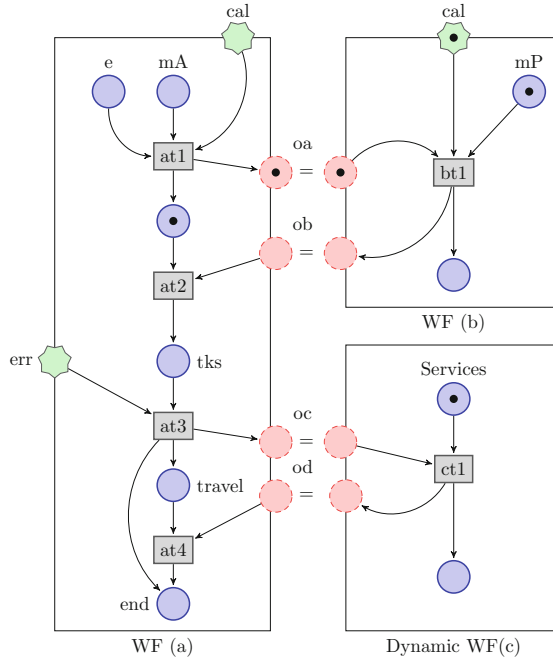


Fig. 10. SysML activity diagram of the CML business mobility use case

- The WF (a), (b) and (c) as shown in Fig. 11 are the resulting Petri Net workflow created by the workflow experts and are available in the central CML repository or the Contract Store. These PN workflows can be accessed by CML App i.e., the users are able to download the required workflows and execute them in the CML App. The sub-workflow (c) is a dynamic workflow and can be invoked to manage unexpected (error) situations. Notice that





**Fig. 11.** Petri Net workflows of the business mobility use case

all three workflows are pre-defined, the workflow experts have created one sub-workflow to manage all unexpected (or error) situations.

- The CML services (such as mobility, parking, etc.) provide tickets, parking lot information, visiting passes for authorized requests similar to an OAuth resource request.

We use the following notations in Fig. 11: employee as  $e$ , manager of company A, B, and public funded project as  $mA$ ,  $mB$ , and  $mP$  respectively, and CML calendar service as  $cal$ . The Petri Net places and transitions are marked with corresponding identifiers such as  $at1$  for WF (a) transition 1 and  $bt1$  for WF (b) transition respectively. Below, we describe step by step process the business mobility use case involving three workflows (a), (b) and (c) as shown in Fig. 11. Assume that the employee ( $e$ ) from company A wants to attend a business meeting organized by the manager ( $mB$ ) in company B.

- The project manager of company B ( $mB$ ) creates a meeting with an identifier ( $mID$ ) in the CML calendar. This identifier is required by the employee ( $e$ ) of company A to initiate the travel request using the CML App.
- The employee ( $e$ ) of Company A makes a travel request using the meeting identifier  $mID$  in his CML App.
- When a travel request is raised, the CML App executes the WF (a) as shown in Fig. 11 i.e., it sends an approval request to his manager ( $mA$ ).

- The manager (mA) approves the request by placing a token in the place *mA* in Fig. 11.
- Next, the Oracle place *cal* performs a GET request with meeting mID to the CML calendar service’s REST interface to retrieve event information such as location, time, etc.
- Assuming that all input tokens are available for the transition (*at1*) of WF (a), transition *at1* evaluates whether the mID, employee email address, and approval from his manager are valid or not. Assume that this is a special trip that requires additional approval from *mP*. Given this special case, the transition (*at1*) executes the transition contract that fires a token in the open place (*oa*) and in the normal output place as shown in Fig. 11.
- Alternatively, if this trip doesn’t require additional approval, then transition *at1* generates a token only in the normal out place and not in the open place (*oa*). The token generated by *at1* has information for next transition *at2* e.g., oAuth token with a secret with which that transition *at2* doesn’t need a token from open place (*ob*). Therefore, the transition *at2* fires only with its normal input place. Similarly, it is possible to execute WF (a) without invoking WF’s (b and c). This scenario describes that was no need for a special approval and there was no error. Note: the tokens generated by each transition contain the information for the next transition i.e., whether the next transition should expect tokens from its respective open places or not.
- Note: we continue the discussion considering that this trip needs a special approval from *mP* as described earlier.
- The CML workflow enforcement engine processes the token from the WF (a) open place (*oa*) and downloads the workflow WF (b) from CML repository to be executed in special cases. The project manager (mP) approves or rejects the trip request. As a result, WF (b) transition contract (*bt1*) evaluates and fires output tokens in the open place (*ob*).
- The token in place (*ob*) provides a secret (similar to an OAuth access-token) required by the transition *at2* to get the tickets from CML mobility services.
- In case of unforeseen circumstances (delay or cancellation of chosen mobility service), the traveler can request an alternative transportation option via CML App. The oracle place (*err*) monitors the information of selected train from the mobility service provider. The transition (*at3*) evaluates the error token, if the traveler wants to end the trip, then it places a token in place (*end*) and places a cancellation/new tickets request in open place (*oc*).
- If the traveler requests alternative tickets, then transition (*at3*) places this request in the open place (*oc*). This token is processed by a dynamically generated WF (c) of the mobility service provider. If the error conditions cannot be solved in an automated fashion, then a human intervention is invoked. Thus, new tickets are delivered via the open place (*od*). Note: Fig. 11 shows the workflow only until this stage, the rest of the workflow steps can be executed with more transitions and places.
- Thanks to the transition contracts in Petri Net based workflows, fine-grained access – such as, temporary access valid during the meeting period – can be granted to enter company B (for example, access to meeting rooms), reim-

bursements can be automated i.e., after a successful trip a waiting time is introduced using timeout transition, if the trip is not successful then a default process is initiated.

- In the end, the organizer of the meeting mB can confirm the attendees through his CML mobile App, therefore the payment transition is activated such that payment to mobility providers, reimbursements to the employees can be handled appropriately.

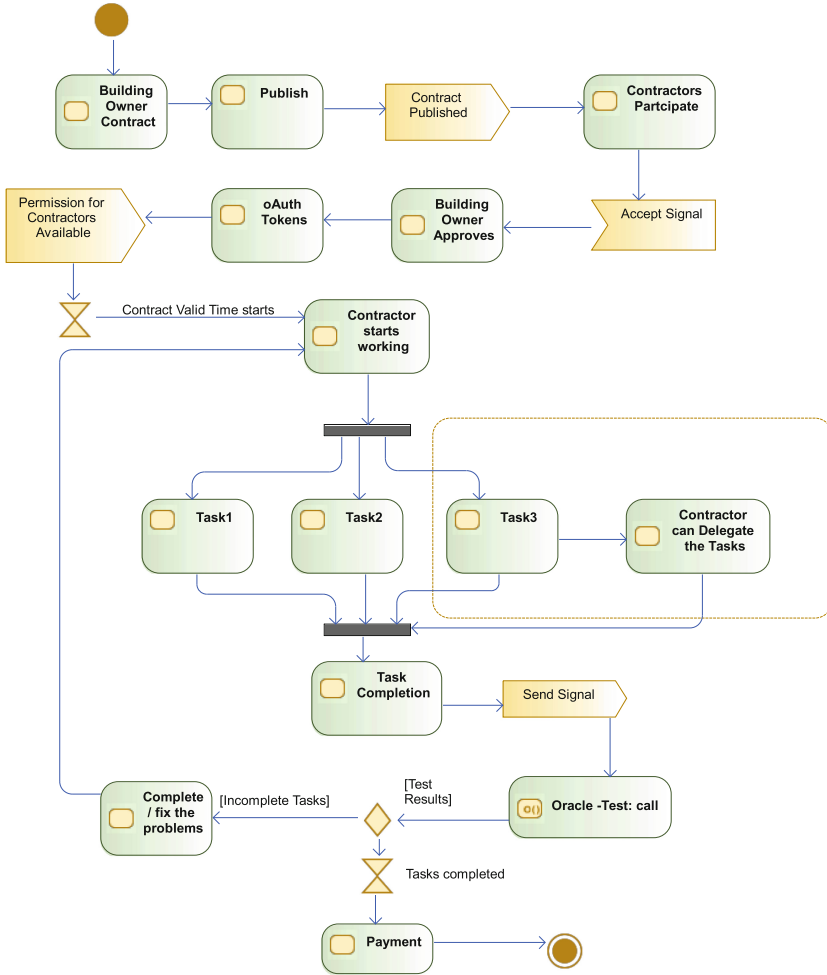
A private blockchain can be used in the CML for accountability. Every Petri Net transitions' input and output tokens are recorded as transactions on the blockchain. This feature provides data immutability and opportunity for future auditing in case of any fraud without a centralized trusted entity. There are several advantages for companies to enforce such business mobility conditions on its employees. The companies could restrict its employees from using transportation service for private purposes. Further, the employees can only use the cost-effective transportation available. By automating this process, the overhead for the employees and its managers is reduced. The companies can satisfy regional policies such as reducing the carbon footprint.

## 6.2 Building Automation

Modern buildings use building automation systems to control lighting, heating, ventilation, and physical safety systems within the building. These building automation systems consist of embedded devices equipped with sensors and actuators, and can collaborate autonomously. For example, the lighting system can adjust the light intensity and color of a room based on the ambient light available in the room; the security system can alert the nearest emergency responders or fire-stations in case of an emergency. In such a scenario, often it is required to perform software-updates, quality-control inspection, fix security patches and upgrade the firmware on the devices. Usually, the building owner delegates the installation or maintenance work to a contracting company. The RFC 7744 [64], provides a summary of authorization problems that emerge during the device life-cycle (commissioning, maintenance, recommissioning, decommissioning). In addition to the authorization problems, the building owners may wish to ensure that only products with a certain provenance or quality are installed, and that the process complies to standard operating procedures.

The building owner also wants that the contractor to obey the conditions agreed in the contract, for instance, the building owner:

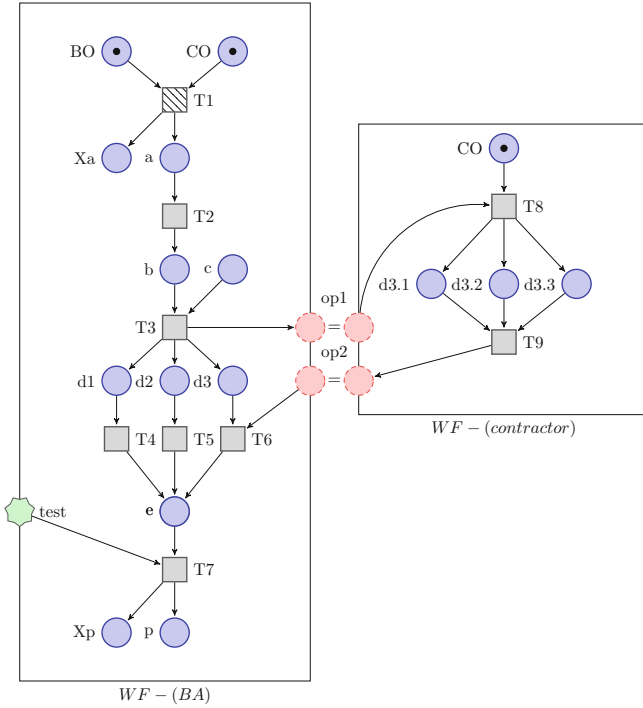
- Wants to track the status of the work in progress remotely.
- Wants to configure the installed devices with custom-rules such that the newly installed devices are interoperable with existing systems and devices.
- Automatically enforce the contract conditions agreed with the contractor. For instance, a penalty if the contractor breaks any agreed condition, or a complete payment if agreed conditions were satisfied.
- Wants to control authorization permissions given to the contractors enforcing fine-grained access control i.e., the least privilege principle.



**Fig. 12.** SysML activity diagram of building automation

First, the requirements elicitation process is conducted to gather the requirements; second, as a result, a SysML activity diagram is created as shown in Fig. 12.

Finally, the building owner with the help of workflow experts has created the Petri Net workflow (*BA*) as shown in Fig. 13. The workflow is published in a private blockchain i.e., in a decentralized contract store as described in Sect. 5.6 after performing strict evaluation. The workflow mobile application certified by the building is downloaded and used by the contractor to execute the workflow. We refer to the similar example described earlier in Fig. 8, where the person executing the workflow gets (security access) tokens for accessing services which are otherwise restricted.



**Fig. 13.** Open PN workflow of building automation

Below, we explain the steps involved in the workflow:

- Once the contract is published, the contractors can evaluate the contract, the workflow, and the requirements to decide whether to participate in the workflow or not. The interested contractor places his decision as a token using the mobile application. The contractor signs the token using his private key, this signed-token is placed in the place (CO).
- Next, let us assume that the building owner selects one of the contractors based on provenance and credibility of the contractor. The building owner uses the mobile application to approve the selected contractor to begin the work. This event creates a token signed by the building owner in the place (BO). The token contains information about the chosen contractor and enables a transition (T1).
- The transition (T1) verifies the tokens in the input places (BO and CO), verify the signature of the token using pre-configured certificates. If both tokens are valid, then T1’s transition contract creates an OAuth-token in place (a). This token in place (a) permits the contractor to access the devices for maintenance purposes as defined in the next steps of the workflow. As expected, only one contractor can be selected i.e., the T1 places the input tokens of contractors not chosen in the output place (Xa).

- A valid token in place (a) triggers Transition (T2). T2 verifies token in place (a). Now, the selected contractor once again confirms by placing a signed token in place (c). By doing this he/she binds to the agreed conditions and begins the work. The transition (T3) requires a token from the selected contractor and creates proof-of-possession OAuth ACE tokens in places (d1, d2, and d3). Tokens in d1, d2, and d3 gives the contractor access to three different tasks/services in the devices, for example, d1 token to perform tests, d2 token to perform firmware updates, and d3 token to configure.
- The Contractor may also delegate one or more tasks to his employees or subordinates by creating his own dynamic workflow. The tokens of completed tasks are exchanged to the main workflow using open places pointing to the transitions expecting the task completion tokens. For example, in the Fig. 13 task d3's token is expected by transition T6. Task d3 is split into three sub-tasks (d3.1, d3.2, and d3.3) and delegated to the subordinates via open place (op1). After completion, the resulting tokens are given as input tokens to the transition T6 via the open place (op2).
- Once all the tasks are completed, the transitions (T4, T5 and T6) evaluate the input tokens and place three tokens in the place (e). The oracle place (test) has a valid token if the automated tests results are successful. If the places (e, and test) have valid tokens then transition (T7) can trigger the payment for the contractor in place (p). If tests were not successful, a token in place (Xp) is placed and requires external evaluation.

The contracting company might want to enforce specific conditions by creating dynamic workflows on their employees (to handle special or error conditions). The open places introduced in the main contract must not change the main objective of the workflow. To enable this feature, the building owner may allow some transitions (for example, T3 and T6 in Fig. 13) to allow open places from authorized participants. Figure 13 shows the owner of the task (the contractor) can create dynamic workflows for other entities to complete a task or resource that he owns. In this way, we have realized a distributed workflow management system. This use case shows how we can execute and enforce a workflow in a distributed setting.

### 6.3 Car Sharing

Car sharing services such as DriveNow and Car2Go are popular for short-term car rental. For instance, DriveNow and Car2Go have their own workflow to rent a car, finish the rental, and for payment. A customer must first register to the service with his/her driving license, proof of address, payment method (credit card or bank account details), and personal identity. The customer is provided with either a card, login credentials, or other means of authentication credentials to access the service. Most car sharing services provide a web-service and mobile application.

Our aim was to apply our framework and methods to solve a real use case. Therefore, as an example, we chose the car hire process of DriveNow and applied

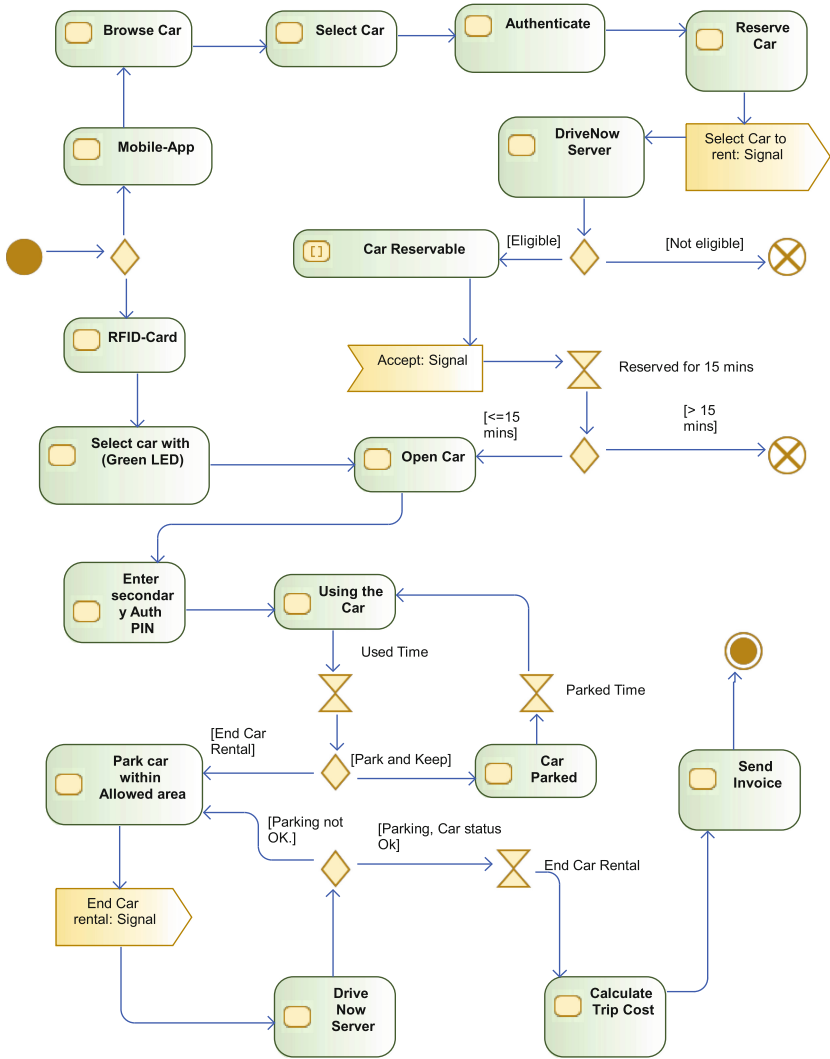
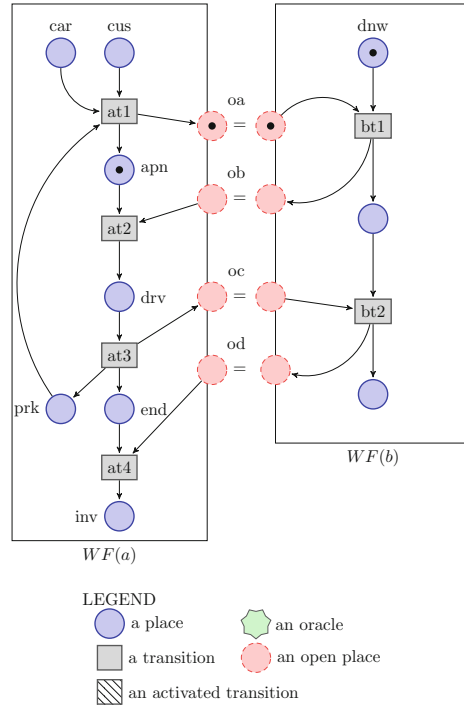


Fig. 14. SysML activity diagram of DriveNow car sharing platform

our methods to solve it. Note: the rental process described in this use case is only based on our experience, and this process can be updated (or outdated) anytime by the service provider and might not be valid anymore. A SysML activity diagram describing the rental process of DriveNow is shown in Fig. 14.

We translate the SysML activity diagram of DriveNow car hire process into our Petri Network workflows as shown in Fig. 15.

The customer chooses one of the two available methods to rent a car: (a) using the DriveNow card; (b) using the DriveNow mobile application (App).



**Fig. 15.** Petri Net workflow specification of DriveNow use case

- Method(a): the customer finds a DriveNow car in the street with Green LED blinking on car’s windshield. Green LED means the DriveNow car is available and Red LED means it is not available. Now, the customer can use his DriveNow card to open the car.
- Method(b): the customer can plan ahead, reserve a DriveNow car for 15 min using his DriveNow mobile application (App). First, an available car is selected in the App. Second, the customer must use his login credentials to authenticate and reserve the car for 15 min. The customer should open the reserved car within 15 min otherwise the reservation is canceled.
- Step1: Assume that the customer used one of the two available methods (a or b) as described above to get inside the car. This action is depicted as placing a token at place  $cus$  by the customer in Fig. 15. The Transition ( $at1$ ) process this token in place ( $cus$ ), availability of the car (with inbuilt car information) in place ( $car$ ) and opens the door.
- Step2: The customer must enter his secondary authentication PIN in place ( $apn$ ) using the car’s touch interface in the dashboard. The transition ( $at2$ ) checks the PIN entered via the information available from DriveNow server. If the PIN is valid, transition  $at2$  places the token in place ( $drv$ ). Now, the customer can start and use the car.



- Dashboard information for the driver: if the car leaves the DriveNow business area of city it belongs to, then a warning notification appears on the dashboard i.e., it is not possible to end the rental outside the business area – *park and keep* option is allowed, but with probably different charges. DriveNow is also offering rental packages for hours and days and with this contract business area restriction does not apply.
- Step3: the customer can park and keep the car or end the car rental via the App or car’s dashboard. This decision is recorded and processed by the transition (at3).
  - Step3.1: if the customer parks and keeps the car using *park and keep* option, then he can re-enter the car using his App or DriveNow card using the same steps described in step1 to continue.
  - Step3.2: Note: this step is not available within the described car sharing service. We included this to show that our method can handle error conditions. Assume an error condition such as breakdown or malfunction, the transition at3 allows the customer to report it via the App and that can be processed by DriveNow to allow new business logic that can help the customer to reach his destination via other methods, etc.
- Step4: the customer can end car rental if the car is in the business area (geofenced area). If the conditions are valid, then transition (at4) allows to end the rental and places a token in place (inv). The trip invoice is calculated and sent to his email based on his usage. If automatic payment is enabled, then the amount is billed to his credit card.

Figure 15 shows the Petri Net workflow of DriveNow use case. The interaction between the customer and a DriveNow car is described on WF (a), and WF (b) describes the DriveNow (DN) server processing the car sharing requests (i.e., in the form of tokens) from WF (a) via open Petri Net places. As you can see, when using a particular service the customer must download an application provided by that particular service provider. If our method is applied, a common workflow application can be used to rent cars from different service providers - only the car hire process and their specific workflows must be modeled and provided to our workflow application.

## 7 A High-Level Summary and Implementation Guide

So far, we presented our method and solved some specific use cases using our framework. Now, we want to summarize the ideas, present a simple guide for solving any generic use case, and a high-level guide to implementation.

First, a use case that one wants to implement must be identified. Next, the process including technical and business details is discussed and finalized with relevant stakeholders. Once the process is defined, an engineer uses a SysML activity diagram using tools such as Modelio to describe the process. Later, this activity diagram can be exported to a Petri Net workflow. Next, a Petri Net simulator is used to check properties of the exported Petri Net Workflow such as

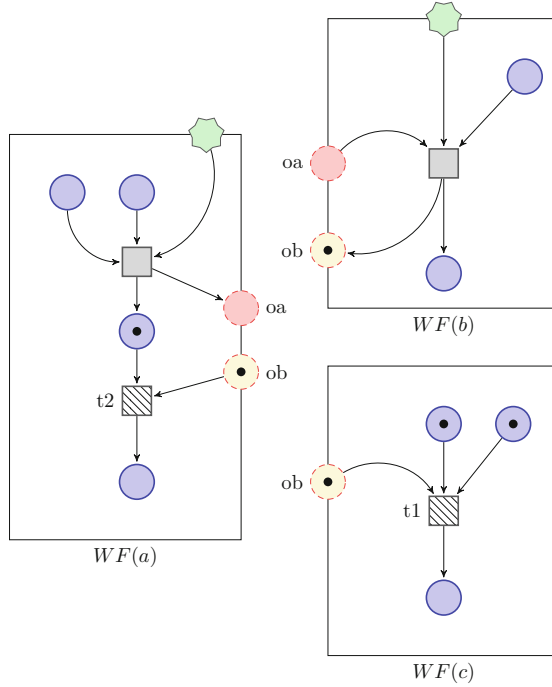
deadlocks, etc. Then a Petri Net library such as Snakes can be used for implementing Petri Net functionalities into the existing software application. After this, a workflow expert should check if the Petri Net workflow and the transition contract conditions represent the process defined. Now, this verified PN workflow is published in a distributed database with appropriate access control such that only authorized persons can access the PN workflow. Now, an entity that needs to execute the process should download the corresponding PN workflow and the workflow execution application. With our framework, we provide workflow-aware access control by enforcing the process integrity. Additionally, for blockchain based solutions, we presented a framework to translate verified Petri Net workflow into Blockchain based smart contracts.

To explain a simple implementation guide, consider a simple use case that includes one or more stakeholders. All stakeholders provide their services as Representational State Transfer (REST) based web services. The workflows are created by practitioners (for example, engineers) and are verified by workflow experts, and finally, approved by the stakeholders. The approved workflow is available within a centralized (or a distributed) repository. A participant can download the application (Trusted App) in his handheld and the required workflow from the repository, and then he may start executing the workflow. The APP provides the communication interface with the core services – standard security protocols are used to protect the communication channel. How participants authenticate with the back end is out of scope. A secret material is used to verify the validity tokens and to create tokens to represent the entity that is executing the workflow, how this secret material is delivered to the App is out of scope. The enforcement of the Petri Net tokens is implemented in the App. We suggest using the ACE-OAuth based protocol to create such tokens. These proof-of-possession tokens are used by the client to prove to the resource server that the client is the valid entity to access the resources. The workflows are executed i.e., transitions and tokens are precisely processed in the Trusted Execution Environment (TEE) of the handheld. We assume that the participants are not able to extract or modify any secrets from the workflow. The Snakes Python library is used in the App to execute the Petri Nets workflows. Weber et al. [75] introduced Petri Net Markup Language (PNML) which is based on XML, and in this work, we propose to use PNML to express Petri Net workflows.

## 8 Limitations of Our Approach

### 8.1 Error Free Petri Net Workflow vs Design Flaw in the Process

A Petri Net simulator cannot detect a design problem or a flaw in the process itself. For example, assume that a Petri Net workflow is developed to protect some assets in the building. For instance, if the process does not include closing the secure door after accessing the assets, so this is a major design flaw and cannot be detected by the Petri Net. Therefore, designing workflows using Petri Nets does not guarantee error freeness.



**Fig. 16.** The token in open place  $ob$  of  $WF(b)$  can be consumed either by  $t2$  of  $WF(a)$  or  $t1$  of  $WF(c)$ . This prevents either  $WFs(a)$  or  $c)$  to proceed forward.

The four-eyes principle is used to verify any process designed using another expert in the same field. This approach could find significant obvious problems in the process itself. The process can be improved without errors when it is reviewed by several experts. Once the process is designed without obvious design flaws, then it can be evaluated with Petri Nets simulators for properties such as deadlocks, etc.

### 8.2 Open Petri Nets and Deadlocks by Merging Different Processes

Consider three small individual processes  $a$ ,  $b$ , and  $c$  designed and verified for Petri Nets properties. We can use open Petri Nets to create interfaces between those three different processes  $a$ ,  $b$  and  $c$ . This enables us to create a main workflow consisting of two or three sub-workflows.

It is possible to have deadlocks when merging two or more sub-workflows without proper validation. For example, when we combine only two of them ( $a$  and  $b$ ) or ( $b$  or  $c$ ) then there may not be any deadlock but, when all three workflows ( $a$ ,  $b$ , and  $c$ ) are combined then there could be a deadlock. Figure 16 shows such an example with three  $WFs$   $a$ ,  $b$ , and  $c$  where the  $WF(b)$  is in a state after producing a token in its open place  $ob$ , then the token in  $ob$  can be either consumed by  $WF(a)$  via transition  $t2$  or  $WF(c)$  via transition  $t1$ . If one

WF consumes the token in *ob* then the other WF cannot proceed. Therefore, it is important to validate and verify the properties before merging sub-workflows with the main workflow.

## 9 Conclusion

In this paper, we presented the Petri Net based workflow specification and enforcement framework and extended it to support emergent IoT applications. We demonstrated how the method can protect the integrity of processes defined as Petri Nets, and how it can be applied to solve different use cases.

We showed that access control permissions should be granted to entities in the form: ‘*You are allowed to execute this task in this workflow*’ instead of ‘You are authorized to access this service during this period of time’. The permission to execute a step in a workflow depends on having executed the required previous steps (i.e., based on the history).

We extended the framework to integrate with practitioner-friendly tools, to support the generation of blockchain based smart contracts from Petri Nets, and to achieve distributed accountability. We showed how the workflow specified in Petri Nets may handle error situations by exchanging information via open Petri Net places. Finally, we demonstrated that our framework provides workflow-aware access control and also enforces the integrity of processes specified as Petri Nets.

**Acknowledgements.** We thank Professor Jonathan P. Bowen for his suggestions and reviewing this article.

## References

1. van der Aalst, W.M.P.: Verification of workflow nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997). [https://doi.org/10.1007/3-540-63139-9\\_48](https://doi.org/10.1007/3-540-63139-9_48)
2. van der Aalst, W.M.P.: Putting high-level Petri nets to work in industry. *Comput. Ind.* **25**(1), 45–54 (1994). [https://doi.org/10.1016/0166-3615\(94\)90031-0](https://doi.org/10.1016/0166-3615(94)90031-0)
3. AIOTI: The Alliance for the Internet of Things Innovation (2018). <https://aioti.eu/>. Accessed Dec 2018
4. Antonakakis, M., et al.: Understanding the Mirai Botnet. In: 26th USENIX Security Symposium, pp. 1092–1110 (2017). <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis>
5. Atluri, V., Huang, W.-K.: An authorization model for workflows. In: Bertino, E., Kurth, H., Martella, G., Montolivo, E. (eds.) ESORICS 1996. LNCS, vol. 1146, pp. 44–64. Springer, Heidelberg (1996). [https://doi.org/10.1007/3-540-61770-1\\_27](https://doi.org/10.1007/3-540-61770-1_27)
6. Atluri, V., Huang, W.: A Petri net based safety analysis of workflow authorization models. *J. Comput. Secur.* **8**(2/3), 209–240 (2000). <http://content.iospress.com/articles/journal-of-computer-security/jcs113>
7. Bahga, A., Madiseti, V.K.: Blockchain platform for industrial internet of things. *J. Softw. Eng. Appl.* **9**, 533–546 (2016). <https://doi.org/10.4236/jsea.2016.910036>

8. Bartoletti, M., Pompianu, L.: An empirical analysis of smart contracts: platforms, applications, and design patterns. In: Brenner, M., et al. (eds.) FC 2017. LNCS, vol. 10323, pp. 494–509. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-70278-0\\_31](https://doi.org/10.1007/978-3-319-70278-0_31)
9. Basin, D., Burri, S.J., Karjoth, G.: Optimal workflow-aware authorizations. In: ACM Symposium on Access Control Models and Technologies (SACMAT 2012), pp. 93–102 (2012). <https://doi.org/10.1145/2295136.2295154>
10. Bertino, E., Ferrari, E., Atluri, V.: The specification and enforcement of authorization constraints in workflow management systems. *ACM Trans. Inf. Syst. Secur.* **2**(1), 65–104 (1999). <https://doi.org/10.1145/300830.300837>
11. Bishop, M.: *Computer Security: Art and Science*. Addison-Wesley, Boston (2002). <https://doi.org/10.1093/toxsci/kft059>. <https://books.google.de/books?id=b4gcsWEACAAJ>
12. Bormann, C., Ersue, M., Keranen, A.: Terminology for constrained-node networks. Technical report, IETF, May 2014. <https://doi.org/10.17487/rfc7228>
13. Castelluccia, C., Francillon, A., Perito, D., Soriente, C.: On the difficulty of software-based attestation of embedded devices. In: Proceedings of the 16th ACM conference on Computer and communications security - CCS 2009, p. 400. ACM Press, New York (2009). <https://doi.org/10.1145/1653662.1653711>
14. Christidis, K., Devetsikiotis, M.: Blockchains and smart contracts for the internet of things. *IEEE Access* **4**, 2292–2303 (2016). <https://doi.org/10.1109/ACCESS.2016.2566339>. <http://ieeexplore.ieee.org/document/7467408/>
15. Compagna, L., dos Santos, D.R., Ponta, S.E., Ranise, S.: Aegis: automatic enforcement of security policies in workflow-driven web applications. In: Proceedings of ACM on Conference on Data and Application Security and Privacy - CODASPY 2017, pp. 321–328 (2017). <https://doi.org/10.1145/3029806.3029813>
16. Cuellar, J., Kasinathan, P., Calvo, D.: Privacy-enhanced-tokens (PAT) profile for ACE. Technical report, IETF (2018). <https://datatracker.ietf.org/doc/draft-cuellar-ace-pat-priv-enhanced-authz-tokens/>
17. Delmolino, K., Arnett, M., Kosba, A.E., Miller, A., Shi, E.: Step by step towards creating a safe smart contract: lessons and insights from a cryptocurrency lab. *IACR Cryptology ePrint Archive* 2015, 460 (2015). [https://doi.org/10.1007/978-3-662-53357-4\\_6](https://doi.org/10.1007/978-3-662-53357-4_6). <https://eprint.iacr.org/2015/460.pdf>
18. Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM* **18**(8), 453–457 (1975). <https://doi.org/10.1145/360933.360975>
19. Esparza, J.: Decidability and complexity of Petri net problems—an introduction. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1491, pp. 374–428. Springer, Heidelberg (1998). [https://doi.org/10.1007/3-540-65306-6\\_20](https://doi.org/10.1007/3-540-65306-6_20)
20. Ethereum: Solidity—Solidity (2018). <https://solidity.readthedocs.io/en/develop/>. Accessed Aug 2018
21. Ethereum: What Are Smart Contracts - EthereumWiki (2018). <http://www.ethereumwiki.com/ethereum-wiki/smart-contracts/>. Accessed Mar 2018
22. European Union (EU): EU GDPR Information Portal (2018). <https://www.eugdpr.org/>. Accessed July 2018
23. Friedenthal, S., Moore, A., Steiner, R.: *A Practical Guide to SysML*, 3rd edn. Morgan Kaufmann, San Francisco (2008). <https://doi.org/10.1016/B978-0-12-374379-4.X0001-X>

24. Gerdes, S., Bergmann, O., Bormann, C., Selander, G., Seitz, L.: Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE) (2018). <https://tools.ietf.org/html/draft-ietf-ace-dtls-authorize-03>. Accessed Mar 2018
25. Hardt, D.: The OAuth 2.0 Authorization Framework (2012). <https://tools.ietf.org/html/rfc6749>. Accessed Dec 2017
26. Harney, H., Muckenhirn, C.: Group Key Management Protocol (GKMP) Specification, July 1997. <https://doi.org/10.17487/rfc2093>
27. Heckel, R.: Open Petri nets as semantic model for workflow integration. In: Ehrig, H., Reisig, W., Rozenberg, G., Weber, H. (eds.) Petri Net Technology for Communication-Based Systems. LNCS, vol. 2472, pp. 281–294. Springer, Heidelberg (2003). [https://doi.org/10.1007/978-3-540-40022-6\\_14](https://doi.org/10.1007/978-3-540-40022-6_14)
28. Huang, W.K., Atluri, V.: SecureFlow: a secure web-enabled workflow management system. In: Proceedings of the Fourth ACM Workshop on Role-Based Access Control - RBAC 1999, pp. 83–94 (1999). <https://doi.org/10.1145/319171.319179>
29. IBM: Energy-Blockchain Labs and IBM Create Carbon Credit Management Platform Using Hyperledger Fabric on the IBM Cloud, pp. 2–3. IBM Press Release (2017). <https://www-03.ibm.com/press/us/en/pressrelease/51839.wss>
30. IBM: Hyperledger Fabric – Hyperledger (2018). <https://www.hyperledger.org/projects/fabric>. Accessed Aug 2018
31. IBM: Hyperledger-Smart Contract Language – Chaincode (2018). <https://hyperledger-fabric.readthedocs.io/en/release-1.2/blockchain.html>. Accessed Aug 2018
32. IETF ACE Working Group: Authentication and Authorization for Constrained Environments (ACE) (2017). <https://datatracker.ietf.org/doc/draft-ietf-ace-oauth-authz/>. Accessed Dec 2017
33. Jamal, M., Zafar, N.A.: Transformation of activity diagram into coloured Petri nets using weighted directed graph. In: 2016 International Conference on Frontiers of Information Technology (FIT), pp. 181–186. IEEE, December 2016. <https://doi.org/10.1109/FIT.2016.041>. <http://ieeexplore.ieee.org/document/7866750/>
34. Jensen, K.: Coloured Petri nets. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) Petri Nets: Central Models and Their Properties. LNCS, vol. 254, pp. 248–299. Springer, Heidelberg (1987). <https://doi.org/10.1007/BFb0046842>
35. Jensen, K.: Coloured Petri nets: a high level language for system design and analysis. In: Rozenberg, G. (ed.) ICATPN 1989. LNCS, vol. 483, pp. 342–416. Springer, Heidelberg (1991). [https://doi.org/10.1007/3-540-53863-1\\_31](https://doi.org/10.1007/3-540-53863-1_31)
36. Jensen, K.: Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use. Monographs in Theoretical Computer Science. An EATCS Series, vol. 1, 2nd edn. Springer, Heidelberg (1996). <https://doi.org/10.1007/978-3-662-03241-1>
37. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri nets and CPN tools for modelling and validation of concurrent systems. STTT **9**(3–4), 213–254 (2007). <https://doi.org/10.1007/s10009-007-0038-x>
38. Jürjens, J.: UMLsec: extending UML for secure systems development. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 412–425. Springer, Heidelberg (2002). [https://doi.org/10.1007/3-540-45800-X\\_32](https://doi.org/10.1007/3-540-45800-X_32)
39. Kasinathan, P., Cuéllar, J.: Securing the integrity of workflows in IoT. In: Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks, EWSN 2018, Madrid, Spain, 14–16 February 2018, pp. 252–257 (2018). <http://dl.acm.org/citation.cfm?id=3234908>

40. Kasinathan, P., Cuellar, J.: Workflow-aware security of integrated mobility services. In: Lopez, J., Zhou, J., Soriano, M. (eds.) ESORICS 2018. LNCS, vol. 11099, pp. 3–19. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-98989-1\\_1](https://doi.org/10.1007/978-3-319-98989-1_1)
41. Knorr, K.: Dynamic access control through Petri net workflows. In: 16th Annual Computer Security Applications Conference (ACSAC 2000), New Orleans, Louisiana, USA, 11–15 December 2000, pp. 159–167 (2000). <https://doi.org/10.1109/ACSAC.2000.898869>
42. Krebs, B., BMW: connected mobility lab – center digitization.bayern (2017). <https://zentrum-digitalisierung.bayern/connected-mobility-lab/>. Accessed Oct 2018
43. Linhares, M.V., da Silva, A.J., de Oliveira, R.S.: Empirical evaluation of SysML through the modeling of an industrial automation unit. In: 2006 IEEE Conference on Emerging Technologies and Factory Automation, pp. 145–152. IEEE, September 2006. <https://doi.org/10.1109/ETFA.2006.355190>. <http://ieeexplore.ieee.org/document/4178305/>
44. Lodderstedt, T., Basin, D., Doser, J.: SecureUML: a UML-based modeling language for model-driven security. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 426–441. Springer, Heidelberg (2002). [https://doi.org/10.1007/3-540-45800-X\\_33](https://doi.org/10.1007/3-540-45800-X_33)
45. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS 2016, pp. 254–269. ACM Press, New York (2016). <https://doi.org/10.1145/2976749.2978309>
46. Merlin, P.M., Farber, D.J.: Recoverability of communication protocols-implications of a theoretical study. IEEE Trans. Commun. (1976). <https://doi.org/10.1109/TCOM.1976.1093424>
47. Miessler, D., Smith, C., Haddix, J.: OWASP Internet of Things Top Ten Project (2014). Accessed Dec 2017
48. Modelio – Open Source Tool: Modelio – the open source modeling tool. <https://www.modelio.org/>. Accessed Aug 2018
49. Morimoto, S.: A survey of formal verification for business process modeling. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008. LNCS, vol. 5102, pp. 514–522. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-69387-1\\_58](https://doi.org/10.1007/978-3-540-69387-1_58)
50. Mortensen, K.H.: Automatic code generation method based on coloured Petri net models applied on an access control system. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 367–386. Springer, Heidelberg (2000). [https://doi.org/10.1007/3-540-44988-4\\_21](https://doi.org/10.1007/3-540-44988-4_21)
51. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989). <https://doi.org/10.1109/5.24143>. <http://ieeexplore.ieee.org/document/24143/>
52. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). <https://bitcoin.org/bitcoin.pdf>. Accessed Oct 2018
53. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, 3rd edn. Springer, New York (2011). <https://doi.org/10.1007/978-1-4419-8834-8>
54. Petri, C.A.: Communication with automata (1966). <http://edoc.sub.uni-hamburg.de/informatik/volltexte/2010/155/>
55. Pohl, K.: Requirements Engineering: An Overview. RWTH, Fachgruppe Informatik, Aachen (1996). <ftp://ftp8.de.freebsd.org/pub/packages/CREWS/CREWS-96-02.pdf>

56. Pommereau, F.: SNAKES: a flexible high-level Petri nets library (tool paper). In: Devillers, R., Valmari, A. (eds.) PETRI NETS 2015. LNCS, vol. 9115, pp. 254–265. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-19488-2\\_13](https://doi.org/10.1007/978-3-319-19488-2_13)
57. Rahim, M., Boukala-Ioualalen, M., Hammad, A.: Petri nets based approach for modular verification of SysML requirements on activity diagrams. In: Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE), Tunis, Tunisia, 23–24 June 2014, pp. 233–248 (2014). <http://ceur-ws.org/Vol-1160/paper14.pdf>
58. Reisig, W.: Petri Nets: An Introduction. EATCS Monographs on Theoretical Computer Science, vol. 4. Springer, Heidelberg (1985). <https://doi.org/10.1007/978-3-642-69968-9>
59. Reisig, W.: A Primer in Petri Net Design. Springer Compass International. Springer, Heidelberg (1992). <https://doi.org/10.1007/978-3-642-75329-9>
60. Reisig, W.: Understanding Petri Nets – Modeling Techniques, Analysis Methods, Case Studies. Springer, Heidelberg (2013). <https://doi.org/10.1007/978-3-642-33278-4>
61. Sadeghi, A.R., Wachsmann, C., Waidner, M.: Security and privacy challenges in industrial internet of things. In: Proceedings of the 52nd Annual Design Automation Conference on - DAC 2015, pp. 1–6. ACM Press, New York (2015). <https://doi.org/10.1145/2744769.2747942>
62. Sandhu, R.S., Samarati, P.: Access control: principles and practice. IEEE Commun. Mag. **32**(9), 40–48 (1994). <https://doi.org/10.1109/35.312842>. <http://ieeexplore.ieee.org/document/312842/>
63. Schaller, R.: Moore’s law: present and future. IEEE Spectr. **34**(6), 52–59 (1997). <https://doi.org/10.1109/6.591665>
64. Seitz, L., Gerdes, S., Selander, G., Mani, M., Kumar, S.: Use cases for authentication and authorization in constrained environments (2016). ISSN 2070-1721. <https://tools.ietf.org/html/rfc7744>
65. Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., Tschofenig, H.: Authentication and authorization for constrained environments (ACE) using the OAuth 2.0 framework (ACE-OAuth). Technical report, IETF (2018)
66. Sicari, S., Rizzardi, A., Grieco, L., Coen-Porisini, A.: Security, privacy and trust in internet of things: the road ahead. Comput. Netw. **76**, 146–164 (2015). <https://doi.org/10.1016/J.COMNET.2014.11.008>. <https://www.sciencedirect.com/science/article/pii/S1389128614003971>
67. van der Stok, P., Kampanakis, P., Kumar, S., Richardson, M., Furuhez, M., Raza, S.: EST over secure CoAP (EST-coaps). Technical report, IETF (2018). <https://datatracker.ietf.org/doc/draft-ietf-ace-coap-est/>
68. Sundmaeker, H., Guillemin, P., Friess, P., Woelfflé, S. (eds.): Vision and Challenges for Realising the Internet of Things. Publications Office of the European Union, Luxembourg (2010). <https://doi.org/10.2759/26127>
69. Szabo, N.: Smart contracts: building blocks for digital markets, 1996. EXTROPY: The Journal of Transhumanist Thought (2001). <http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.2.html>
70. Tandon, L., Fong, P.W.L., Safavi-Naini, R.: HCAP: a history-based capability system for IoT devices. In: Proceedings of the 23rd ACM on Symposium on Access Control Models and Technologies, SACMAT 2018, Indianapolis, IN, USA, 13–15 June 2018, pp. 247–258 (2018). <https://doi.org/10.1145/3205977.3205978>



71. TCG WG: TCG guidance for securing resource-constrained devices. Technical report, Trusted Computing Group (TCG) (2017). <https://trustedcomputinggroup.org/wp-content/uploads/TCG-Guidance-for-Securing-Resource-Constrained-Devices-v1r22.pdf>
72. Thaler, D., Waltermire, D., Housley, R.: Software Updates for Internet of Things (suit) (2018). <https://datatracker.ietf.org/wg/suit/about/>. Accessed Oct 2018
73. The Official OMG SysML site: What Is OMG SysML? (2012). <http://www.omgsysml.org/>. Accessed Apr 2018
74. Tiloca, M., Selander, G., Palombini, F., Park, J.: Secure group communication for CoAP (2018). <https://datatracker.ietf.org/doc/draft-tiloca-core-multicast-oscoop/>. Accessed Oct 2018
75. Weber, M., Kindler, E.: The Petri net markup language. In: Ehrig, H., Reisig, W., Rozenberg, G., Weber, H. (eds.) Petri Net Technology for Communication-Based Systems. LNCS, vol. 2472, pp. 124–144. Springer, Heidelberg (2003). [https://doi.org/10.1007/978-3-540-40022-6\\_7](https://doi.org/10.1007/978-3-540-40022-6_7)
76. Weber, R.H.: Internet of things – new security and privacy challenges. *Comput. Law Secur. Rev.* **26**(1), 23–30 (2010). <https://doi.org/10.1016/J.CLSR.2009.11.008>. <https://www.sciencedirect.com/science/article/pii/S0267364909001939>
77. WfMC: Workflow Management Coalition (2009). <http://www.wfmc.org/>. Accessed July 2017
78. Van der Aalst, W.M.P.: The application of Petri nets to workflow management. *J. Circuits Syst. Comput.* **08**(01), 21–66 (1998). <https://doi.org/10.1142/S0218126698000004>. <http://www.worldscientific.com/doi/abs/10.1142/S0218126698000004>
79. Wolter, C., Menzel, M., Schaad, A., Miseldine, P., Meinel, C.: Model-driven business process security requirement specification. *J. Syst. Arch.* **55**(4), 211–223 (2009). <https://doi.org/10.1016/J.SYSARC.2008.10.002>. <https://www.sciencedirect.com/science/article/pii/S1383762108001471>
80. Wolter, C., Schaad, A.: Modeling of task-based authorization constraints in BPMN. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 64–79. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-75183-0\\_5](https://doi.org/10.1007/978-3-540-75183-0_5)
81. Wolter, C., Schaad, A., Meinel, C.: Task-based entailment constraints for basic workflow patterns. In: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies - SACMAT 2008, p. 51. ACM Press, New York (2008). <https://doi.org/10.1145/1377836.1377844>
82. Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.: Formal methods: practice and experience. *ACM Comput. Surv.* **41**(4), 1–36 (2009). <https://doi.org/10.1145/1592434.1592436>
83. Zhang, F., Cecchetti, E., Croman, K., Juels, A., Shi, E.: Town Crier: an authenticated data feed for smart contracts. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS 2016, pp. 270–282. ACM, New York (2016). <https://doi.org/10.1145/2976749.2978326>