UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

# Workflow-Aware Access Control for the Internet of Things

Prabhakaran Kasinathan

Dissertation eingereicht an der Fakultät für Informatik und Mathematik der Universität Passau zur Erlangung des Grades eines Doktors der Naturwissenschaften

A dissertation submitted to the faculty of computer science and mathematics in partial fulfillment of the requirements for the degree of doctor of natural sciences

Advisors: Prof. Dr. Joachim Posegga
University of Passau, Germany
Prof. Dr. Javier López
University of Malaga, Spain

Submission: Passau, September 2020.

*Dedicated to my beloved family, friends,*
*teachers, and everyone*
*who supported and believed in me.*

# Acknowledgements

As a Ph.D. student, my last four years went remarkably fast and was filled with good memories. I want to thank everyone who made this journey fun, exciting, and enjoyable. All the knowledge that is being taught, shared, and preserved in various forms helped to write this thesis. Now, I take this opportunity to show my gratitude to the people who advised and helped me to finish this thesis.

I thank my advisor Prof. Dr. Jorge Cuellar, who was instrumental for the thesis. Jorge was a great advisor and mentor. I have had the privilege to sit next to him and ask questions whenever I had to and learn from him many things, including the Emacs Org-Mode, which I used to write my thesis. This has been an excellent experience for me. I acknowledge and sincerely thank Jorge for supporting, discussing ideas related to the thesis, and explaining things to me in an understandable way.

I thank Prof. Dr. Joachim Posegga, for giving me this opportunity to do my Ph.D. under his chair and supervision. Prof. Posegga and his team provided valuable feedback and helped me refine my thesis and presentations. I also thank Prof. Dr. Javier Lopez, for being my second supervisor of the Ph.D. thesis, and for providing valuable review and comments that helped to improve my thesis.

I am grateful to the anonymous reviewers from the conferences, book series editors, especially Prof. Dr. Jonathan P. Bowen, and my colleagues for their feedback after carefully reading my papers and listening to my presentations. I enjoyed collaborating with Nejc Zupan while developing the Petri Nets visual modeler. My colleagues, friends, and students who worked with me at Siemens AG provided a friendly, but scientifically thriving environment that played an important role in my thesis, to name a few, Dr. Santiago Suppan, Tiago Gasiba, Tiange Zhao, Fabiola Moyon Constante, Avinash Halsnad, Wolfgang Popp, Susanne Stahnke, and Filip Rezabek.

Last but not least, I thank my manager Dr. Stefan Seltzsam and my funding organization Siemens AG, Corporate Technology, Munich, for funding my research, providing me a great work environment, and the opportunity to work with and learn from industry experts.

Finally, I thank my family for their support, and my wife Radhika for her patience with me while finishing my Ph.D. thesis.

# Abstract

Internet-of-Things (IoT) is defined as a paradigm where "things" have sensing, actuating, communicating, and self-configuring abilities, and are connected to each other and to the Internet. Recent advancements in the manufacturing industry have helped to produce embedded devices with various sensors and actuators in mass numbers at a reduced cost. As part of the IoT revolution, everyday devices such as television, refrigerator, cars, even industrial machines are now connected IoT devices. Recent studies have predicted that by 2025 there will be over 75 billion of such IoT devices connected to the Internet.

The providers of IoT based services want to integrate their services to satisfy customer requirements. For example, in the mobility scenario, different mobility solution providers want to offer a multi-modal ticket to their customers jointly. In such a distributed and loosely coupled environment, each owner and stakeholder wants to secure his/her own integrity, confidentiality, and functionality goals. This means that distributed rules and conditions defined by the individual owners must be enforced on the participating entities (e.g., customers or partners using their services). The owners and stakeholders may not necessarily trust each other's actions. Therefore, a mechanism is required that guarantees the rules and conditions specified by the different owners.

Attacks on IoT devices and similar computing systems are increasing and getting more advanced. IoT devices are often constrained, i.e., they have limited processing power, memory, and energy. Security mechanisms designed for traditional computing systems, e.g., computers, servers, or mobile computing devices such as smartphones, may not fit in those constrained IoT devices. Weak security mechanisms and unenforced security measures were one of the main reasons for recent successful attacks on IoT devices and services. As IoT is now used in many sensitive places, including critical infrastructures, securing them becomes more critical than ever. This thesis focuses on developing mechanisms that secure IoT devices and services and enforcing the rules and conditions specified by the owners on entities that want to access owners' resources.

In classical computer systems, security automata are used for specifying security policies and monitoring mechanisms are used for enforcing such policies. For instance, a reference monitor observes and stops the execution when the security policies are about to be violated, thus, the security policies are enforced. To restrict the adversary from using protected IoT devices or services for malicious purposes,

it is required to ensure that a workflow must be followed to access the protected resource. In distributed IoT systems where the policies are governed by different owners, each owner would like to specify their rules and conditions in their workflows. The workflows contain tasks that must be performed in a particular order. The goal of this thesis is to develop mechanisms to specify and enforce these workflows in the distributed IoT environment.

This thesis introduces a distributed Workflow-Aware (or Workflow-Driven) Access Control (WFAC) framework that restricts the entities to do only what they are allowed to do in a collaborative environment. To gain access to a service protected by the WFAC framework, every workflow participant must prove that he/she is in a particular state of an authorized workflow. Authorized means two things: (a) the owner has authorized the workflow to be executed; (b) the workflow participant is authorized to execute it. This restricts the adversary's access to the devices and its services. The security policies defined by different owners are modeled as workflows and specified using Petri Nets. The policies are then enforced with the help of the WFAC framework which supports error-handling, accountability, integration of practitioner-friendly tools, and interoperability with existing security mechanisms such as OAuth. Thus, the WFAC guarantees the integrity of workflows in a distributed environment.

# Thesis Outline

This thesis is organized into five main parts and appendices. Each part is divided into several chapters.

- Part I presents the introduction, problem statement, methodology, contributions of the thesis, the scope of the thesis, and challenges and requirements focused on this thesis.
- Part II describes existing background research, related work, and state-of-the-art mechanisms on the topics that are focused on this thesis.
- Part III presents the contributions of this thesis that address the research issues presented earlier in part I of this thesis.
- Part IV discusses the evaluation of the proposed framework and mechanisms. Standard methodologies are used to evaluate the framework, protocols, and mechanisms presented in the thesis. Use cases are described and evaluated using the introduced approach.
- Part V discusses the conclusion of the thesis by highlighting the capabilities and limitations of the introduced approach. Also, it proposes future research directions through which this research work can be extended.
- Finally, Part VI includes additional information that provides examples and implementation details.

# Contents

# List of Figures

# List of Tables

# List of Listings

# List of Definitions

# Part I

# Overview

# Chapter 1

# Introduction

This thesis focuses on developing security mechanisms to protect and secure the Internet-of-Things (IoT). IoT is defined as a paradigm where "things" have sensing, actuating, communicating, and self-configuring abilities and are connected to the Internet. Recent studies have predicted that by 2025 there will be over 75 billion of such IoT devices connected to the Internet [1]. Nowadays, IoT is used in different application domains and platforms, including critical infrastructures and industrial systems such as energy generation, health care, and water supply systems. Some IoT devices may have constraints in terms of processing power, memory, storage, and energy source, therefore, conventional security mechanisms developed for standard computing systems such as Internet Protocol Security (IPSec), Transport Layer Security (TLS), and Hypertext Transfer Protocol Secure (HTTPS) may not fit in IoT [4]. In particular, IoT devices are more vulnerable to attacks because of their characteristics such as constraints (processing, memory, power, and storage), direct or indirect connection to the Internet, and physical accessibility (e.g., without physical security measures) [5]. Most of the IoT applications are distributed and the devices usually communicate with their master nodes to make access control decisions and in some situations, the devices themselves make their own access control decisions.

Attacks on IoT and other similar computing systems that are connected to the Internet are evolving rapidly. A successful attack on such critical infrastructures will have serious impacts on the economy, essential commodities, and even on human safety [6]. Recent attacks on IoT systems, in particular, on critical infrastructures are presented by Stellios et al., in their survey paper [7] and the results showed that the attacks happened because of one or more of the following characteristics: (a) physical access to IoT device; (b) connectivity to IoT devices; and (b) unnecessary functionality exposed by IoT devices. An example of attacks on consumer IoT devices is the *Mirai* botnet attack (see [8]), where people failed to enforce some basic best security practices such as changing the default username/password or using strong credentials. In general, the above mentioned attacks show that IoT devices are poorly protected, and when IoT devices used in critical infrastructure

[1] `https://www.cisco.com/c/en/us/solutions/internet-of-things/future-of-iot.html`

are compromised, then the damage can be really serious, therefore, IoT must be protected. This is one of the main motivations of this thesis.

In critical infrastructure and safety systems, protecting and guaranteeing the *integrity and availability goals* of the IoT system is more important than protecting the confidentiality goals [9]. These goals are usually use case specific and when they are elaborated, they transform into a set of rules and conditions that must be followed in a predefined order and this is known as a *workflow*. A detailed definition of workflow is presented later in this thesis in Sec. 9.2. To ensure correct and safe operation of distributed IoT systems the integrity of the workflow must be guaranteed. Therefore, enforcing a *"need to access"* principle is required which is an integrity policy in contrast to the well-known confidentiality policy known as the *need to know* or *principle of least privilege*.

In a typical IoT application scenario, different entities provide and consume services from one another. Each owner that provides a service wants to enforce some specific conditions (e.g., integrity, confidentiality, and functionality conditions) on the consumers of the service. One of the main issues in such multi-tenant distributed IoT systems and architecture is the *trust* issue which is explained in the following example.

**Definition 1.1** (Trust)**.** *"A characteristic of an entity that indicates its ability to perform certain functions or services correctly, fairly and impartially, along with assurance that the entity and its identifier are genuine" (see [10]).*

> **Example:**
>
> Assume that three owners (A, B, and C) agreed to a workflow to provide integrated services to their customers or end-users (i.e., consumers). All owners want their customers to behave in a particular way defined by each of them i.e., by means of a commonly agreed workflow. Ideally, the owner would like to specify a *workflow* in a particular language that declares owner's rules and conditions. Also, this workflow should be binding and guarantee the consumer about the commitment of the owners. The challenges here are the following: (a) How can the owner-A trust other parties (e.g., owner-B, owner-C) and their services to behave according to the agreed workflow? (b) How can the owner be sure that the consumers do not *cheat*?

The use case presented above is only an example of a distributed IoT scenario. In Part Evaluation (IV) - Sec. 15, this thesis discusses further several IoT use cases and scenarios and how the proposed approach addresses them. Those use cases are used to derive the problem statement, goals, and requirements focused on this thesis. The methodology is presented in Sec. 1.3.

Also, usability and interoperability features play an important role in any technology to be adopted by the industrial community i.e., the proposed or developed method must be easy to understand, interoperable with conventional protocols, security mechanisms, and standards such as Authentication and Authorization for Constrained Environments (ACE) and Web Authorization (OAuth). For instance, during the workflow execution, if there are any foreseen or unforeseen error conditions, then the proposed framework should have the ability to process them and if possible, recover from the issues.

## 1.1 Problem Statement

The overall aim of this thesis is to investigate and develop security mechanisms that restrict an adversary from accessing or using IoT devices and services exposed by them for malicious purposes. To achieve this, this thesis focuses on developing a framework where the owners can specify and enforce workflows on end-users accessing their IoT devices or services. In an integrated and distributed IoT application, there will different IoT devices and services provided by different owners. Therefore, the owners might not necessarily trust other owner's (or service provider's) actions. Therefore, an accountability mechanism is required to enforce workflows in a distributed IoT environment.

### 1.1.1 Research Questions

The research questions of this thesis are the following:

**RQ 1:** How to bind access control to the tasks to be executed in a workflow?

**RQ 2:** How to guarantee the integrity of the workflows in a distributed environment without the use of centralized workflow manager and without requiring continuous synchronization with workflow servers?

**RQ 3:** How to model and specify such workflows in a user-friendly way and how to integrate practitioner-friendly tools and interoperable security mechanisms?

**RQ 4:** How to handle error conditions?

**RQ 5:** How to integrate accountability features without using a centralized trusted party?

### 1.1.2 Goals

Based on the research questions, the goal of this thesis is to design and develop mechanisms that ensure that the consumers obey the workflows or contracts defined by the owners in a distributed environment, particularly, supporting distributed IoT applications. This goal can be divided into the following parts:

- (a) Identify or develop a workflow specification language that can support conditions defined by different owners - related to RQ 1.
- (b) Develop a framework that enforces and guarantees workflow integrity. Precisely, the entities participating in the workflow should be able to access the resources only when the workflow is executed as specified, i.e., a method that binds access control to the tasks specified in workflows - related to RQ 2 .
- (c) The developed framework should be easy to understand, modular and interoperable, support integration of practitioner-friendly tools, and have the ability to handle foreseen and unforeseen error situations - related to RQ 3 and RQ 4.
- (d) The developed framework should support accountability features in a distributed environment. A method should guarantee that entities participating in the workflow are held accountable for their actions without using a trusted third party or a trusted centralized database - related to RQ 5.

## 1.2 Related Work

This section presents only the highlights of the related work of this thesis. The Chap. 8 presents an in-depth analysis of related work.

In computer security, a security policy can be defined "as a statement that says what is and what is not, allowed" [11]. For instance, the Biba integrity model [12], and the Clark-Wilson model [13] are integrity policies that focus on protecting data integrity whereas, the Bell-LaPadula model [14] is a confidentiality policy that focuses on protecting data confidentiality.

Fred Schneider introduced *security automata* in [15] for enforcing such confidentiality and integrity policies in a computer system. Basin in [16] studied further in detail what conditions and policies are enforceable. Service Automata presented by Gay et al., in [17] extended the concept of security automata to enforce security policies in a distributed fashion, it uses a *coordinator* to exchange and synchronize information with other service automata, and a traditional *reference monitor* to enforce local policies. Basin et al., in [18] presented a *process algebra*-based approach to align security and business objectives. The aforementioned approaches

do not support the loosely coupled distributed architecture common in IoT where devices work autonomously without permanently having to synchronize with their manager.

An approach to enforce processes on IoT is presented by Tandon et al., in [19] called "History-based Capability systems for IoT" (HCAP) extends the concepts of Security Automata to IoT environment, but the HCAP approach has its limitations because of the following reasons:

- (a) an IoT device exposing some services (or Resource Server (RS)) needs to maintain the state information locally and has to synchronize the state information to the Authorization Server (AS) via other IoT devices in the network,
- (b) when some messages exchanged are lost then it brings also additional overhead to the introduced protocol,
- (c) besides, this approach does not focus on establishing a language for workflow specification, or to provide accountability features,
- (d) as it is based on process algebra, it is not practitioner friendly[2] (see [20]).

Existing research on workflow modeling suggested the use of Petri Nets and its extensions such as *Open Petri Nets* or *Workflow-Nets* for workflow specification. Some researchers proposed using Petri Nets for workflow authorization purposes [21, 22], but all existing work and proposed systems considered *centralized workflow-management* systems with one central workflow management system monitoring and controlling the control-flow of the workflow.

Existing methods and mechanisms are not sufficient to guarantee workflow integrity in a distributed setup where different entities want to protect their goals because of the above mentioned and the following reasons. Approaches based on process algebra are difficult to understand[2] (see [20]), not practitioner-friendly, and difficult to integrate with existing security mechanisms or tools.

Also, the previously mentioned approaches do not support accountability in a decentralized and trust-less environment. To tackle this, other researchers proposed to use blockchain-based access control mechanisms. In [23], a blockchain-based access control method is presented and in their approach the attribute-based access control policies and concepts are used in the bitcoin blockchain system. In [24, 25] a smart contract-based access control solution for IoT is proposed. In [24], each smart contract may contain only one access control rule i.e., subject-object pair with associated rights, in addition, to track misbehaviour of resource access, each resource maintains a list that contains the subject that has accessed the resource

[2] "Despite its strengths, after almost 30 years the use of process algebra outside the academia is still very limited. Due to some of its technicalities, process algebra is perceived by practitioners as being difficult to learn and use …" [20].

with timestamp info and if the subject already had any penalty or not. Moreover, the implementation was based on Ethereum Blockchain and Raspberry Pi devices were used as IoT devices. In contrast, in [25], a single smart contract may contain many access control rules and this approach uses a *management hub* as a proxy component between the IoT devices. The management hub could be a powerful device to handle complex computational tasks that the IoT devices might not be able to perform.

Other than the History-based Capability systems for IoT (HCAP) approach [19], so far, no other approach tackles the core-problem of workflow-aware or history-based access control in an IoT environment. In addition to that WFAC tackles i.e., accountability, a way to handle error conditions, or support integration of practitioner-friendly tools and support the interoperable methods such as OAuth in a distributed IoT environment. In addition, most of the other approaches do not focus on privacy issues (e.g., protecting the identity of the client) in situations where the IoT devices are not able to protect the communication channel between the client and the resource server.

## 1.3 Methodology

First, this thesis will explore how IoT devices are deployed and used in different concrete use cases that are relevant to both consumer and industrial IoT scenarios. From the use cases, the thesis explores what kind of rules and conditions the owners want to enforce on consumers and the generic requirements of the consumer. Second, this thesis will review the existing work that focuses on the identified research questions. In particular, the state-of-the-art mechanisms, modeling approaches and techniques are evaluated to understand their applicability to the identified requirements, selected use case scenarios, benefits, and limitations, i.e., whether they can enforce the goals and requirements identified in these use cases or not. Third, this thesis analyzes the requirements, identifies the problems, and addresses the research questions by developing a new approach that fulfils the goals of this thesis and requirements identified from the use cases. Fourth, to validate the proposed framework a generic prototype is developed and a simple use case is evaluated. Next, a generic attacker-model is formulated and the possible attack scenarios are evaluated against the proposed framework. Next, the idea of the proposed framework and the results are disseminated by publishing in peer-reviewed scientific conferences. Finally, the thesis concludes by presenting what can and cannot be achieved by the proposed framework i.e., possibilities and limitations, and what could be the future directions of this research.

## 1.4 Research Outcome

The outcome of the research work is the design and development of the WFAC framework and mechanisms for IoT devices and services. WFAC framework provides a generic, interoperable, distributed access control framework that guarantees the integrity of workflows, where different owners of distributed IoT resources specify the workflows. The proposed workflow-aware access control framework allows only the authorized entities to execute only the actions specified in the workflow in a specific order, thus enforcing the rules and conditions defined by the owners of the services. Access to resources is granted by enforcing workflows, thus, the *least privilege* and *need-to-access* mechanism are enforced. An adversary must be able to prove that he/she is in a particular state of an authorized workflow to gain access to a service protected by the WFAC approach. WFAC framework uses a practitioner friendly and easy to understand workflow specification language which is amicable to formal verification. WFAC framework integrates accountability features that enable the workflow participants to prove their actions committed, as well as, supports the owners in the auditing process, for example, via an audit trail. WFAC framework provides a workflow modeling tool that can be integrated with practitioner-friendly tools. The modeling tool allows exporting smart contract templates that can be later used together with blockchain platforms such as Ethereum.

Main publications related to this thesis are the following:

- Prabhakaran Kasinathan and Jorge Cuellar. Securing the integrity of workflows in iot. In *Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks, EWSN 2018. Madrid, Spain, February 14-16, 2018*, pages 252–257, 2018

- Prabhakaran Kasinathan and Jorge Cuellar. Workflow-aware security of integrated mobility services. In *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II*, pages 3–19, 2018

- Prabhakaran Kasinathan and Jorge Cuellar. Securing emergent iot applications. In *Engineering Trustworthy Software Systems: 4th International School, SETSS 2018, Chongqing, China, April 7–12, 2018, Tutorial Lectures*, pages 99–147, Cham, 2019. Springer International Publishing

- Nejc Zupan, Prabhakaran Kasinathan, Jorge Cuellar, and Markus Sauer. Secure smart contract generation based on petri nets. In *Blockchain Technologies for Industry 4.0*, Cham, 2019. In Press: Springer Singapore Publishing

- Jorge Cuellar, Prabhakaran Kasinathan, and Daniel Calvo. Privacy-Enhanced-Tokens (PAT) profile for ACE. Internet-Draft draft-cuellar-ace-pat-priv-enhanced-

authz-tokens-06, Internet Engineering Task Force, January 2018. Work in Progress

The thesis also investigates and develops a method that can be integrated with IoT protocols such as the ACE OAuth framework. As a result, an Internet Engineering Task Force (IETF) draft [30] was published (see Sec. 11.1 for more details).

### 1.4.1 Contributions

Research in workflow modeling, validation, and verification exists for many years [31, 32]. Petri Nets has applications to model and specify workflows [33, 34, 35] because it is formally amicable to verification of workflow properties like deadlock and workflow soundness properties. Also, Petri Nets are easier to understand without any prior knowledge [34]. In addition, the extendable nature of Petri Nets supported the requirements identified in the thesis. Petri Nets also supports the integration of distributed architecture with different devices, entities, and services. Therefore, this thesis investigated and developed further extensions to Petri Nets that could support distributed workflow execution and enforcement.

The contributions of this thesis can be categorized in three different domains:

#### Workflow Specification and Execution

- Extended the original Petri Nets concepts and adapting existing extensions to fulfill the general requirements that are necessary to develop the envisioned WFAC, they are listed below.
  - *Oracle places* to support communication with external services and entities
  - *Error places and tokens* to handle error conditions
  - *If-then-else conditional guarded commands* and *transition contracts* with no recursions, therefore, they are not Turing complete.
  - *Timeout transitions* to handle certain types of error conditions such as indefinite waiting and to support application logic that requires alternative actions after a certain timeout
  - Workflow synchronization and exchange of information using Open places (adopted from Open Petri Nets), this need not synchronize with a central workflow server but with other similar workflow execution devices or services

- Using the SysML activity diagram to model the business or technical process of owners. The support and integration of practitioner-friendly tools such as SysML increase the usability aspects of the proposed WFAC framework
- Introducing the Petri Nets as the execution engine of workflow application used in the WFAC. The workflow application can be installed in handhelds such as a smartphone. This workflow application processes the tokens, validate the conditions and rules written in the transition contract, and supports interoperable protocols such as OAuth.
- Distributed workflow execution between several entities, for example, consumers or users (in OAuth terms *clients*) use their handheld to execute a workflow and on the other end, there could be the owner executing the other part of the workflow on his/her handheld. A seamless workflow synchronization and error handling is possible with the help of open places to exchange workflow information between Petri Nets workflows, and error places for error handling respectively
- Integration of accountability features with workflow execution ensures that the entities behave properly. This thesis introduced the logging of decisions and actions taken by the entities in a private blockchain network without assuming a central trusted authority. If there is a requirement to penalize misbehaving entities for their actions in some use cases, then, it is possible to extend this concept even to penalizing the entities that misbehave.

### IoT Devices and Services

- To ensure that a client has completed a workflow task or action, the resource server provides a signed acknowledgment token called the *receipt token* with workflow specific data. This receipt token is then provided to the workflow execution engine or to other resource servers which processes them to verify if the appropriate workflow action or task is done correctly or not.
- To enhance privacy, an OAuth profile for IoT devices is developed and published as an IETF draft [30].

### Smart Contract Generation

- The developed WFAC framework can be used to generate smart contracts for blockchain applications. The Petri Nets workflows are translated as smart contract templates using a generic smart contract code generation module. The thesis has contributed to the development of the Petri Nets based smart contract generation framework. Once, the smart contract template is generated, then, with little additional development effort, it can be deployed

on a blockchain. For instance, the framework now supports the generation of solidity smart contract code template from the Petri Nets which can be deployed in Ethereum blockchain.

## 1.5 Scope of the Thesis

The scope of the thesis is defined based on the requirements identified in the use cases and the aim of the thesis i.e., research questions and goals defined in the Sec. 1.1.1.

The following points are considered within the scope of this thesis:

- A generic functional prototype and architecture of WFAC
  - Architecture design and definition of the framework and its sub-frameworks were considered. A generic prototype implementation for evaluation and demonstrating the WFAC. This includes necessary web-applications or handheld applications that demonstrate the components used in the framework.
- Case Studies and Evaluation
  - Description of concrete use cases that reflect both consumer and industrial use cases and how the proposed method can be used to solve the identified use cases.
- IoT Interoperability
  - Investigation and integration of mechanisms that enable the proposed WFAC framework to support constrained IoT devices, for instance, a guide to integrate and use the proposed WFAC framework with generic IoT platforms and services.
- Accountability Features
  - Integration of accountability features with the proposed WFAC framework where the consumers (or workflow participants) and the owners must be held accountable for their actions. Investigation of how accountability aspects can be integrated with the proposed WFAC framework.

The following points are considered out of the scope of this thesis; the rationale behind defining them is also provided below.

- Formal verification of workflow specification, validation, and verification aspects of a workflow.

  - Petri Nets are well studied and have been used for formal verification. Existing studies have shown how to achieve this [36, 37]. Therefore, formally verifying each workflow is use case specific and therefore, was not the scope of this thesis.

- Usability study of the proposed WFAC framework in terms of how intuitive and easy it is to use the proposed system
  - Usability study needs more stable implementation and interviews with a wide range of population. This thesis intends to develop a prototype of the WFAC framework. The usability claim that "Systems Modeling Language (SysML) is practitioner friendly" is based on already proven research (see [38, 39, 40]) that has conducted systematic usability studies. As the proposed framework can integrate SysML with our approach, we assume that our method is also practitioner friendly. However, this can be studied in the future.

- Conflict resolution during negotiation phase:
  - In distributed IoT scenarios, as described earlier there might be different owners who want to enforce their own security goals. These conflicts can occur during any negotiation phase and the developed framework is not intended to solve the conflicts itself. However, the WFAC framework presents a workflow specification method using which each entity can clearly describe only their interests to other participating entities without completely showing the internal confidential workflow.

- Implementation / validation in constrained IoT devices
  - Nowadays IoT devices come in different flavors i.e., the underlying technology, constrains and communication standards are different. In this thesis, the evaluation is performed on one selected constrained IoT device (e.g., which complies with the IETF draft "Terminologies for constrained-node networks" [41]) and it is not intended to support all range of available IoT constrained devices in the market.

- Regulatory compliance to Privacy Regulations such as General Data Protection Regulation (GDPR)
  - The use of technologies such as Blockchain in the proposed approach may raise a conflict with adherence or compliance to regional privacy regulations such as GDPR e.g., "Right to be forgotten" etc. A recent study (see [42]) submitted to the EU Parliament discusses these issues carefully and proposes three different policy recommendations to address this issue. This topic is currently being researched, discussed, and still an open issue. This topic is out of scope as it was not the main focus of this thesis, however, as it is an important topic and could be investigated as future work.

# Chapter 2

# Challenges and Requirements

This chapter presents the challenges and requirements in generic IoT applications. The challenges and requirements are gathered from concrete use cases identified and presented in the thesis (see Sec. 15 for detailed use case description) and are grouped into two categories and described in detailed: (a) security and privacy and (b) usability and interoperability.

## 2.1 Security and Privacy

Security and Privacy challenges in IoT and Industrial Internet of Things (IIoT) are discussed in [43, 44, 45, 6] where the authors discuss technical, financial, and legal issues involved in IoT and existing solutions. This thesis discusses only the technical aspects of security and privacy challenges in IoT and IIoT. The Open Web Application Security Project (OWASP) presented the Top 10 IoT vulnerabilities and attack surfaces (see [46]), now the topics relevant to this thesis are discussed below.

### 2.1.1 Distributed 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. Academic and industrial researchers are working towards addressing them (see IETF ACE working group [47]). 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, attackers 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 [8]. 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.

Distributed authorization mechanisms are important to support a growing number of IoT devices. Authorization in distributed systems is complex to achieve [48] as the resources are spread across a network of devices in different domains. The service provided by multi-tenant systems (e.g., including IoT devices) might know each other or not. 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 Access Control (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. Following such a standard approach (for instance, IETF ACE [47]) ensures interoperability. More information about this topic is presented in Sec. 5.2.

### 2.1.2 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 TLS. Light-weight protocols, such as Datagram Transport Layer Security (DTLS) over Constrained Application Protocol (CoAP) (See [49]) have been designed to support the confidentiality and integrity of transported data. For instance, IoT constrained devices (e.g., class 1 or class 2 devices) may not handle DTLS properly, e.g., even if the firmware fit it might have performance issues such as loss of messages, retransmission of messages, and thus, affecting the application and battery performance of IoT.

*Privacy* deals with unlinkability or anonymity of the identity of a person and confidentiality and control over that person's private information. In the era of social networking, it is easy to identify, track, locate a person using information collected from various sources, and this information can be used for malicious purposes. Nowadays, there are regulations that demand companies protect the private information of their customers. For instance, to protect the privacy of people, the European Union recently introduced the GDPR regulation to enforce and regulate how companies can collect and process information of the people living in the EU. Similar regulations exist in different parts of the world. 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 [44].

Depending on the use case, involved actors, and sensitive (e.g., private) information involved different actions might be needed to be compliant with such regulations. In such a scenario, the owners of IoT services might want to be compliant with such regulations in terms of customers' and service providers' data. Let us consider the following example to understand the situation better.

> **Example:**
>
> Assume a car owner (*a client*) charges his car in several charging stations (*resources*). The charging stations might be owned by different owners and energy suppliers. If two or more car charging station owners share the information of their customers, then it is possible to track the car's identity and as a result, private information of the end-user driving the car can be revealed.

Protecting the end-users' identity in situations like the aforementioned use cases are important. Therefore, this thesis adopts or develops *privacy-enhancing techniques* in the communication protocol used between client devices and resource servers (i.e., IoT devices).

### 2.1.3 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 therefore, various resilience mechanisms are necessary to solve this problem. Authorization coupled with the context of task execution workflow is one way to stop the attack escalation problem.

> **Example:**
>
> 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, proper security mechanisms to restrict the attacker from compromising other devices or equipment via a weak compromised device are necessary.

To prevent such attacks, let us assume that a workflow must be followed (with respect to WFAC) for initializing software updates or updating the configuration of devices inside the manufacturing plant. With WFAC in place, the attacker cannot perform this attack unless he was able to execute that workflow and reach

the state which allows him to perform a software update or compromise several devices that are part of the workflow.

### 2.1.4 Self-Configuration and Multi-Tenancy

It is evident that IoT devices are getting powerful, cheaper, (see [50]) and energy-efficient day-by-day. Installing and configuring such advanced IoT devices with existing IoT applications should not require too much human involvement. IoT devices should have self-configuring features i.e., backward compatibility, resilient to connection loses 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.

### 2.1.5 Fine-Grained Access Control and Integrity

Access control mechanisms restrict access to resources, for example, to unlock a smartphone the owner of the smartphone uses a Personal Identification Number (PIN) or his/her fingerprint. This is a typical access control scenario where the owner (*the subject*) is granted access to the smartphone (*the object*) if the presented PIN or the fingerprint is correct. The purpose of an AC mechanism is to stop unintended subjects from accessing protected objects.

In Role-Based Access Control (RBAC) see [51], access control and authorization decisions are made based on the Role given to an entity. A role like an *admin* is very powerful and by default has (almost all) permissions such as to change, add, and delete features of a system. For example, it is a bad idea to give access to all resources in all IoT devices to one single administrator account, because if that admin credential is stolen or misused, then the attacker is able to access all the resources associated with the credential. Therefore, it is important to limit the set of permissions (fine-grained) given to an entity. To achieve this, a *least privilege* principle for task authorizations is required within each workflow. The principle "Need to Know" is a confidentiality policy that states that no subject should be able to read objects unless reading them is necessary for that subject to perform its functions [11]. A similar policy, but regarding integrity is required which is called the "Need to Access" principle which states that no subject should be able to *access* the objects to perform either *read, write, or update*. Therefore, only when it is necessary to complete the required task in a workflow, access is given to the required resource. This motivates the need for a more restrictive fine-grained access control model such as the proposed *workflow-aware access control* in this thesis.

### Workflow Integrity in Distributed Environments

Integrity aspects are vital for ensuring accountability and as well as to provide assurance that the data or process is not updated in an unauthorized way. The first aspect of integrity deals with *data communication integrity* – the assurance that the data transferred from one entity to another has not been altered or tampered with. The second aspect is *data integrity* – the assurance that data stored in memory including, firmware, key material, data, or programs stored in memory is not altered or tampered. The third aspect of integrity is "*workflow integrity*". The assurance that the workflow (as defined) is completed successfully without skipping or adding any further steps is important and is known as "*workflow integrity*". As part of this thesis, a more detailed definition of workflow and workflow integrity is presented in Sec. 9.2. The simplest way to enforce workflow integrity is to have a centralized workflow management server that monitors the execution of the workflow and gives access to participants (see Sec. 8) accordingly. But this solution is not suitable for a distributed IoT scenarios because of the following reasons: (a) scalability issues - one server cannot handle millions of requests at a time (b) failure with centralized workflow management system means all tasks / or resources are not accessible; (c) all owners of IoT services must trust that the centralized server will handle sensitive (e.g., private) information confidentially and protect the integrity of the data; (d) depending on the use cases, the owners might not want to reveal information about private workflow to other participants. Thus, when distributed IoT devices and services are involved as presented in the use cases (see Sec. 15), the assurance that the workflow is executed properly in a distributed environment is difficult to achieve. As mentioned earlier, the motivation of this thesis is to develop a solution to guarantee workflow integrity in such distributed environments.

### Validation and Verification of workflows.

The workflow specified by the owners must not have errors such as deadlocks, because when such errors exist a workflow might end in a state where it cannot proceed. 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 [52]. 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, a modeling language is required to verify some properties such as deadlocks and soundness properties of a workflow.

### 2.1.6 Distributed Accountability

**Definition 2.1.1** (Accountability)**.** *"The property of a system or system resource that ensures that the actions of a system entity may be traced uniquely to that entity, which can then be held responsible for its actions" (see [53]).*

*Accountability* refers to the responsibility of actions committed by an entity. Accountability is often associated with the recording of entities' actions or computer events and storing (*logging*) this information in a secure way. This gives the ability to audit a computer system or process. *Auditing* is one of the applications of accountability. *Non-repudiation* is a special property that provides *proof* that an entity committed an action and this cannot be denied in the future. Asymmetric cryptography combined with digital signatures is an application example of non-repudiation.

To achieve accountability in a system, first, the system must record (e.g., log) all important actions/interactions of entities with the system, including solicitation and execution. Second, the system must guarantee data integrity requirements of logs data processed and stored. But in the case of distributed systems where some systems handling the data may not behave in the expected way (i.e., cannot be trusted) then, achieving accountability there is more complex.

Accountability protocols go one step further and provide the ability to ensure two following features: (a) *Non-repudiation of actions*: neither party can deny their actions in the interaction; (b) *Fairness*: both participants receive the expected outcome of the transaction, or neither do (see [54, 55]). Kuesters et al., in [56] presented a definition of accountability which allows to precisely capture the level of accountability a protocol provides.

Logging is a standard feature in many computing systems. Logging information usually includes data such as system activities, process executions, user interactions, etc. with relevant information such as timestamps and user identifier. Auditing is an independent analysis of accounting records i.e., in a computer system, it can be a program trace, log information, etc. For instance, in case of system/equipment failure, the production plant auditors must have the capability to find the root cause of the incident. In case of multiple parties involved, a judge or a protocol is used to backtrack the events and find what has led to the failure or problem.

> **Example:**
>
> Bitcoin's (e.g., of blockchain) is a good example where distributed accountability is achieved. In Bitcoin platform, different parties exchanging money (bitcoins) in the form of transactions without trusting a single trusted entity (e.g., a bank), but trust the underlying cryptography and consensus protocol which relies upon the fact that more than 50% of the participants validating

the transactions are behaving good (for more information see [57]).

## 2.2 Interoperability and Usability

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 [58]). Also, the research community such as ACE (see [47]) 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.

one of the important factors that influence the adoption of a new computer security mechanism is usability and interoperability of the mechanism. First, the usability aspects are discussed below from both end-user's and owner's perspective.

- End-user: The user who will be using the introduced security mechanism to access resources. The introduced mechanism must support the end-user to continue his day-to-day without introducing additional complexity.
- Owners: The resource owners and system integrators should benefit from using the proposed mechanism. The owner's decision is usually influenced by the cost involved, market advantage, improved process efficiency and security guarantees, and last but not least customer acceptance.

Second, interoperability with most common technology and protocols that are used in IoT devices is important. The proposed method should be technology or protocol agnostic. For instance, the ability to execute a workflow in different client devices such as smartphones and interact with different IoT devices and services, i.e., even a small IoT device must be able to understand the access tokens presented by the clients.

- End-resources: Often, industrial systems have a longer lifetime than commercial systems and the introduced security mechanisms should be interoperable and support the existing systems.

### 2.2.1 Exception Handling

It is difficult to model all possible edge cases of a use case in a workflow model. Therefore, handling exception is an important feature for a workflow execution system. An IoT application should be capable of recovering from unforeseen error situations to an extent. One way to recover from unforeseen situations that cannot

be fixed by the system itself is by allowing human interaction to solve a problem. A workflow may allow the owner of the services to create on the fly sub-workflows to recover from error conditions. The owners must take care that the sub-workflows do not change the main objective of the main-workflow. Also, the owners must ensure that relevant information is documented for accountability purposes. The ability to adapt and recover from error situations is an important requirement to build usable security in an IoT application.

### 2.2.2 Support for Requirements Elicitation

The requirements engineering process can be divided into four tasks namely the elicitation, negotiation, specification/documentation, and verification/validation of requirements [59]. The process of collecting requirements is called *requirements elicitation*. The focus of the thesis is not to provide a solution for requirement elicitation but only to support it. As this thesis focuses on handling distributed IoT applications, the requirements come from different owners. Therefore, elicitation of the requirements and identification of the relevant sources (owners or stakeholders) during the elicitation task is crucial. Modern tools such as Unified Modeling Language (UML) or SysML allow us to collect requirements from the use cases, and represent the use case activities via activity diagrams. SysML provides tight integration of both software and hardware components such that a use case can be modeled efficiently.

# Part II

# Background and Related Work

# Chapter 3

# Computer Security and Access Control

To ensure that the reader understands the terminologies used in this thesis, this chapter presents a brief introduction to some key computer security terminologies and concepts. Computer security's three main principles are *confidentiality*, *integrity*, and *availability*, also popularly known as the **CIA** triad.

*Confidentiality* is about hiding or concealing some information or resources (for more information, see [11]). Access Control mechanisms (e.g., of security mechanism) can be used to restrict the access given to confidential information. One of the applications of cryptography's *encryption and decryption* mechanism is preserving the confidentiality of some digital information (i.e., data confidentiality) available in the form of documents, etc. An encryption key is used to *encrypt* a document to make it incomprehensible (or also known as *ciphertext*), and the corresponding key-pair (decryption key) is used to *decrypt* the ciphertext to retrieve the original document. Depending on the type of cryptography used, for instance, in symmetric cryptography the encryption and decryption keys are the same, but in asymmetric cryptography, the encryption and decryption keys are different i.e., the *public & private* keys.

*Integrity* mechanisms are used to prevent or detect an improper or unauthorized change of data or resources (see [11]). There are at least three aspects of integrity. First, the integrity of data communication i.e., the assurance that the data transferred from one entity to another has not been altered or tampered with. Second, the integrity of data stored (i.e., data integrity) i.e., data stored in memory, disks, or databases, etc. are not altered or tampered e.g., firmware, key material, or programs. The third aspect of integrity is the "*workflow integrity*" - a *workflow* is a set of tasks that must be carried out in a predefined order (see Sec. 9.2) - i.e., no unauthorized entity can change the workflow by adding or deleting or modifying the tasks or the order in which they are processed.

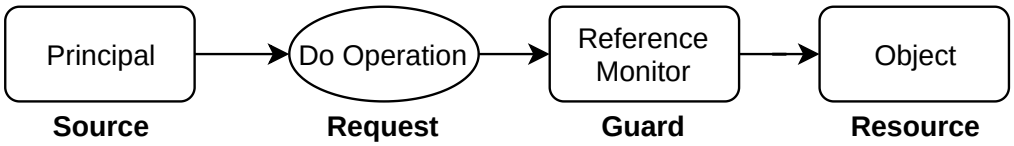



Figure 3.1: The Generic Access Control Model [1]

> **Note:** This thesis focuses on the third aspect of integrity i.e., the workflow integrity.

*Availability* deals with the ability to use a system, service, information, or resources when required. Availability of a computing system is particularly very important in scenarios such as manufacturing or banking applications where the system must be available at all times. Availability mechanisms help to build reliable and resilient systems.

The security of a computer system rests on the assumptions which are specific to the requirements of the system design and the situation where it is used [11]. In computer security, a system can stay in one of the two states: "*secure*" or "*non-secure*" states. A security mechanism helps a computer system to stay in a secure state using different methods, for example, a security mechanism may prevent the system from entering into a "non-secure" state, or it might stop the system. If the system moves to a "non-secure" state, then it is considered a *compromised* system. In practice, this is very difficult to guarantee that a system always stays in the secure state, because of the complexity of today's computer systems and many external factors e.g., human errors. Therefore, a computer system can never be 100% secure. For more information on computer security topics, please refer to the Matt Bishop's book "*Computer Security: art and science*" [11].

Now, an introduction to access control, state-of-the-art access control mechanisms and security policies is presented. A generic access control model is shown in Fig. 3.1 where the following different elements are involved (see [1]):

- *Principals*: source of the access request,
- *Do (Request) Operation*: contains what operations to perform on the objects e.g., read/write/update etc.,
- *Reference Monitor*: examines each request for the object, checks the conditions (guard), and decides whether to grant the *principal* access to the *objects* or not,
- *Object*: The final resource such as files, devices, or processes.

Gollman in [60] states that: "access control consists of two steps, authentication and authorization". *Authentication* is a process that involves correctly identifying

the entity principal. Usually, the entity presents some identification information e.g., username and a password (known only to the entity) to the authentication server. The authentication server checks this information and if it is right, then it allows the entity to proceed next step i.e., usually, the authorization. *Authorization* is a process that evaluates whether an authenticated entity is allowed to access particular information or perform a certain action (request operation) on a resource or not.

Security policies can be categorized based on the security goals it deals with. A *confidentiality policy* is a security policy that deals only with confidentiality. An *integrity policy* is a security policy that deals only with integrity [11].

**Definition 3.0.1** (Access Control)**.** *"The process of granting access to information system resources only to authorized users, programs, processes, or other systems" (see National Institute of Standards and Technology (NIST) [61])*

**Definition 3.0.2** (Access Control Mechanism)**.** *"The logical component that serves to receive the access request from the subject, to decide, and to enforce the access decision" (see NIST [62]).*

**Definition 3.0.3** (Reference Monitor)**.** *"An access control concept that refers to an abstract machine that mediates all accesses to objects by subjects" (see Trusted Computer System Evaluation Criteria (TCSEC) [63]).*

There are three main types of access control:

- (a) *static access control* - access control rules are static (do not-change) and predefined i.e., whenever an access control request with the same data is performed, one gets the same access decision response.
- (B) *dynamic access control* - access control rules are dynamic based on different aspects such as temporal or cardinality constraints - e.g., during weekends no access, access granted at most twice a day. Therefore, depending on the dynamic constraints and the request, the access decisions are made.
- (C) *history-based access control* - access control decisions depend on the actions committed in the past (history-based). E.g., in the Chinese wall policies [64], a person cannot access data on two or more Conflict of Interest (COI) classes are defined i.e., assume *A, B, and C* are COI, if a *subject* has accessed COI-A, then he/she cannot access COI-B or COI-C. One could see history-based access control also a type of dynamic access control. This thesis, deals with history-based access control.

## 3.1 Static Access Control

There are two main types of static access control, one is the Mandatory Access Control (MAC) and the other is Discretionary Access Control (DAC) - they are used alone or in combination in many access control systems.

**Definition 3.1.1** (Mandatory Access Control)**.** *"an access control policy that is uniformly enforced across all subjects and objects within the boundary of an information system" [65].*

In Mandatory Access Control (MAC), a subject that has been granted access to information is constrained from doing any of the following:

- (i) passing the information to unauthorized subjects or objects;
- (ii) granting its privileges to other subjects;
- (iii) changing one or more security attributes on subjects, objects, the information system, or system components;
- (iv) choosing the security attributes to be associated with newly-created or modified objects; or
- (v) changing the rules governing access control.

Organization-defined subjects may explicitly be granted organization-defined privileges (i.e., they are trusted subjects) such that they are not limited by some or all of the above constraints

**Definition 3.1.2** (Discretionary Access Control)**.** *NIST [65] states that: "an access control policy that is enforced over all subjects and objects in an information system where the policy specifies that a subject that has been granted access to information can do one or more of the following:*

- *(i) pass the information to other subjects or objects;*
- *(ii) grant its privileges to other subjects;*
- *(iii) change security attributes on subjects, objects, information systems, or system components;*
- *(iv) choose the security attributes to be associated with newly-created or revised objects; or*
- *(v) change the rules governing access control."*

In MAC, a computer access control mechanism controls access to the objects and the user has no control. In DAC, the individual user (usually the owner) has control over the object i.e., the user may decide the access control policies for the object. The definitions of MAC and DAC are further refined in the DoD Trusted Computer System Evaluation Criteria (TCSEC) or "Orange Book" [TCSEC] [63]. Sandhu and Samarati in [51] presented a holistic view of access control principles and how it is used in practice.

**Definition 3.1.3** (Security Model)**.** *A* security model *is a model that represents a set of policies or a particular policy [11].*

The Bell-LaPadula model is a security model that (see [14]) deals with confidentiality policies. It is usually described in short with "*No Read Up and No Write Down*" policy. The Biba Model (see [12]) deals with integrity policies. It is usually described in short with *No Write Up and No Read Down* policy.

**Example:**

Consider that some military documents (*Objects*) are classified into four different classes (*security classification*) in a linear order e.g., Top-Secret, Secret, Confidential, and Unclassified - the order represents sensitivity level of those documents; and, the military officials (*subjects*) are given some *security clearances* matching those of the security classifications.

No Read Up: an official (*subject*) with low clearance cannot read a *high* classified document. No Write Down: an official (*subject*) with high clearance cannot write a piece of high sensitive information into a *low* sensitive object. The Bell-LaPadula model protects information flow from low to high or vice-versa: by preventing the low classified subjects from reading high sensitive documents and the high classified subjects writing into low sensitive documents.

No Write Up: an official (*subject*) with low clearance cannot write into a *high* classified document. No Read Down: an official (*subject*) with high clearance cannot read a piece of low sensitive information. The Biba model protects the data integrity: by preventing the low classified subjects from writing into high sensitive documents and the high classified subjects reading from the low sensitive documents.

The Clark-Wilson Integrity model (see [13]) presented the importance of the commercial environment's integrity requirements, and introduced a data integrity policy that presents two important principles of data integrity principles, the *separation of duties* and the *well-formed transactions*. A well-formed transaction must satisfy several integrity properties that ensure the integrity of a transaction - this prevents the subjects to modify the data in an arbitrary fashion by enforcing some

constrains that the final data must posses. To have a complete overview of the models presented and other models please see [11].

Some of the important principles used for designing access control mechanisms are the following:

- *separation of duty* requires more than one *subject* to complete a process - this property is employed to prevent fraud or error, and usually implemented when high integrity requirements are necessary to protect a process or an object.
- "The principle of least privilege states that a subject should be given only those privileges that it needs in order to complete its task" [11].
- The *need-to-know* and *need-to-access* are closely associated with the principle of least privilege. The *need-to-know* policy focuses on the confidentiality of information. The *need-to-know* states that permission to *read* some information is only given if the subject needs that information to perform his tasks or duties. Whereas, the *need-to-access* policy deals with the integrity of a resource or information, where the permission to write, append or change the information is given to the subject if the subject needs that permission to complete his task or duties.

### 3.1.1 Access Control Matrix

The simplest and the abstract form of describing the conditions of a secure state or (*protection state*) of a system is the access control matrix model [11]. The set of entities (e.g., *files*) that are protected is the set of *Objects* $O$. The set of active entities (e.g., *user or process*) that tries to perform some actions on protected objects is the set of *Subjects* $S$. The permissions or rights that subject $S$ has on objects $O$ is given as a matrix $A$ with a set of *Rights* $R$ with each entry as $a[s, o]$, where $s \in S$, $o \in O$, and $a[s, o] \subseteq R$ [66, 67, 68]. A subject $s$ has a set of rights $a[s, o]$ on object $o$. The set of protection state can now be represented as a triple $(S, O, A)$.

> **Example:**
>
> The access control matrix presented in Tab. 3.1 shows that there are two processes (*p1 and p2*) and files (*f1 and f2*). Process (p1) has access to *read* and *write* file (f1), but only *read* access on file (f2).

> **Note:** The interpretation of the meaning of rights changes from one system to another and it depends on how the system implements it. The right *own* is usually associated with the subject that created the resource or object in this

Table 3.1: An access control matrix example

| | file (f1) | file (f2) | process (p1) | process (p2) |
|---|---|---|---|---|
| process (p1) | read, write | read | read, write, execute, own | write |
| process (p2) | append | read | read | read, write, execute, own |

case, and it may have special privileges over the object.

In a traditional access control matrix, the objects are usually files and processes, but they can be easily extended to functions, messages sent between processes, etc. Several extensions of the access control matrix also exist in the literature but they are out-of-scope of this thesis.

### 3.1.2 Access Control List (ACL)

Access Control Lists (ACLs) are associated with the objects. An ACL of an object contains a set of pairs, with each pair containing a subject and a set of rights required to access that object. The access control list of objects can be derived from each column of an access control matrix.

**Example:**

If you consider the access control matrix presented in Tab. 3.1, then the corresponding ACL of file (f1) is:

- acl(f1) = { (p1, {read,write}), (p2,{append}) }

An access control list can be either used to give a subject access to an object (*whitelist*) or even to deny a subject access to an object (*blacklist*). As mentioned earlier, the meaning of the rights depends on the interpretation and how it is implemented in the system.

### 3.1.3 Capabilities

*Capabilities* are associated with the subjects. The capabilities of subjects can be derived from each row of an access control matrix. In capabilities, each subject is associated with a set of pairs, with each pair containing an object and a set of rights that the subject has over the object.

**Example:**

If you consider the access control matrix presented in Tab. 3.1, then the corresponding capability list of the subject process (p1) is:

- cap(p1) = { (f1, {read, write}), (f2, {read}), (p1, {read, write, execute, own}), (p2, {write}) }

Capabilities are very interesting because the rights to access an object are bound with the subject's identity. When a subject needs to access an object, it needs only the capability list with permissions or rights to access that particular object. The concept of capability is widely used in modern access tokens such as *oauth* token - a particular form of short-lived capabilities created by an authorization server to access a protected resource using OAuth protocol. Capability is used often to give a subject access to an object, rather than to restriction purposes.

**Example:**

When the process (p1) needs to access the file (f1), then it requires only to present the following capability list to the operating system:

- cap(p1) = { (f1, {read, write}) }

In a sense, this capability can be compared to an access token to access the protected object, i.e., here, the file (f1).

*Remark*: This thesis uses the concept of capability, when it discusses the workflow-aware access control mechanism.

### 3.1.4 Role Based Access Control (RBAC)

In Role-Based Access Control (RBAC), the users (*subjects*) are assigned to a *role*, and each role is associated with a set of rights on the *objects*. RBAC is widely used in many modern access control systems because it simplifies the management of permissions. The RBAC includes various components such as role-permissions, user-role, and role-role relationships. Two main extensions to the plain RBAC are the *role-hierarchies* $RBAC^1$ – where a child role can inherit permissions from the parent roles – and the *constraints* $RBAC^2$– where constraints such as mutual exclusion or cardinality constraints can be specified. The consolidated model $RBAC^3$ which includes both $RBAC^1$ and $RBAC^2$ was presented with many open issues. These different relationships can be used to support previously discussed principles such as the least privilege and separation of duties. Though RBAC supports these principles, the enforcement of the properties relies on the implementation. Furthermore, special care must be taken to manage the RBAC itself . The authors

introduce an additional RBAC layer for managing administrative permissions called the ARBAC (see [69]).

### 3.1.5 Task Based Access Control (TBAC)

In classical subject object based access control, the protection state $P \subseteq S \times O \times A$, where $S$ is the set of subjects, $O$ is the set of objects, and $A$ is the set of actions or rights. Thomas and Sandhu in [70] presented an access control paradigm where access control authorization decisions are based on *tasks* defined in a workflow. By doing this, they wanted to include the contextual information of the ongoing tasks or activities when making access control decisions. The Task-Based Access Control (TBAC)'s protection state extended the subject/object access control representation as $P \subseteq S \times O \times A \times U \times AS$ where, $U$ is the usage and validity counts specification, and $AS$ is the set of authorization-steps. The $U$ specifies the number of times the task can be executed. Each authorization-step instance (can be seen as a data structure) includes several components that include basic contextual information about the tasks, which users can invoke/grant the authorization-step, etc. Furthermore, two extensions (a) composition of two authorization-steps TBAC$^1$ and (b) constraints TBAC$^2$ (similar to the RBAC$^2$), and (c) a consolidated model TBAC$^3$ which includes both TBAC$^2$ and TBAC$^3$ were proposed with many open issues left for further discussion.

### 3.1.6 Attribute Based Access Control (ABAC)

**Definition 3.1.4** (Attribute-based Access Control)**.** *"An access control method where subject requests to perform operations on objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions" (see [62]).*

In Attribute-Based Access Control (ABAC), arbitrary *attributes* (e.g. of users, groups, roles, data resources, environmental attributes, etc) are used to create flexible and complex access control policies, which are then used to make access control decisions [71, 62]. A popular approach is to use Extensible Access Control Markup Language (XACML), where attributes are specified as name-value pairs [72].

There are several variations of access control now being used and deployed in the industrial setting. One such example is the claim based access identity and access control [73], where a claim represents "a statement that one subject makes about oneself or another subject"[1]. For example, a subject may claim its identity, role,

[1] `http://msdn.microsoft.com/en-us/library/ee534975.aspx`

group membership, etc. In one aspect, you can see claims as security tokens with arbitrary attributes.

### 3.1.7 Discussion

The access control matrix is a good method to express a security policy and its simplicity makes it a good candidate for theoretical analyzes of security problems, but in practice it is not used because of space requirements required to map all the objects and subjects [11]. Also, in [74], the authors have discussed in detail different issues with existing access control mechanisms for interconnected systems connected to the Internet such as the IoT and how one single access control method might not be suitable for all use cases. Access Control List (ACL) also faces similar problems with space requirements. Managing the permissions in the ACL – who can add, delete, append information to it? or how to assign default permissions to subjects – is a drawback of ACL. Capabilities are flexible, simple, and can be seen as a security token given to a subject with fine grained access to a particular object. Currently, they are used together with many other access control mechanisms. In RBAC, the management of permissions is made easier by mapping a set of permissions to a role, and users are assigned to a role. RBAC is very common and it works well in a centralized setup. When the system grows bigger, then it becomes cumbersome to manage it. To support distributed systems, another layer of administrative RBAC is necessary. TBAC extends the traditional subject/object access control by adding contextual information about ongoing tasks and the number of times a subject may perform a task, without specifying a way to implement or integrate it with standard systems. In ABAC, arbitrary attributes are used to have more flexibility and to express information that was not possible before, for example, only with roles in RBAC. However, it gets more complex to specify ABAC. To support this XACML or similar policy description language (policy language) is used in ABAC.

## 3.2 Dynamic and History-based Access Control

The Chinese Wall security policy presented in [64], was one of the first papers to discuss access control based on historical actions committed by a subject. Protected information is classified into groups and Conflict of Interest (COI) classes. In the beginning, a subject is allowed to access any information from any group because there are no conflicts, but once some information is accessed, the next access requests are evaluated for any conflicts. For instance, if there is a conflict of interest between the data requested and data already accessed, then the request is denied, otherwise, the request is granted. This type of policy is called the Chinese Wall security policy.

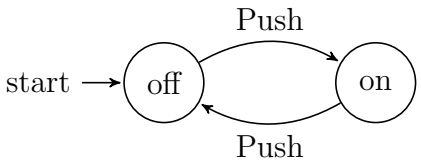



Figure 3.2: An example of finite automaton modeling an on/off switch [2]

In modern computer programs and applications we see multiple computations are handled simultaneously without affecting the final outcome of the individual program, this is called *concurrency*. Every program is designed to accomplish a set of computations to reach the final state with a set of inputs. Automata theory is widely used to study such sequences, especially in modeling and proving some properties in concurrent programs. This section focuses on the application of automata theory in computer security with a special focus on how particular types of automata can be used to enforce security policies.

> **Example:**
>
> A simple on/off switch is modeled using a finite automaton and shown in Fig. 3.2. The device has two states: *on* and *off* state. When the user presses a button, depending on the existing state, the state changes from one to another. For example, if the current state of the device is *on*, then after the press of a button, the state changes to *off* and vice versa.

A *safety property* sates that "some *bad thing* does not happen during execution" and *liveness property* stipulates that "a *good thing* happens during execution" (see [75]). In practice, the examples of safety properties are mutual exclusion and deadlocks; and the examples of liveness properties are starvation freedom, termination, and guaranteed service.

### 3.2.1 Security Automata

Fred Schneider in [15] introduced the "Security Automata", and discussed the class of security policies that can be enforced with mechanisms that work by monitoring system execution. This work concludes that safety property is enforceable and liveness properties are not enforceable. Also, for a mechanism to be enforceable, all the information required by the program must be observable by the execution monitor or commonly known as the *reference monitor* of the security automata. This works very well in operating systems, for instance, if any transition state tries to violate the safety property, then the security automata with the help of the reference monitor stops the program execution.

**Example:**

Consider an e-commerce scenario. If a customer pays for a service, then the service owner should give service to a customer. Given this scenario, it is possible to enforce (a) the service owner must provide the service only after the customer pays which is a safety property, however, (b) it is not possible to enforce that the owner gives the service for the customer after payment which is not a safety property but a liveness property.

### 3.2.2 Service Automata

R.Gay et al., in [17] presented *service automata* which extended the concepts of security automata [15] to be applicable in distributed systems with the help of *coordinator* that can communicate with other service automata and enforce the local policy by using the traditional *reference monitor*. The primary goal of this work was to enable decentralized enforcement in a coordinated fashion. The coordinator uses local policy to check if the action is allowed, if it not sufficient, it can delegate to some other service automaton.

### 3.2.3 Edit Automata

Ligatti et al., [76] presented *edit automata*, abstract machines that examine the sequence of application program action and transform the sequence when it deviates from a specified policy. The sequence of the program can be transformed into three different automata:

- *truncation automata* which can only terminate applications
- *suppression automata* which can terminate applications and suppress individual actions, and
- *insertion automata* which can terminate and insert other automata - i.e, another set of new sequences.

This chapter summarized the evolution of access control systems, and how automata theory is used to enforce security policies.

## 3.3 Common Access Control Protocols & Frameworks

Several protocols such as Kerberos, OAuth 2.0, SAML, etc. were developed to provide and support authentication and authorization. The discussion usually involves

several entities or actors. In one end there is a resource (*resource server (RS)*) and the owner (*resource owner (RO)*) of this resource. On the other end, an entity (*a client (C)*) wants to access this resource. Authorization of the client is handled by the *authorization server (AS)*.

The OAuth 2.0 protocol is one of the widely used authorization protocols because it is flexibility and interoperability with a wide range of applications. This thesis discusses OAuth because it can be used together with other generic access control mechanisms such as RBAC and as well as the proposed WFAC framework.

### 3.3.1 OAuth 2.0 Authorization Framework

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. In OAuth 2.0 [48] context, an authorization server provides an OAuth access token (representing specific permission) to the client to access a resource in the server.

### 3.3.2 Extensible Access Control Markup Language (XACML)

XACML is an Organization for the Advancement of Structured Information Standards (OASIS) standard. XACML is primarily used for implementing Attribute-Based Access Control (ABAC), but it can also be used to implement Role-Based Access Control (RBAC) as a specification of ABAC. A simple flow of XACML as specified in the Request for Comments (RFC) (see [77]) and is shown in Fig. 3.3. A brief explanation is presented below.

- A user sends a request which is intercepted by the Policy Enforcement Point (PEP)
- The PEP converts the request into an XACML authorization request
- The PEP forwards the authorization request to the Policy Decision Point (PDP)
- The PDP evaluates the authorization request against the policies it is configured with. The policies are acquired via the Policy Retrieval Point (PRP) and managed by the Policy Administration Point (PAP). If needed it also retrieves attribute values from underlying Policy Information Point (PIP).
- The PDP reaches a decision (Permit / Deny / Not-Applicable / Indeterminate) and returns it to the PEP

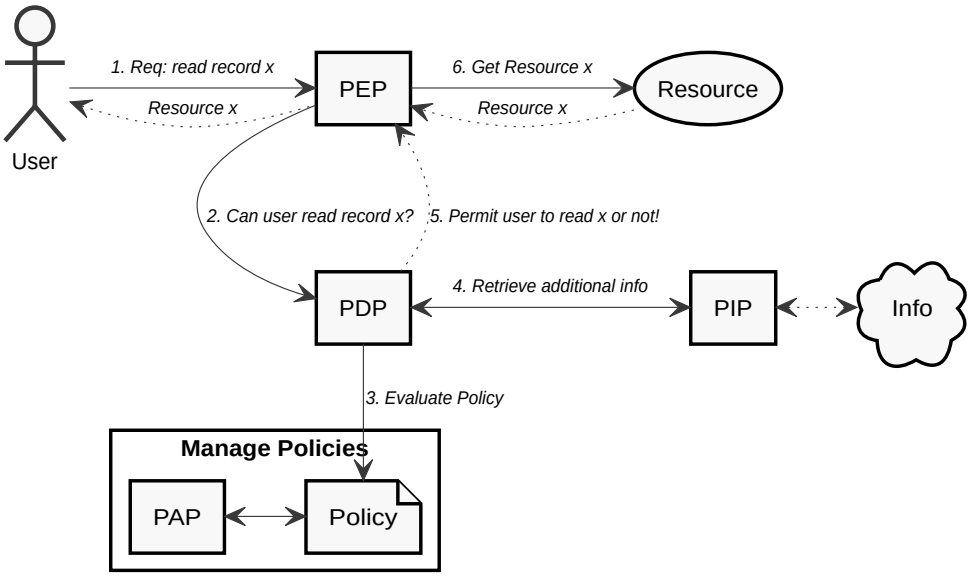



Figure 3.3: An example XACML Authorization Flow

# Chapter 4

# Modeling Workflows

This chapter presents different modeling techniques and systematically analyzes them. The advantages and disadvantages of each method are discussed in Sec. 4.4. Finally, based on the requirements presented in Chap. 2 a modeling methodology is selected. In particular, Sec. 4.4.1 presents the advantages of the proposed method.

The two widely used concurrency theory approaches for modeling processes are the net theory and the calculi theory. Their applications include modeling communication between parallel processes, detect deadlocks, choice of actions, etc. They are primarily used for specification and verification of properties and conditions in a process.

## 4.1 Petri Nets

In the net theory, one of the ways to represent a process is using *labelled Petri Net*. Petri Nets have evolved over the decades since their inception in the year 1966. In traditional Petri Nets (PN) (see [78]), there are places, tokens, and transitions. Petri Nets have the advantage that many properties such as liveness (deadlock-freeness), reachability are easy to verify (see [79, 80, 81]).

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. The classical definition of a Petri Net (P/T net) [34, 33] is:

**Definition 4.1.1** (Petri Net Tokens)**.** *If a token contains some arbitrary data, then it is referred to as colored tokens else as a black token. In general, a token in a Petri Net may or may not contain data. A token moves from one place to another via a transition. A transition consumes the tokens from the input place and produces the same or new tokens in the output places. The tokens show the current state of the Petri Net via Markings.*

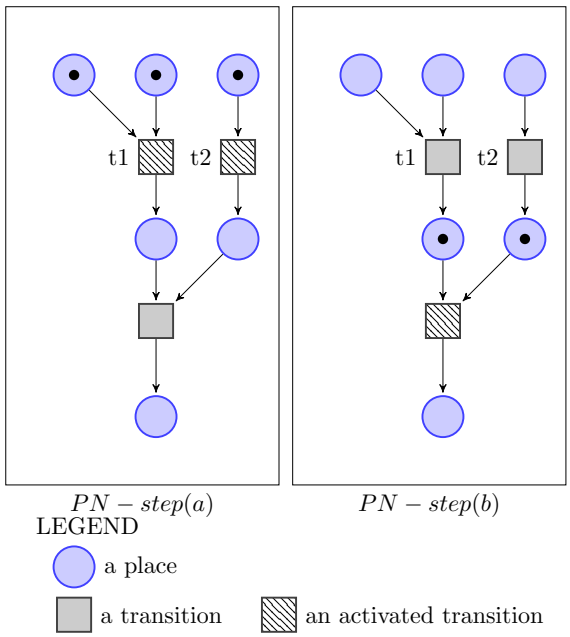



Figure 4.1: PN-Step(a) shows the initial state of a Petri Net and PN-Step(b) shows the state of the Petri Net after transitions *t1* and *t2* have fired.

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

- $P$ is a finite set of places,
- $T$ is a finite set of transitions $(P \cap T = \oslash)$
- $F \subseteq (P \times T) \cup (T \times P)$ 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 $M : P \to N$ that assigns to every place $p$ the number of tokens M $(p)$ that reside in $p$. We use the notation $\bullet t$ to denote the set of input places for a transition $t$; similarly, $t\bullet$, $\bullet p$ and $p\bullet$. Figure 4.1 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, resource optimization, etc. For a deeper understanding of Petri Nets, we recommend the book of Reisig [34] to the readers. We briefly present the most important extensions of Petri Net relevant for our work below.

### 4.1.1 Timed Petri Nets (TPN)

Timed 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 [82]). The classical Petri Net is not capable of handling this.

### 4.1.2 Colored Petri Nets (CPN)

Colored Petri Nets (CPN) is an extension of Petri Nets where different types of tokens can exist in the same place (see [83, 84]). In a colored Petri Net, each token is represented by specific colors (types). CPNs have the same kind of concurrency properties as Place/Transition Nets. Different tools such as CPN-Tools [85] are available to model and validate concurrent systems.

### 4.1.3 High-level Petri Nets

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 an in-depth description of 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 [86, 87]).

### 4.1.4 Workflow Nets (WF-net)

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 mapping workflow concepts such as task execution, synchronization (split and join) actions, etc. into Petri Nets (see [33, 31]). Workflow Nets showed that Petri Nets could 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.

### 4.1.5 Open Petri 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 [88]). One

of the goals of this work is to support multi-tenancy, i.e., to support activities, tasks from different organizations. The *Composition* is a common approach in software engineering i.e., to assemble small systems into larger ones. Reisig in [34] 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 [79, 31, 81].

## 4.2 Automata and Process Algebra

**Definition 4.2.1** (Automaton)**.** *"An automaton consists of a set of states, actions, transitions between states, and an initial state. Labels denote the transition from one state to another" see [32].*

Many specification models have adopted automata to express and validate the system behavior because they can represent safety properties for verification in finite state spaces using Linear Temporal Logic (LTL) (see [89, 90]). In particular, Process algebras are a diverse family of related approaches to formally modeling concurrent systems and their semantic foundation is based on automata (see [32]).

Mathematical models such as calculus are used for describing the statements about (e.g., atomic actions of) processes that are formulated in algebraical language. Also, these precise mathematical models are also used for specifying and validating (e.g., verification of certain properties) the processes. For instance, in *trace theory*, a process can fully be determined by a sequence of atomic actions (*traces*) that it can perform (see [91]). Communicating Sequential Processes (CSP) introduced by Hoare in [92] discusses concepts of synchronous communication channels between different processes, parallel composition, and hiding some internal communication. In CSP, the processes are represented as *failure sets*. A set of all failure states paired with refused states of the process is called the failure sets of the process. For example, if a process enters an unwanted state (e.g., a deadlock which is considered to be observable) in an experiment after a sequence of trace, then this an example of a failure state. Calculus of Communicating System (CCS) introduced by R. Milner in [93] represents a process by a *synchronization tree* which is also known as the *state transition diagram* or *process graph*.

**Definition 4.2.2** (Process Algebra)**.** *"The systematic exploration of (families of) algebraical calculi is called* process algebra*" [91].*

Several approaches use *Petri Nets* or *Process Algebra* for their inherent benefits. In [94], the author studied both approaches for reasoning complex systems, and finally uses the Petri Net theory approach for modeling workflows.

## 4.3 Business Process Modeling

A survey of formal verification approaches for business process modeling and their shortcomings is presented in [32]. The basic approach for analyzing the business process models is to translate them into other approaches such as the Process Algebra or Petri Nets. When translating a process specified in Business Process Model and Notation (BPMN) to Petri Nets, the chances of misinterpretations are high because of semantics used in BPMN. In addition, there is an additional overhead whenever the process is updated i.e., the updated BPMN process must be translated again and checked via Petri Nets. This shows that the semantics of Petri Nets are simple and enough for specifying processes and validating them.

The Object Management Group (OMG) developed the BPMN [95] standard for business processes diagrams which is intended to be directly used by the stakeholders who involve in designing, managing and realizing the business processes in organizations. The main motivation is to have a unified notation language for representing business processes. The OASIS developed the Web Services Business Process Execution Language (WS-BPEL) which uses Extensible Markup Language (XML) to describes business process activities as Web services and defines how they can be used to execute those tasks. Similarly, the Open Group consortium developed ArchiMate [1], a standard that describes a graphical language for modeling enterprise architectures in three layers: business layer, application layer, and technology layer. The business layer of ArchiMate includes business processes modeling. Surveys of different visual modeling techniques and business process modeling methods are presented in the papers [96, 97]. For example, BPMN can be used to model business processes involving IoT devices and services (see [98]). However, as pointed out earlier existing modeling methods have complex semantics i.e., possibility to represent a business process in different ways, and this introduces further problems such as misinterpretations. Therefore, they are not optimal for representing access control in a distributed IoT environment. Thus, modeling techniques presented above are not optimal for addressing the requirements and goals focused in this thesis. Therefore, they were not adopted as a modeling and specification language.

## 4.4 Advantages and Disadvantages of different Modeling Techniques

The Tab. 4.1 shows the advantages and disadvantages of different modeling techniques discussed above. In addition, Sec. 4.4.1 presents the advantages of Petri Nets compared to other approaches.

[1] `https://pubs.opengroup.org/architecture/archimate3-doc/`

Table 4.1: Advantages and Disadvantages of Modeling Techniques

| **Modeling-Approaches** | **Advantages** | **Disadvantages** |
|---|---|---|
| Process Algebra and Automata | - Strong formal specification and verification techniques<br>- Strong research background and community<br>- Ability to model and verify concurrent processes<br>- There exists real world implementations, for instance, in operating systems<br>- Complexity can be handled via Compositional approaches | - Requires mathematical knowledge to model and understand<br>- Popular in academia only<br>- Not user or practitioner friendly |
| Business Process | - Graphical view and interface - easy to understand and model processes<br>- Used in many commercial business process modeling tools<br>- Complexity can be handled via Compositional approaches<br>- Popular in industry | - Does not have strong verification techniques by default<br>- Ambiguity in describing the same process in multiple different ways and notations<br>- Does not offer capabilities to simulate, validate whether a process ends, and so on<br>- Only a graphical notation to describe processes and cannot be used to execute the process itself |
| Petri Nets | - Graphical view and interface - easy to understand and model processes<br>- Ability to simulate the process via token-game semantics | - Detailed modeling of business use cases can get complex |

Continued on next page

Continued from previous page

| Modeling-Approaches | Advantages | Disadvantages |
|---|---|---|
| | - Strength in modeling inter-process communication and synchronization of concurrent processes<br>- Strong formal verification abilities i.e., ability to verify whether a process ends, validate whether the process has any dead place with markings, etc.<br>- Popular both in academia and industry - used for network protocol verification<br>- Availability of actively developed open source tools to model and validate Petri Net models<br>- The semantics of the Petri Nets can be implemented directly with the help of security tokens like OAuth-tokens<br>- Complexity can be handled via Compositional and Hierarchical Petri Nets | |

### 4.4.1 Advantages of Petri Nets over other approaches

This thesis adopts the Petri Nets modeling approach because of its simplicity, usability, synchronization (compositional) ability, flexibility, expressivity, and the possibility to perform formal verification analysis.

This thesis also presents important extensions of Petri Net that help us to specify and verify workflows. By enforcing verified workflows combined with fine-grained access control, this enables us to provide assurance in workflow-aware access control. The advantages of using Petri Nets for specifying and modeling workflows are the following:

- Petri Nets use the graphical interface to model concurrent process and therefore, they are easy to understand even by a business person without a strong mathematical knowledge
- Petri Nets are intuitive and the process can be simulated and verified with the help of *token-game* semantics.
- Petri Nets are amicable to formal verification.
- Petri Nets have a strong notion of inter-process communication and synchronization features, i.e., inter-process communication and composition of workflows can be modeled. This is usually an important requirement for modeling complex processes.

On the other hand, Process algebras require good mathematical knowledge to specify a process, and to understand the models, different states of the workflow, and to verify them.

Widely used approaches for modeling concurrent processes were presented in this chapter. The advantages of using Petri Nets over other approaches were presented. As Petri Nets have more advantages over other approaches this thesis uses Petri Nets as for modeling, specifying, and validating the processes or workflows.

# Chapter 5

# Internet of Things (IoT)

**Definition 5.0.1** (Internet of Things)**.** *Internet-of-Things (IoT) is defined as "[..] a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual* things *have identities, physical attributes, virtual personalities, use intelligent interfaces, and are seamlessly integrated into the information network" [99].*

IoT devices usually have multiple sensors and actuators attached to them to collect information from, and interact with, the environment where they are deployed. IoT devices are categorized by their ability to process and store data, energy consumption, and communication capabilities (see [41]). In particular, the most common constrained IoT are classified into three groups: *Class 0* devices are really constrained sensor-like motes with less than 10 Kilo Bytes (KB) of Random Access Memory (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 Hypertext Transfer Protocol (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 CoAP for Internet connectivity.

Constrained IoT devices have several advantages, they are cheap, compact, easy to deploy, and consume less energy. The advancements in the semiconductor field enabled the development of increased processing, storage capabilities and at the same time energy efficient hardware. In addition, mass manufacturing may decrease the overall cost of the device itself. Most of the IoT applications use data collected from the IoT devices to get insights, predict, and 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.

In general, for an IoT device to communicate with other devices whether or not it belongs to a different owner - its operating system and hardware capabilities should be able to run interoperable communication and security protocols as described in Sec. 2.2. This is an important and a basic requirement for any IoT devices to be integrated with other systems such as a workflow platform or a middleware through which other systems can communicate, access, and control the IoT devices.

## 5.1 Issues in Constrained IoT Devices

As Constrained IoT devices used in various applications they are more exposed and therefore, vulnerable to attacks because existing state-of-the-art security mechanisms do not fit within the constrained devices. Often, these devices have poor physical and software security and can be easily compromised by a motivated attacker. For example, modern security mechanisms are difficult to achieve in constrained IoT devices because of the lack of memory space and processing power (see [100, 43]). Implementing secure key generation and key storage in constrained IoT devices are hard (see guidelines from Trusted Computing Group [101]) because these devices lack sufficient entropy to generate random numbers and are prone to side-channel attacks. Several attacks on industrial IoT devices are also presented in [43]. A report from European Union Agency for Network and Information Security (ENISA) also presents similar issues in IoT but with a special focus on critical information infrastructures (see [6]). Due to their inherent weakness and vulnerabilities, hackers frequently target IoT devices to escalate attacks on valuable assets. 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 [102]) has enforced strict regulations for handling private information of users.

In IoT applications, multi-tenant architecture is more prominent and in such applications, different entities provide and consume services from one another and each entity might want to enforce their own integrity, confidentiality or functionality goals on other entities consuming his service.

One of the main problems with multi-tenant, distributed systems and architecture is the "trust problem": each participating entity [1] in such a distributed system

[1] An "*entity*" may refer to the owner of the resource, or a client using that resource in this context. For instance, a resource owner might want the client to authenticate, then do some action,

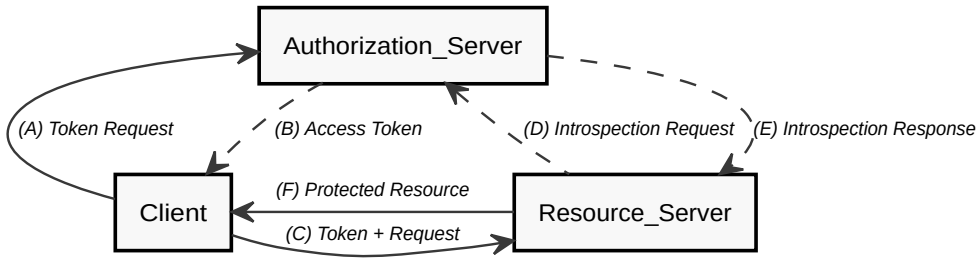



Figure 5.1: ACE OAuth 2.0 Flow [3]

requires that the others behave in a particular way in order to achieve their goals. Ideally, an entity would like to specify a "contract" that declares her assumptions about the behavior of other entities as well as the guarantees that she offers to them about her actions. But how can she trust other parties to behave according to the contract? How can she be sure that they do not "cheat" her? This thesis also investigates a way to automatically enforce such a contract. Each party imposes his rules on entities while they interact with his services. In general, those sequences of interactions defined are a set of allowed actions, or in other words, a workflow. 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 (see [103, 104]) in a workflow is also the key for its validation and verification.

## 5.2 Authorization for Constrained IoT Devices

Authorization mechanisms are important to restrict or allow an entity to access a resource in an IoT device. The IETF working group (WG) Authentication and Authorization for constrained devices (ACE) [47] is specifying a framework for authentication and authorization in IoT environments called "ACE-OAuth" [3]. ACE-OAuth is based on OAuth 2.0 and Constrained Application Protocol (CoAP) and its message flow is shown in Fig. 5.1. To describe the ACE-OAuth actors, let us consider the following example.

> **Example:**
>
> 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

and then log her actions before accessing the resource. On the other hand, the client might want the guarantee that after authenticating, and doing some action she must be guaranteed that the resource will be granted.

> 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 (CL), the smartwatch is Resource Server (RS), the cloud service is the Authorization Server (AS), and John is the Resource Owner (RO).

In Fig. 5.1, the messages exchanged are the following:

- (A) Requesting an Access Token: The CL performs an access token request to the token endpoint at the AS. The client may include authentication credentials (e.g., client credentials) according to the OAuth 2.0 specifications, and information about what resources it wants to access from RS.
- (B) Access Token Response: If the request from CL was successful, then AS returns an access token to the CL with optional additional information such as Proof-of-Possession (PoP) access token extension. The AT issued can also be just a bearer assertion. The Framework also defines parameters that inform the client about RS capabilities and the profiles supported by the RS.
- (C) Resource Request: The CL performs a resource request to RS. Depending on the resource server limitations and the chosen profile and communication protocol, the resource request exchange may differ. Usually, this request contains the access token and other information obtained from the AS in step B.
- Messages (D) and (E) are optional in ACE-OAuth. The RS must be online or should have direct communication to its AS to make introspection request. Introspection request is usually not necessary when the access tokens are self-contained RS can validate on its own.
- (F) Protected Resource: When CL's request is authorized, then RS uses the dynamically established keys to protect the response according to the communication protocol and the security configurations specified in the profile.

# Chapter 6

# Distributed Ledger Technology

A chain of blocks that guarantees the integrity and immutability of data inside every block is known as the blockchain. Each block is linked to each other using a cryptographic hash of the previous block, a timestamp, and transaction data. The first cryptocurrency blockchain introduced was Bitcoin [105], but the ideas of a blockchain originated from the work of Haber and Stornetta in 1991 [106], where the authors presented a mechanism to time-stamp documents (any digital file such as a small word document or a big video file) by linking them together with the help of one-way hash function.

## 6.1 Blockchain

Blockchains are currently used in different applications, in particular, they are mainly used to create distributed ledgers that ensure the immutability of data, data redundancy, provenance and transparency of records to the participants and users of the blockchain. The records often referred to as transactions are stored in an append-only log called a *ledger*. The ledger is replicated across a peer-to-peer blockchain network where each node participating in the network maintains a consistent copy of it, thus forming a distributed database of all transactions. A consensus protocol such as "Proof-of-Work" is used to decide how new transactions are added to the blockchain and how to validate existing transactions.

The peer-to-peer network is made of different entities (node, full-node, etc) for instance, a full-node keeps a copy of all transactions of the blockchain. For the interconnection of nodes, an overlay network is used, where every network node is connected to an alterable set of neighbors in a peer-to-peer fashion. For the data distribution between neighboring peers, a flooding algorithm called *gossip protocol* is commonly used (see [57]). In general, based on the consensus protocol and the participants, a blockchain system can be categorized into two types: a *public* or a *private* blockchain. In a public blockchain network, anyone is free to join the blockchain network as a peer, and start contributing to the network by verifying transactions and maintaining a local copy of the ledger. In a private blockchain,

participants are restricted i.e., only a specific set of nodes is allowed to join the network which is governed by a predefined set of rules. Developers and software architects decide which type of blockchain system (either private or public) and consensus algorithm to use based on the requirements of the use cases.

Satoshi Nakamoto in [105] introduced a cryptocurrency known as "Bitcoin" that uses blockchain to create an immutable distributed ledger for storing integrity protected cryptocurrency and its transactions. Bitcoin introduced a consensus mechanism to solve the double-spending problem without having a trusted authority (i.e., a typical Banking authority) to check every transaction. Bitcoin is a classic example of public permissionless blockchain, it revolutionized the financial industry and it is termed as "Blockchain 1.0" in [107]. A very good introduction to blockchain, Bitcoin, and other cryptocurrencies based on blockchain technology is presented in [57].

## 6.2 Smart Contracts

Smart contracts (SC), introduced by Nick Szabo in [108], become popular with the advancements in blockchain technology, termed as "Blockchain 2.0" in [107]. Smart contracts are often written to ensure fairness between two or more participating entities without a trusted third-party. We explain the differences between a smart contract and a typical judicial system's written contract. In the judicial scenario, two or more parties agree to a written contract, and bind to those contractual conditions. In case of a dispute, the judicial system (the trusted authority by the involved parties) solves the dispute with fairness. Similarly, in the smart contract setup, the owner creates the smart contract and publishes it in the blockchain. The entities that agree with the SC conditions interact with it. The following conditions hold for any generic smart contract on a blockchain. First, depending on the underlying blockchain systems, updating a smart contract after it is published on the blockchain is not possible or trivial even by the smart contract owner. For instance, in most public Blockchain systems, once a smart contract is published then it cannot be updated (i.e., business logic cannot be changed) even by the owner of the smart contract. Many permissioned blockchain systems, for instance, Hyperledger Fabric (HLF) allows upgrading smart contracts by satisfying predefined smart contract upgrading policies e.g., other members or majority of the involved participants should approve for upgrading the smart contract. Second, the SC conditions are enforced by the code written and the underlying blockchain system's consensus mechanism and policies. Therefore, this ensures fairness to the involved parties without involving a trusted third-party (see [109]).

Smart contracts (SC) are arbitrary computer code (similar to a programming language's code) expressing one or more business logic. Unlike a typical computer code, smart contract code is deployed in a blockchain network, it is executed and

the results are verified by the nodes participating in the blockchain network. In Bitcoin, a simple stack language is used to express the rules and conditions for a successful transaction, validating double spending, how new coins are produced and consumed. Therefore, it is vital that the smart contracts deployed are correct and behave in a deterministic way. There are several incidents where bugs in a smart contract code were exploited by adversaries for their benefit: one famous incident is the hack on the Decentralized Autonomous Organization (DAO)'s SC deployed in Ethereum blockchain network (see [110]).

Different implementations of blockchain exist such as Ethereum, Hyperledger Fabric, Hyperledger Sawtooth and others. They all provide a blockchain based distributed transaction-processing platform that allows the implementation of business logics as smart contracts. But they use different features such as smart contract specification language, blockchain type, and consensus mechanisms. Ethereum and similar blockchain implementations refer to the code written to enforce the business logic as *smart contract* [111] whereas, the Hyperledger Fabric refers to the code as *Chaincode*. Ethereum uses a specially designed smart contract programming language called *Solidity* [109] whereas for example, Hyperledger Fabric uses standard programming languages such as Go, Node.js, and Java to write chaincode.

In [112], the authors presented the problems of using Turing Complete languages for writing smart contracts and some methods to make SCs less vulnerable. Solidity and chaincode are Turing complete because they use Turing complete languages to write smart contracts. Researchers and organizations are working towards addressing these problems in smart contracts for example, in [113, 114], the authors explain methods to create safe smart contracts. Security start-ups and established companies are offering services to audit smart contracts and ensure that no vulnerabilities exist before deploying SCs on the blockchain network.

# Chapter 7

# State-of-the-Art Analysis

This chapter presents different methods and technologies that can be used to partially address the problems identified in the distributed IoT scenario and some of its challenges and requirements were presented in Chap. 1 and Chap. 2.

The brief description of the distributed IoT scenario is the following: IoT is used in many applications areas where these devices are distributed and more often, owned by different owners. Therefore, the owners of the IoT devices have their own rules also, the entities (who want to access the services provided by the IoT devices) must obey and follow the rules enforced by corresponding owners. In such scenarios, the owners may not trust each other therefore, we need an approach that works in such distributed scenarios. This thesis presents an abstract version of the mobility and car sharing use case presented in Sec. 15.2 and 15.3 respectively to perform state-of-the-art analysis. In the following sections, this use case is referred to as the *mobility use case* where multiple mobility providers share their mobility service to provide a comprehensive mobility solution to end users. Each mobility provider wants to enforce his own rules, e.g., if a user rents a car from company-A, then the user should follow a particular workflow to reserve, rent, use the car, and finally, to return and park the car in a predefined geo-fenced area. Similarly, for other mobility providers such as trains and flights, the rules could be different. And, in case of a dispute, or error, the method should be able to solve the issue using distributed accountability without relying on a single trusted entity.

To our best knowledge, no solutions exist as of now which takes into consideration all the challenges and requirements focused on this thesis (see Sec. 2) and is able to solve the described distributed IoT scenario. This section presents an analysis of different mechanisms, technologies, and tools that could be used to partially address some of the challenges independently.

## 7.1 Legal and Contractual Frameworks

In case a conflict arises between two or more parties in the presented mobility use case, e.g., the user fails to follow part of the car rental process a legal contractual framework is used to resolve it. Usually this is a time consuming and complex process, involving a trusted third-party like an insurance agency that performs an investigation to identify the root cause and resolution.

A legal contract describes the rules that must be followed by the participating entities and the penalties to be applied in specific cases. In this approach, all involved entities are assumed to trust the legal process and the judicial/insurance system - which is open, not biased, and decides solely based on contract and the applicable legislation. It is possible to specify a business use case and conditions in the form of text. Such text is usually based on natural language and is subjected to interpretation. More often, a text description is not as precise as digitally specified contracts - where the chance of misinterpretation is low.

A natural language contract requires a manual auditing process involving technical and legal experts that consumes a lot of time to resolve disputes. In addition, this process is very expensive as it is now. As described, the legal and contractual frameworks are alone not sufficient to solve all the requirements and challenges faced in the distributed IoT scenarios. However, it is usually used as a compliment to technical solutions even now. The technical details on how each owner implement their IoT devices and enforce the agreed conditions are usually not described in such legal contracts. In general, IoT devices (e.g., in the mobility use case, the car) should have inter-operable protocols such that they could interact with other devices that are owned by different participating entities, e.g., the smartphone of the user renting the car. This thesis presents the WFAC framework to enforce the contractual conditions digitally and semi-automatically.

## 7.2 Workflow Management Systems

Existing workflow management systems were built for managing and automating workflows that include repetitive tasks using a centralized approach. Also, those workflow systems focus on enforcing centrally governed policies within an organization via workflows or business processes - usually, using systems controlled by the same organization. This is different from the scenario this thesis is focusing on where different owners may have different goals and they work collaboratively.

In the mobility use case, as multiple mobility providers are involved - such as a train service provider e.g., Deutsche Bahn; a long term car rental provider e.g., Sixt; and a short term car sharing providers e.g., Sharenow - they want to enforce different restrictions (i.e., workflows) on the users using their services and do not

want to share their customer data to one centralized service. However, they want to make use of a platform with which integrated mobility service can be provided.

Existing centralized workflow management systems are not suitable for this. Commercial workflow management systems do not focus on constrained IoT environment or the scenarios with accountability, error-recovery, and validation of workflows that this thesis focuses on. For instance, SAP's Business Workflow Systems [1] primarily focus on Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) systems, and how they can be integrated with internal systems such as Human Resource (HR) systems to increase efficiency and productivity. However, with the latest advancements in workflow systems, with additional add-on techniques such as SAP's WebFlow Function [2], one organization can send XML documents with workflow specific info to other organizations where the workflow could be continued and extended.

Also, existing workflow management applications are resource intensive and need powerful servers to design, govern, execute, and manage the workflows. In addition, the main focus of these workflows is to ease repetitive tasks and not to guarantee workflow integrity or not to enforce access control integrated. In addition, internal HR management systems provide the role functionality, with that RBAC is used together with a workflow. Therefore, a State-of-the-Art analysis of commercial workflow systems was not performed within the scope of this thesis.

## 7.3 Access Control Mechanisms

Access control mechanisms such as Attribute-Based Access Control (ABAC) can be used to restrict access to a resource based on historical events. In order to do this, the ABAC's PIP should have access to historical events in real time, based on which the PDP and PEP may either grant or deny access to the resource. Recent researches including, [115, 116] explored how to propagate concurrent state information across distributed databases for PDPs to make access control decision. Decat et al., in [115] introduced a protocol to exchange concurrent information using distributed coordinators. In this protocol, each subject and resource in question identifies a unique coordinator from the list of coordinators. Each coordinator contains a set of workers nodes that update and query an attribute database e.g., MongoDB or Cassandra that is designed as eventually consistent and scalable. Race conditions are avoided by ensuring that the same coordinator is used for policy evaluation before or even after a conflict resolution procedure. In [116], the authors introduce FACADE (Fast Access Control Algorithm with Distributed Evaluation)

[1] https://help.sap.com/viewer/a602ff71a47c441bb3000504ec938fea/7.5.6/en-US/4f41e8a0dd89535fe10000000a421937.html

[2] https://help.sap.com/viewer/a602ff71a47c441bb3000504ec938fea/7.5.6/en-US/4f371476f5d95541e10000000a421937.html

which uses a similar architecture of [115], but their algorithm provides low latency and higher throughput compared to [115] approach.

This is challenging in the thesis scenario, where IoT devices might be constrained, connectivity to a common database might not provide the best suitable approach where different IoT owners might not want to have a single concentrated source which has access to all historical events i.e., PDP of ABAC needs a PIP to make access decisions. In addition, the ABAC should have workflow-related attributed such as tasks sequencing, in practice, ABAC is not designed to handle this kind of stateful access control. The number of static rules and the generic attributes including the workflow-related attributes - that need to be configured to enforce a particular workflow - at some point in time becomes unmanageable. In addition, without strong modeling and validation of workflow rules in ABAC, a combination of certain rules may lead to unintended access / or deny-of access, therefore, a policy validation or verification tool must be used whenever a rule is updated. Current research and some of the related open issues in ABAC are presented in [117].

# Chapter 8

# Related Work

In this chapter, detailed related work to this are presented and they are grouped into three different categories (a) workflow specification, (b) Internet-of-Things, and (c) smart contract generation.

## 8.1 Workflow Specification

Extensive work on the specification and enforcement of workflows can be found in the literature; in particular, Bertino et al. [22] 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 [118] calls them "Dynamic access control"), but this requires a centralized WF enforcement engine. Basin et al., [119] model the business process activities as workflows with a special focus on optimizing the authorizations permissions. Van der Aalst in [31, 33] studied how Petri Nets can be used to model complex workflows and verify some of the workflow properties. Atluri et al., [120, 21] studied how to model workflows using Petri Nets, but did not describe the implementation details. Huang et al., [121] presented a web-enabled workflow management system, and Compagna et al., [122] presented automatic enforcement of security policies based on the workflow-driven web application, but both works presented a centralized architecture. Reiko Heckel [88] showed how Open Petri Nets are suitable for modeling workflows spanning different enterprises. No existing work discusses how to handle error conditions during workflow execution, support or integrate practitioner-friendly design and specification tools, enforce 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.

Mortensen [123] presented a method for automatic implementation of systems based on 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.

Lodderstedt, Basin, and Doser in [124] presented SecureUML, 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, Jan Jurjens [125] presented UMLsec (an extension of UML) for secure software development.

Linhares et al., in [126] presented an empirical evaluation of OMG SysML's to model an industrial automation unit using the open source modeling tool Modelio [127] but not in the context of modeling workflows for access control.

Wolter et al., [128] 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.

## 8.2 Internet of Things

Access control solutions for IoT systems have been extensively researched. Some of the issues in access control in IoT, and a Capability-Based Access Control (CapBAC) approach is presented in [129], where the authors investigated how such CapBAC system can support delegation, revocation, and granularity. However, their approach is implemented using Java and has not been evaluated on IoT devices, and do not focus on workflow-aware or history-based access control. A similar capability-based model intended for federated machine-to-machine communication in IoT networks is introduced in [130], in this approach, in addition to capabilities, context-aware information i.e., a context attribute is combined to accommodate dynamic access policy enforcement. In Capability-Based Context-Aware Access Control (CCAAC) [130], the authors focused on introducing a capability propagation protocol for authority delegation. However, these capability-based approaches did not focus on history-based access control.

In [19], an HCAP system is proposed for enforcing permission sequencing constraints in a distributed authorization environment. 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.

The History-based Capability systems for IoT (HCAP) approach extends the concepts of Security Automata to the IoT environment, but the HCAP approach has its limitations because of the following reasons:

- (a) the IoT (or Resource Server (RS)) needs to maintain the state information locally and has to synchronize the state information to other devices and the Authorization Server (AS) in the network,
- (b) when some messages exchanged are lost then it brings also additional overhead to the introduced protocol,
- (c) besides, this approach does not focus on establishing a language for workflow specification, or to provide accountability features,
- (d) as it is based on process algebra, it is not practitioner friendly[1].

Previously mentioned approaches do not support accountability in a decentralized and trust-less environment. To tackle this, other researchers proposed to use blockchain-based access control mechanisms. In [23], a blockchain-based access control method is presented and in their approach the attribute-based access control policies and concepts are used in the bitcoin blockchain system. In [131], a survey of blockchain-based access control solutions and their challenges are presented. In this section, the solutions that focus on IoT and are relevant to this thesis are discussed. In [24, 25] a smart contract-based access control solution for IoT is proposed. In [24], each smart contract may contain only one access control rule i.e., subject-object pair with associated rights, in addition, to track misbehaviour of resource access, each resource maintains a list that contains the subject that has accessed the resource with timestamp info and if the subject already had any penalty or not. Moreover, the implementation was based on Ethereum Blockchain and Raspberry Pi devices were used as IoT devices. In contrast, in [25], a single smart contract may contain many access control rules and this approach uses a *management hub* as a proxy component between the IoT devices. The management hub could be a powerful device to handle complex computational tasks that the IoT devices might not be able to perform. In [132], the authors present a novel attribute-based access control scheme for IoT using blockchain, the proposed scheme uses the elliptic curve cryptography to create public/private keys and the requester proves to the resource server that he/she has access to the subset of the attributes by signing appropriate request-info it with the private key he/she holds. The resource server, then checks the signature with the help of information published in the blockchain and decides whether to provide access or not.

The authors of [133] introduced 'ControlChain' to restrict access control in IoT with the help of four different blockchains: Context Blockchain, Relationships

[1] "Despite its strengths, after almost 30 years the use of process algebra outside the academia is still very limited. Due to some of its technicalities, process algebra is perceived by practitioners as being difficult to learn and use …" [20].

Blockchain, Rules Blockchain, and Accountability Blockchain to store context information, relationships, access control rules, and accountability information respectively. To support different access control models such as RBAC, ABAC, and CapBAC, they propose to use a decoder that converts required access control policies into compatible rules - which is a combination of ACL, capabilities, and attributes. However, the paper did not explain how it can be done without affecting the access control properties of different models.

Nevertheless, none of the blockchain-based access control approaches for IoT uses or proposes the concept of history-based access control concept i.e., the workflow-aware access control methodology and other challenges such as practitioner-friendly modeling, validating smart contracts, and error-recovery features that this thesis focuses.

Other than the HCAP approach [19], so far, no other approach tackles the core-problem of workflow-aware or history-based access control in an IoT environment. In addition to that WFAC tackles i.e., accountability, a way to handle error conditions, or support integration of practitioner-friendly tools and support the interoperable methods such as OAuth in a distributed IoT environment. In addition, most of the other approaches do not focus on privacy issues (e.g., protecting the identity of the client) in situations where the IoT devices are not able to protect the communication channel between the client and the resource server.

A State-of-the-Art analysis of related work in terms of specification and enforcement of distributed workflow execution and access control that supports the IoT environment is performed and a comparison is shown in Tab. 8.1. The description of the legend is as follows:

- (+) supports the marked feature;
- (-) does not support the marked feature; and
- (o) partially supports the marked feature.

This comparison is performed only with IoT context and distributed Workflow-Aware access control and features this thesis is focusing on.

Table 8.1: WFAC Related Work Analysis with overall distributed workflow and IoT Focuses

| Related Work | IoT Focused | Distributed Access Control | Distributed WF Execution | Process Aware | Practitioner Friendly | Error Resilience |
|---|---|---|---|---|---|---|
| WFAC ([26, 27, 28, 29]) | + | + | + | + | + | + |

Continued on next page

Continued from previous page

| Related Work | IoT Focused | Distributed Access Control | Distributed WF Execution | Process Aware | Practitioner Friendly | Error Resilience |
|---|---|---|---|---|---|---|
| Security Automata ([15, 16, 134]) | - | - | - | + | - | - |
| Service Automata ([17]) | - | + | - | + | - | - |
| HCAP ([19]) | + | + | o | + | - | o |
| Dynamic Access Control through PN ([118]) | - | - | - | + | + | - |
| Workflow Nets ([31]) | - | - | - | + | + | - |
| SecureUML ([124]) | - | + | - | + | + | - |
| Blockchain and Smart Contract-based access control ([24, 25, 133, 132]) | + | + | o | - | - | - |

## 8.3 Smart Contract Generation

This section presents the related work in the area of a) generating smart contracts - with a focus on security and b) translating business process models, state machines, Petri Nets, or similar workflow models into a generic code. The first approach (a) reviews the state-of-the-art approaches in generating smart contracts, and the second approach (b) reviews the related work in the area of code generation by translating business processes or workflows into a generic code. However, in this section, we do not cover the related work in the area of identifying existing vulnerabilities in already deployed smart contracts by symbolic code analysis or similar methods - this is out of the scope of our research.

In our research, we focus on avoiding the errors prior to smart contract generation. But integrating vulnerability analysis tools before deploying a smart contract will certainly help to find and avoid existing library or code-based security issues. In each paragraph below, we present one topic of related work and in the end, we discuss the problems and advantages of each method (if they exist). We considered

the experiences, knowledge, and recommendations from the presented related works for designing and implementing our approach.

Garcia et al., in [135] presented a method for compiling a Business Process Modeling Notation (BPMN) process model into a smart contract defined in the Solidity language. The authors used the approach of the control flow of *workflow-nets* from [136] to translate BPMN to Petri Nets, to eliminate invisible transitions and spurious places, and thereby, optimizing the gas costs for the deployed Solidity smart contract in Ethereum blockchain. Nakamura et al., in [137] presented a similar approach: first, they use BPMN to model inter-organizational business processes, but it is translated into state charts and represented as State Chart XML (SCXML); second, they optimize the state chart and then produce the chaincode (using Go language) for Hyperledger Fabric. The papers [135, 136, 137] did not focus their work to build secure smart contracts, which is one of the limitations of their work.

Mavridou et al., presented two frameworks in [138, 114, 139] respectively: (a) finite state machines (FSMs) based framework called *FSolidM*, which provides a graphical editor for specifying Ethereum smart contracts as transitions systems and a Solidity code generator; (b) *VeriSolid* framework, which introduces formal verification capabilities. Their FSolidM framework provides a graphical user interface (GUI) and has the ability to integrate custom plugins for developers to add further functionality such as automated timed transitions and access control. Thereby, their approach is very close to ours in terms of correctness-by-design development of smart contracts. As this work was done in parallel to ours and also suitable to generate secure smart contracts, we complement their approach to generate secure smart contracts. A separate study is required to compare our approach, which could be future work. However, the semantics of the state machines can be hard to understand compared to the semantics of Petri Nets. Furthermore, we argue that the semantics of Petri Nets are much more suitable than FSMs to model concurrent and interactive systems such as smart contracts.

Philippi in [140] presents an overview of different code generation approaches from high-level Petri Nets. In particular, they investigated the simulation-based code generation approach. Mortensen in [123] presented a method for generating code from CPN. First, the developed CPN model is used to generate the standard ML code then, the ML code is used to generate a platform-specific code, for example, in C language. They evaluated this approach by generating code for an access control system. Their work also shows that the introduced automatic code generation method reduces the code development time and errors in comparison to the manual system development methods. Pinna et al., in [141] analyzed the Bitcoin blockchain network using a Petri Net model to find the group of Bitcoin addresses owned by the same owner, disposable addresses and explained the advantages of Petri Nets modeling. However, this approach is not related to smart contract generation, but shows the advantages of using Petri Net modeling. The approaches taken in the papers [140, 123] show the advantages of using Petri Nets such as automatic

code generation, reducing code development time and errors, and evaluating the generated code for security systems like access control.

Choudhury et al., in [142] presented an automatic smart contract template generation framework that uses ontologies and semantic rules to encode domain-specific knowledge and uses the structure of abstract syntax trees (AST) to incorporate the required constraints into the generated template. The authors evaluated two use cases and presented a few examples of the generated SC code snippets in Solidity language. Tateishi et al., in [143] presented another approach to generate automatic smart contracts that can be deployed in Hyperledger Fabric, their approach uses a document template and a controlled natural language (CNL) that provides a formal model which is then used to generate the smart contract with a toolchain. Both the approaches [142, 143], point out that we need human-understandable methods that can help to create smart contracts, but they do not focus on generating secure or *error-free* smart contracts.

# Part III

# Contributions

# Chapter 9

# Actors and Definitions

This chapter introduces the actors and definitions involved in the WFAC framework and presents an overview of the contributions to be detailed in Chap. 10, 11, and 12.

## 9.1 Actors

This section presents definitions of actors involved in the proposed WFAC framework. Some definitions are taken from standard computer security textbooks, and standardization bodies, such as NIST, and when necessary, new definitions are introduced as part of the thesis.

**Definition 9.1.1** (Entity)**.** *"An individual (person), organization, device, or process. "Party" is a synonym" (see [144]).*

**Definition 9.1.2** (Principal)**.** *"A principal is an entity that can be granted access to objects or can make statements affecting access control decisions" (see [145]).*

**Definition 9.1.3** (Subjects and Objects)**.** *"A subject is an active entity within an IT system. The objects of access control are files or resources, such as memory, printers, or nodes in a computer network. […] Depending on circumstances, an entity can be a subject in one access request and an object in another. The terms subject and object merely distinguish between the active and passive party in an access request." (see [60])*

This thesis discusses actors involved in IoT using the terminologies used in the OAuth, and in addition, how the terms are used within the proposed WFAC context is explained below.

**Definition 9.1.4** (Resource Owner (RO))**.** *An entity capable of granting access to a protected resource in a Resource Server (RS) (see [48]).*

In the context of the WFAC, the Resource Owner (RO) is referred to as the actual owner of RS. Also, the RO can delegate his capabilities to its subordinate, also referred to as *authority* who then is capable of granting access to a protected resource in RS.

**Definition 9.1.5** (Resource Server (RS))**.** *The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens (see [48]).*

In the context of the WFAC, the RS is usually a constrained device, but it can also be a normal resource available via a web service.

**Definition 9.1.6** (Client (CL))**.** *An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices) (see [48]).*

In the context of the WFAC, the client is referred to as the application available inside a handheld or smart phone.

**Definition 9.1.7** (Authorization Server (AS))**.** *The server (typically, a powerful machine) issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization (see [48]).*

**Definition 9.1.8** (Owner)**.** *An entity, usually a person or an organization, who owns the object in the discussion. The object could be either physical or digital, e.g., a device or a workflow.*

**Definition 9.1.9** (Authority)**.** *An entity different from the owner of the object in the discussion (e.g., a resource server) and is capable of performing some privileged actions (e.g., granting access to a protected resource in the resource server) on the object.*

**Definition 9.1.10** (Stakeholder)**.** *Individual or organization having a right, share, claim, or interest in a system or in its possession of characteristics that meet their needs and expectations (see [146])*

In other words, an entity who may or may not be the owner of the object in the discussion (e.g., a resource server) and has some rights, powers, or interest to control the behavior or usage of the object.

**Definition 9.1.11** (User or Workflow Participant)**.** *An entity that is participating or executing the workflow by doing tasks defined in the workflow.*

## 9.2 WFAC Definitions

This section presents the workflow related definitions.

**Definition 9.2.1** (Workflow)**.** *A workflow describes a set of tasks that must be performed in a particular order, i.e., there are some constraints that determine which orders of the tasks are admissible and which are not. From a mathematical point of view, those constraints generate a partial order on the tasks. The workflow does not contain any loops. If a loop is required, then a new workflow could be started.*

A workflow (in the context of this thesis)

- is specified by one or more *owners* or *stakeholders*,
- is **distributed** - in terms of:
  - *ownership aspects*:
    - when owners and stakeholders specify a workflow (also known as the main-workflow), the objectives of the workflow, interfaces between different resources, and possible penalties for misbehavior are agreed by all involved owners and stakeholders. A main-workflow can have different sub-workflows from different owners and stakeholders. The sub-workflows and its internal aspects need not be shared with other owners or stakeholders, by doing this, the confidentiality aspects of individual owners or stakeholders are preserved. In case of misbehavior, a judge (a trusted third party) or a decentralized consensus mechanism is used to blame the party that misbehaved.
    - in order for the workflow to be executable, each owner must ensure that their devices (or resources) are available and they respond as specified or agreed at all times. Note: workflow specific conditions may apply when the promised guarantees are not satisfied.
  - *execution aspects*:

    * the main workflow or its sub-workflows can be executed by different workflow participants simultaneously. Therefore, each participant will have his/her own view of the main workflow.

- has one start place and one end place. The initial state of the workflow may have static tokens already available - representing the services available - and the workflow participants provide dynamic tokens during the runtime to the workflow which changes the workflow behavior. As the workflows are modeled in Petri Nets with the above mentioned properties, they can be analyzed for certain properties, such as soundness and liveness - refer to "Workflow-Nets" which was introduced and studied by Van der Aalst (see [147]) extensively.
- is considered completed when a token reaches the end state of the workflow.

**Definition 9.2.2** (Task)**.** *A task in a workflow is atomic and cannot be sub-divided. When a task is completed, then it changes the workflow state. Note: at the implementation level, a task is executed as a series of "micro steps", that means depending on the abstraction level, it can be divided into a sequence of multiple small steps.*

A task (in the context of this thesis),

- is interpreted as the actions committed by individual participants that help to synchronize different participant actions to a particular step of the workflow
- requires one or more entities (e.g., computer process or human workflow participant) to perform the task,
- takes some input data (e.g., from different participating entities participating) in the form of tokens and processes them according to the predefined conditions, and
- finally, it produces one or more outputs tokens (e.g., task completed, additional tasks or even new sub-workflows).

**Definition 9.2.3** (Workflow Integrity)**.** *The property that guarantees that the workflow is executed as specified by the owners and stakeholders is known as* workflow integrity *(i.e., no unauthorized entity can change the workflow by adding or deleting or modifying the tasks or the order in which they are processed). As an exception, to handle error conditions, the responsible authority (an authorized entity) is able to provide an error-token or a new workflow to handle foreseen and unforeseen conditions respectively.*

One of the thesis goals is to ensure that the workflow is able to handle error conditions. There are two different types of errors:

- *foreseen errors*:
  - errors that are expected to happen on certain conditions, e.g., failure of a device
  - errors that can be handled by integrating the alternative logic within the workflow and without manual intervention by the responsible authority

  To handle this, authorized entities (e.g., usually the owner of the workflow) provide error-recovery tokens (e.g., a token signed by the owner's private key) to the workflow participant; this error-recovery token triggers the alternative logic embedded within the workflow to handle error conditions.
- *unforeseen errors*:
  - errors that are not expected, e.g., an error that has not occurred or known previously,
  - errors that require manual intervention from the responsible authority

  To handle this, the responsible authority may provide a new workflow to the participant and terminate the old workflow manually. The authority must perform required accountability measures, such as logging the reason for the failure (if known), collect and document available context information, and inform relevant entities to take necessary action to prevent this in the future. Once a root cause of the error and an appropriate solution is found, the solution or preferred recovery mechanism could be embedded within the workflow logic itself.

**Definition 9.2.4** (Error-Recovery Tokens)**.** *Error-Recovery Tokens are tokens produced and signed by entities (usually, the owners or stakeholders) that enable the workflow participants to recover from error situations faced while executing the workflow.*

An error-recovery token:

- includes necessary information encoded in a recognizable format by the workflow execution system, for instance, the workflow application running in the handheld.
  - e.g., a token

**Definition 9.2.5** (Proof-of-Possession Tokens)**.** *A token may be bound to a cryptographic key, which is then used to bind the token to a request authorized by the token. Such tokens are called proof-of-possession tokens (or PoP tokens) (see [3]).*

**Definition 9.2.6** (Workflow-Aware or Workflow-Drive Access Control)**.** *An access control method where subject requests to perform operations on objects are granted or denied based on the workflow information, i.e., (tokens) provided by subjects, workflow state, objects attributes and a set of conditions that are specified in the workflow task which is executed by the subject.*

This thesis focuses on enforcing the WFAC in a distributed environment, therefore, known as the Distributed WFAC.

## 9.3 Contributions Overview

This section presents an overview of the contributions of this thesis. The contributions focus on addressing the research questions and goals presented in Sec. 1.1.1 and Sec. 1.1.2, and challenges presented in Sec. 2.

The State-of-the-Art analysis addressing the scenario focused on the thesis, IoT technologies, modeling workflows and enforcing them are presented in Part. II - see Chap. 7 and Chap. 8. Existing mechanisms (see Chap. 8) are not sufficient to satisfy the goals and requirements of this thesis. Thus, centralized workflow management systems and mechanisms that require constant synchronization with a centralized workflow server are not suitable for a distributed IoT scenario where devices may not have connectivity to that central server at all times.

The requirements of the envisioned workflow specification and enforcement framework are the following:

- A method that binds an access control to the tasks defined in an authorized workflow and enforces the integrity of the workflow in a distributed IoT environment.
- The proposed method should support dynamic workflows that can handle error conditions, i.e., allowing to integrate on the fly sub-workflows without changing the objective of the main workflow.
- The developed method and its components should be modular, interoperable, and user-friendly. For instance, it should support existing authorization standards such as OAuth and should support the integration of practitioner-friendly tools for modeling workflow.
- The proposed framework must support accountability features. For instance, the workflow participants should be held accountable for their actions committed, i.e., when necessary, the framework should support retrieval of information related to actions and prove which entity committed that action while executing the workflow.

As a result, the contributions of the thesis are categorized into three different aspects: (a) workflow specification and enforcement; (b) IoT protocols and services to support the workflow execution; and (c) to support accountability in a distributed environment.

- Chap. 10: Introduces the Petri Nets based workflow specification and execution framework (also known as the WFAC)
- Chap. 11: Presents the IoT Protocols and Services to support the introduced WFAC framework.
- Chap. 12: Presents the accountability efforts using the distributed ledger, e.g., blockchain for storing workflow execution related information. As part of the WFAC framework, the Petri Nets based secure smart contract generation framework is introduced which generates smart contract templates from workflows specified in Petri Nets.

Finally, in Chap. 13 the WFAC framework and its components with architecture description and their interaction with each other are described with a demo use case and a prototypical implementation.

# Chapter 10

# Workflow Specification and Execution

In this chapter, Petri Nets based workflow specification and execution related contributions are described in detail. In particular, it describes how Petri Nets are extended to support the requirements and goals identified in the thesis.

## 10.1 Petri Nets for Workflow Specification and Execution

The advantages of using Petri Nets (PN) over other workflow specification and verification approaches are presented in the Sec. 4.4.1. The advantages of PN can be categorized into technical and non-technical advantages.

The technical advantages are the following:

- PN provide the formal semantics for designing workflows such that PN workflows are amenable to verification of certain properties, such as being deadlock free. Another possibility is to analyze the soundness properties of a workflow.
- PN provide the ability to model concurrent and distributed processes.
- With PN one can limit the expressiveness of workflows, i.e., Turing completeness can be avoided and therefore, the workflows and their execution path are computationally analyzable.
- PN can be extended, for example, colored Petri Nets, Open Petri Nets, and Hierarchical Petri Nets.
- PN workflows are technology or platform-independent. Therefore, it can be used to implement and integrate platform or technology dependent multi-tenant processes

The non-technical advantages are the following:

- Workflows modeled via PN are easy to understand because of its intuitive graphical nature and easy to model tools.
- The interactive token-game semantics of PN allows even a novice business user to design, validate, and analyze the workflow.
- As described earlier, existing practitioner-friendly tools that collect requirements and create activity diagrams can be integrated with custom tools to generate Petri Net workflows.

This thesis has chosen Petri Nets for workflow specification and enforcement framework because of the above mentioned technical and non-technical advantages. In the literature, Petri Nets were used only for modeling and analyzing workflows, but this thesis is using the Petri Net model also for enforcing workflow integrity and integrating access control mechanisms suitable for IoT. The expressiveness of Petri Nets and the state-transition model of Petri Nets support the basic primitives needed to model a workflow process precisely. Extensions of Petri Nets such as high-level Petri Nets, Open Petri Nets, and Workflow Nets enable specifying and modeling complex workflows that involve solving different issues such as concurrent task execution and separation-of-duties.

Overall, Petri Nets satisfies the important requirements to specify and model workflows, but not all the requirements and goals required to fulfill the use cases that this thesis envisioned to solve. The use cases are presented in Sec. 15.

The following extensions are introduced to the Petri Nets:

- (a) to support interaction between distributed processes in IoT that needs interaction between devices that execute workflow processes, devices, and other services.
- (b) to support and evaluate complex workflow conditional requirements, for instance, timeouts involved in a certain transaction.
- (c) to support the capability of handling error conditions and dynamic workflows.

### 10.1.1 Extension to Petri Nets Places and Tokens

This thesis has introduced the following additional concepts as extensions to classical Petri Nets places and tokens:

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

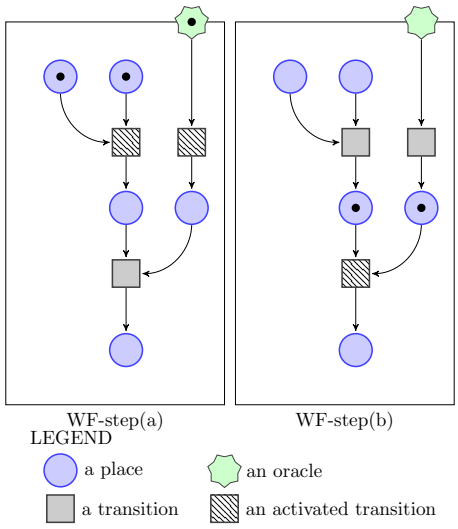



Figure 10.1: WF-step(a) shows the initial state of a Petri Net Workflow specification with an Oracle. WF-step(b) shows the state of the workflow after the first two activated transitions have fired.

information between them. In particular, OAuth tokens are used to enforce access control in a step wise manner as specified in the workflow.

- An *Oracle* is a type of place, represented in star shape that can receive tokens from an external source. In classical Petri Nets, places are represented as circles and they 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. The introduced concept of an Oracle is similar to the Oracles introduced in blockchain, i.e., it is used to receive external information in a smart contract deployed in a blockchain system like Ethereum. The difference is the following: An Oracle place mentioned in this thesis need not be a contract that is accessed by other contracts to pull information as described in [148, 149]. 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.

The main difference between an oracle and an Open Petri Nets (Open-PN) place is an oracle can receive information from external sources whereas, 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.

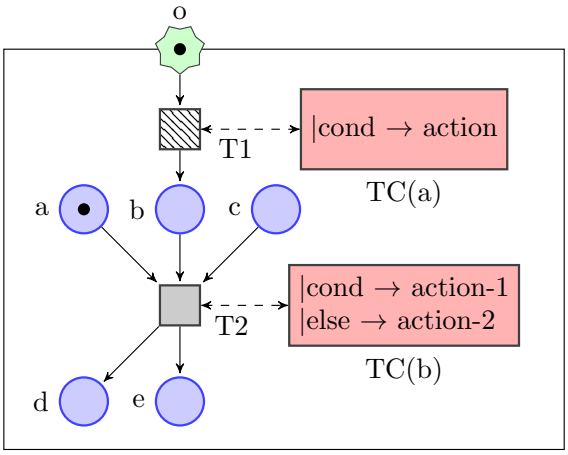



Figure 10.2: Petri Net with Transition Contracts t1 and t2.

### 10.1.2 Extension to Petri Nets Transitions

As described earlier, with extensions to Petri Net places and tokens, workflows can be specified with expressiveness, but we need a mechanism to evaluate complex workflow conditions. To realize this, the transitions should be able to verify conditions and evaluate information encoded in the tokens.

For this purpose, small *contracts* are embedded into the required transition of the Petri Net. For our requirements both Bitcoin and Ethereum languages are not suitable. Bitcoin's stack language is not flexible and Ethereum's solidity language could be vulnerable (see [112]), as verifying such contracts is not possible. Therefore, a language that is flexible specifying conditions and at the same time verifiable is required. We use a simple guarded command (a conditionally executed statement) language (similar to [150]) to write such contracts. Therefore, the conditions written on a single transition using a simple smart contract language is called a *transition contract*. On a high level, a complete Petri Net workflow with small transition contracts can be seen as a typical 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*.

Figure 10.2 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 the 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*. 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. The combination of Petri Nets and transition contracts is used to specify, enforce sequences of atomic transitions (transactions), such that process integrity is guaranteed.

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 and 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 the proposed WFAC method is used, compromising one device may not compromise other devices.

> **Example:**
>
> 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. Note: if the attacker is able to compromise one of the devices that are involved in the workflow to update the firmware, then it becomes a serious issue - this particular attack is discussed in detail in Sec. 14.3.

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 minutes, and others might need some hours. If a transition is waiting for a token to arrive in one of its input places, it probably does not want to wait indefinitely.

Timeouts 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

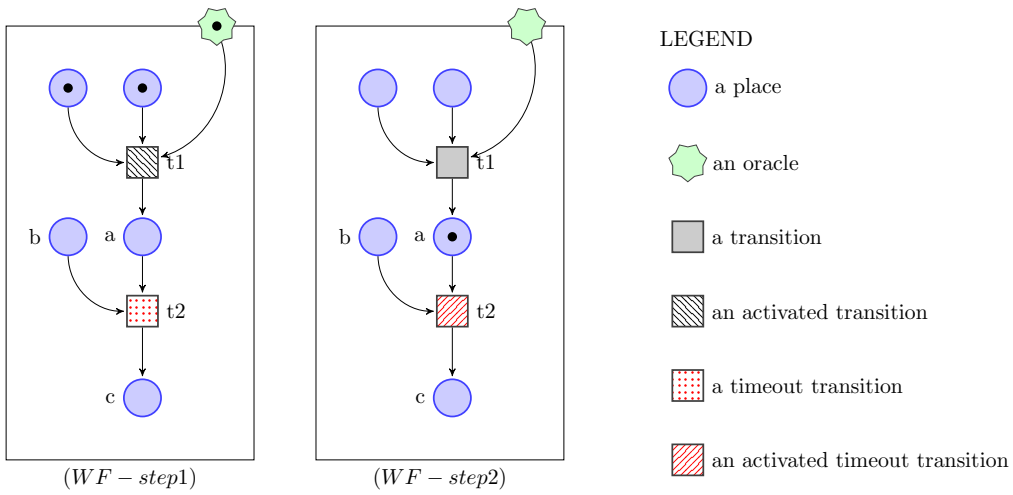



Figure 10.3: Timeout Transitions in Petri Net Workflows

be completed within a certain time. That transition may wait forever to get a token in one of its input places. *Timeout transitions* are introduced to solve this problem, i.e., after a predefined time expires, the timeout transition executes a 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 contracts are executed to produce tokens. Therefore, every timeout transition has two instructions: first, a timeout instruction (timeout contract) is enforced when a 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. 10.3 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 define 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) already has 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.

### 10.1.3 Analyzability of Petri Nets Extensions

Petri Net extensions presented in this thesis (which we now refer to as $\xi$-PN) has transitions of the form: $\tau : g \rightarrow a$, where $\tau$ is the transition, $g$ is a guard, and $a$ is an action, uses an arbitrary program (e.g., a Python program) to implement the guard $g$. For instance, it is common to use such a program to validate the token. We assume that all the python programs used in different transitions halt because the user will notice it and eventually update the program.

In basic Petri nets, the transitions exhibit the so called 'AND' property, i.e., if a transition has sufficient input tokens then the transition fires and produces a token in all output places. To analyze $\xi$-PN introduced in this thesis, the transitions are translated into conventional Petri Nets with the 'OR' property ($\zeta$-PN), i.e., when a transition has sufficient input tokens then the transition produces tokens in one or several places.

Let $\xi$-PN be the Petri Nets with extensions introduced in this thesis and $\zeta$-PN be translated basic Petri Nets with 'OR' transitions. In $\zeta$-PN transitions fire in a non-deterministic way, therefore, every *trace* in the $\xi$-PN is also a trace in the $\zeta$-PN, but not vice-versa. Security is a safety property. To prove that a system is secure, we must ensure that the system satisfies a safety property in terms of Schneider (see [15]). Therefore, when a safety property is satisfied in the translated $\zeta$-PN, then it is also satisfied in $\xi$-PN also, i.e., when the translated Petri Net $\zeta$-PN is proven secure, then the extended Petri Net $\xi$-PN is also secure but not vice-versa. Therefore, the extensions introduced to the Petri Nets are analyzable.

### 10.1.4 Dynamic Workflows

The proposed Petri Nets 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. *Dynamic Workflows* are introduced to handle such special situations 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 the "authorized" entity can create dynamic workflows and any misuse must be penalized. Therefore, the requirement to have the accountability of actions performed by the participants while executing the workflow is important.

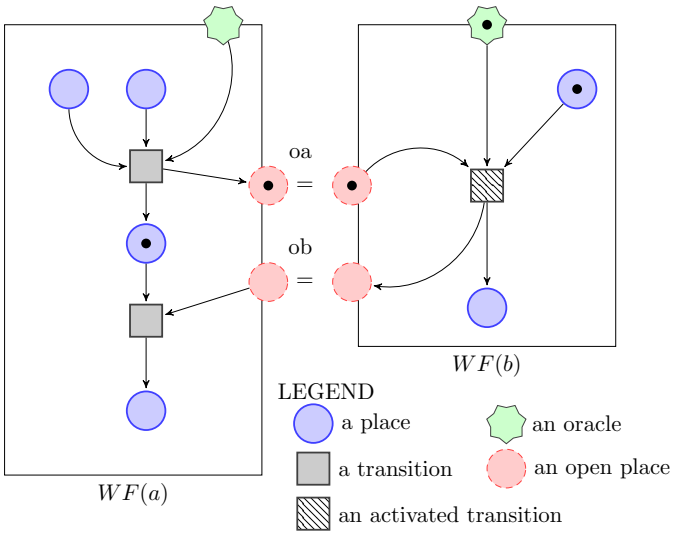



Figure 10.4: Two different workflows WF (a) and (b) exchange information using Open Petri Net places (oa and ob)

Thanks to Open-PN (see [88]), which can create an entry and exit points between Petri Nets, i.e., Open Petri Net places can be used to exchange information between Petri Net workflows. Exchanging information in the form of tokens simplifies the integrating of two or more Petri Net workflows. Open Petri Nets enables the interaction between other Petri Net workflows or processes. Open Petri Nets are also used to design, build, and evaluate Hierarchical Petri Nets.

**Example:**

Fig. 10.4 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 an entry place for WF (b).

## 10.2 Usability

To ensure that the introduced security procedure is adopted and followed by the end-user and practitioners, the security experts must design the security procedures or mechanisms in a user-friendly way. People try to circumvent the newly introduced security procedures when additional security procedures interrupt their normal work (see [151]).

This thesis envisions that the proposed access control method is usable with the practitioners and can be integrated with day-to-day work performed by the people.

Wherever one has a workflow that must be followed in their day-to-day work can be identified, then apply the proposed method to reduce privilege abuse, i.e., grant access to protected resources via the WFAC. Also, the thesis ensures that practitioners can easily migrate to the proposed security mechanism with little additional effort by focuses on interoperability with existing tools and protocols.

To summarize, the main focus of this thesis on usability aspects are:

- Practitioners should be able to model workflows only with the business knowledge or with little efforts to learn the modeling tools
- The deployed WFAC mechanism should be easy to use for the end-users. This thesis recommends the practitioners to get feedback from the end-users and try to fix the issues pointed out in the feedback.
- The proposed WFAC mechanism should be interoperable with existing infrastructure, i.e., compatible with existing security protocols and software architecture. This is another important factor for a proposed system to get adopted by the practitioners and system developers.

In the next section, this thesis presents its approach which enhances the practitioner's usability of the proposed WFAC approach. The modeling specification language Petri Nets, even though it is supported by graphical modeling tools and intuitive, is directly not suitable to be modeled by the business people, for example, managers.

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

This thesis investigated how a practitioner (a software developer or engineer, even a business manager) could use the WFAC method with existing and familiar tools. One of the challenges is to model the interconnectivity among systems, i.e., distributed and interactive multi-organizational processes. Therefore, systems can no longer be treated as stand-alone silos, but behave as part of a larger ecosystem including human interaction. Such complex systems are known as the System of Systems (SoS) [152]. It could be complex to design during the first phase such as SoS models that include different software and hardware components only using Petri Net tools. 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 notations. SysML supports the practice of Model-Based Systems Engineering (MBSE) and is an extension of UML version 2. SysML's usability has been studied by conducting usability experiments in [39, 40, 126] and the empirical results showed that SysML is usable and practitioner friendly for general purpose modeling and domain specific modeling.

The generally accepted method is to refine the specification in a stepwise manner using software engineering tools such as the OMG SysML activity diagram presented in [153]. The Object Management Group's OMG SysML [153] 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 can represent different aspects of systems, components, and other entities [152] 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, the SysML activity diagram describes how the actions transform the inputs into 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). SysML activity diagrams lack mathematical semantics to check for inconsistencies, but the SysML activities are based on token-flow semantics related to Petri-Nets [154]. Therefore, these activity diagrams can be converted into Petri Nets (for example, colored) and then can be verified using model checking tools (see [155, 156, 157, 158]).

Therefore, a practitioner-friendly open source modeling tool that supports SysML known as "Modelio" [127] is proposed 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. 15.4. SysML is used to model a high-level activity diagram of complex processes and systems or SoS.

# Chapter 11

# Internet-of-Things (IoT) Protocols and Services

As part of the chapter, the IoT related contributions are classified into two sections, they are:

- (a) Privacy Enhanced Tokens for IoT devices: A profile for ACE-OAuth 2.0 for constrained devices was developed. This profile helps to protect the identity of workflow participants accessing resources from resource servers that are not able to protect the communication channel.
- (b) Receipt Tokens for Workflow Execution: A mechanism is developed to guarantee whether the workflow participant has completed a task or not. This mechanism produces tokens, also known as "*Receipt Tokens*" created usually by the resource servers and given to the workflow participant.

This section discusses these two contributions in detail.

## 11.1 Privacy Enhanced Tokens (PAT) for constrained IoT Devices

IoT devices also include constrained devices and some of those devices might not be able to protect the communication channel, i.e., constrained IoT devices (RS) are not able to use communication channel protection mechanisms such as DTLS.

In this scenario, the two following major problems must be addressed:

- (a) While accessing a resource from RS the responses from RS must be available only for the intended participant, i.e., Workflow Participant (WP).
- (b) identity of the workflow participant (which may include private information) must be protected to anonymous entities eavesdropping the communication channel.

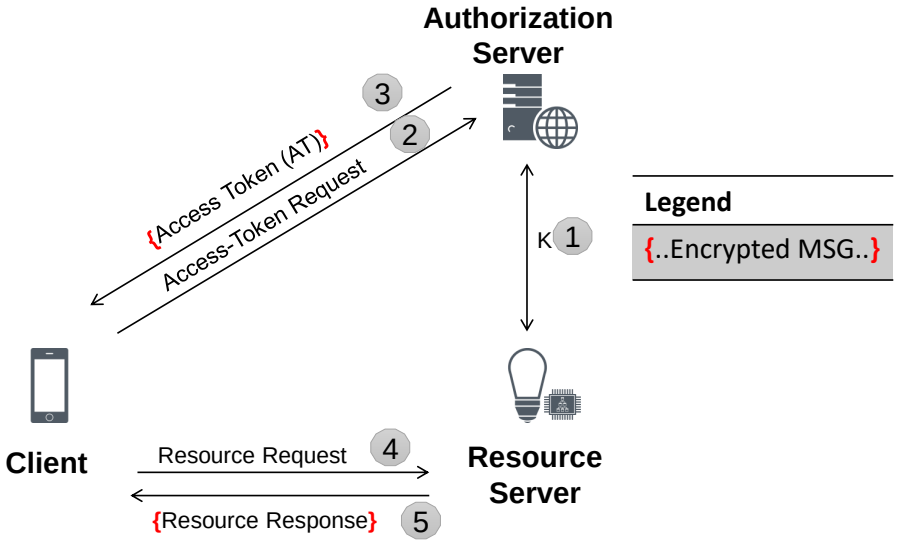



Figure 11.1: An example ACE-OAuth PAT scenario and actors involved. The numbers explain the sequence of an authorization process and resource request between three actors. Notations: K is a shared secret (can be both symmetric or asymmetric) and {encrypted message}.

To address these problems, this thesis has developed a profile for ACE-OAuth to ensure the workflow participant's identity is not leaked even in situations where RSs are really constrained and cannot protect the communication channel. The profile was also developed within the context of the European Union Project REliable, Resilient and secUre IoT for sMart city applications (RERUM). Below, the ACE-OAuth message exchanges are presented where a WP first acquires an access token and then with that access a resource in the resource server.

The Privacy Enhanced Tokens for ACE OAuth (PAT) protocol is designed to work with ACE framework [47] and ACE actors who were already introduced in Sec. 5.2. Please note that the PAT protocol specification was designed when the ACE OAuth draft was in its 15$^{th}$ revision and was also published as an IETF draft (see [30]).

In PAT profile for ACE, the following assumptions hold:

- A RS has one or more Resources (Rs) and it is registered with an AS
- The AS provides an AT for the CL to access R of RS. The corresponding RO of the RS may assign allowed-permissions for WP in AS.
- The RS is offline after commissioning, i.e., RS cannot make any introspective queries to the AS to verify the authorization information provided by CL.

- A CL is either registered with an AS or it knows how to reach the RS for accessing the required resources.
  - To access a resource on a RS, a CL should request an AT from AS, either directly or using its Client Authorization Server (CAS). For the sake of simplicity, this memo does not include the actor CAS.

Based on the above described scenario, a simple ACE-OAuth PAT message flow as shown in Fig. 11.1. A CL may perform a resource-request to RS without a valid AT, then RS will reject, and it may provide AS information to CL in the response. Such that, CL may go to the AS to get a valid AT. The RO may define access control policies on the AS describing who can access the resources on a RS.

The messages exchanged are the following:

- (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 CL. This secret can either be symmetric or asymmetric in ACE-OAuth context. In PAT, a symmetric secret is considered.
- (2) The CL performs an Access-Request to AS to ask for AT to access R on RS. The AS checks if CL can access the R on RS or not, based on permissions assigned by the RO.
- (3) If CL has sufficient permissions, then AS generates an AT plus a PoP key bounded to the AT and the secret ($k$). AS sends both the AT and the PoP key to CL via a secure encrypted channel. Note: CL and AS are not constrained therefore, having a secure encrypted channel between them is possible and is important for transferring this secret.
- (4) After receiving AT and PoP key, CL performs a resource-request to RS by ACE-OAuth token construction method defined in PAT profiles - which is described in detail 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.

At the end of this phase, both CL and RS has established a common derived secret. Later, CL can generate unlinkable Derived-Token (DT) from the initial AT as described in section construction of derived tokens [30]. In particular, PAT is designed to be used in contexts where unlinkability (privacy) and efficiency are the main goals: The Token (Tk) are constructed in such a way that they do not leak information about the workflow participant requesting the resource even via multiple requests. For example, if an eavesdropper observes the messages from different Workflow Participants to and from the Resource Servers, the protocol does not give him information about which messages correspond to the same Workflow Participant. Of course, other information like the IP-addresses or the contents

themselves of the requests/responses from lower-layer protocols may leak some information, and this can be treated separately via other methods.

The main features of PAT protocol are described below:

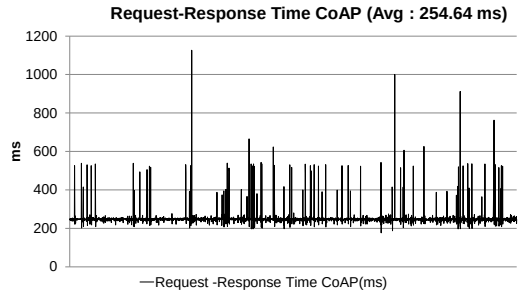
- The PAT method allows a RO, or an AS on its behalf, to authorize one or several clients (C) to access resources (R) on a constrained RS. The CL can also be constrained devices. The AT response from AS to CL MUST be performed via secure channels.
- The RO is able to decide (if he wishes: in a fine-grained way) which client under which circumstances may access the resources exposed by the RS. This can be used to provide consent (in terms of privacy) from RO.
- The ATs are crafted in such a way that the client can derive Tk also known as DT. The message exchange between CL and RS for the presentation of the tokens MAY be performed via insecure channels. But the payload content – if CL is performing a POST/PUT/DELETE request – from CL to RS or the response payload from RS to CL MUST be encrypted.
- The RS can derive the PoP key from the AT of Resource Request message from CL, but the response from RS to CL are encrypted.
- The Tks do not provide any information about any associated identities such as identifiers of the clients, the tokens themselves, and the resource-servers.
- The Tks are supported by a "proof-of-possession" (PoP) key derived from the initial AT. The PoP key allows an authorized entity (a client) to prove to the verifier (here, the RS), that CL is indeed the intended authorized owner of the token and not simply the bearer of the token.

To be coherent with the ACE Authorization framework [48], this draft also specifies an ACE profile to use PAT and for efficient encoding it uses CWT and COSE. The PAT profile is signaled when CL requests token from the AS or via RS in response to unauthorized request response. The PAT profile will cover all the requirements described in [48].

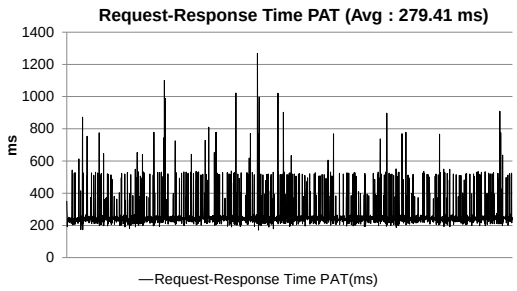
The detailed description of PAT protocol is presented in this IETF draft [30] and is described in the appendix Sec. B of this thesis.

To calculate the overhead and performance, PAT was implemented in a constrained device known as RE-Mote [1] using the IoT operating system known as Contiki [2]. The implementation was done via a joint work with ATOS as part of the EU project RERUM [159]. As a result, PAT profile added ~ 10 % overhead on an average in comparison to plain CoAP messages. The Fig. 11.2a shows the request-response

[1] `https://zolertia.io/product/re-mote/`
[2] `https://github.com/contiki-ng/contiki-ng`




(a) Request-Response Time of COAP




(b) Request-Response Time of COAP + PAT profile

Figure 11.2: Performance Evaluation of PAT profile for ACE-OAuth

time when just CoAP is used and Fig. 11.2b shows request-response time when PAT profile is used.

## 11.2 Receipt Tokens

This section presents a concept that enables workflow validation entities such as resource servers and applications to validate whether a workflow participant has performed the activity or not. Also, this mechanism enables the workflow participant to prove to another entity that it has completed a task in case of a dispute. This thesis presents this concept in the form of tokens that contain workflow specific information signed usually by the resource servers, these tokens are introduced in this thesis as "*receipt tokens*".

**Definition 11.2.1** (Receipt Tokens)**.** *Receipt tokens are tokens produced and signed by entities (usually, the resource servers or IoT devices) attesting a task completed by the workflow executor (e.g., a user) as part of a workflow.*

In Fig. 11.3, an example Workflow (WF-a) and its use of receipt tokens are presented. In the shown example, a CL after getting an AT from an AS tries to access a set of resource servers ($RS_1$, $RS_2$, $RS_3$ and $RS_4$) and their resources in a sequence.

Let us consider that RO of all four Resource Servers has configured the RS in the following way:

- $RS_1$ requires a valid AT from AS for accessing its resources to complete WF-a.
- $RS_2$ requires a valid AT and a receipt-token from $RS_1$ for WF-a.
- $RS_3$ requires a valid AT and a receipt-token from $RS_2$ for WF-a.
- $RS_4$ requires a valid AT and a set of all receipt-token from ($RS_1$, $RS_2$, and $RS_3$) to access its special protected resource for WF-a.

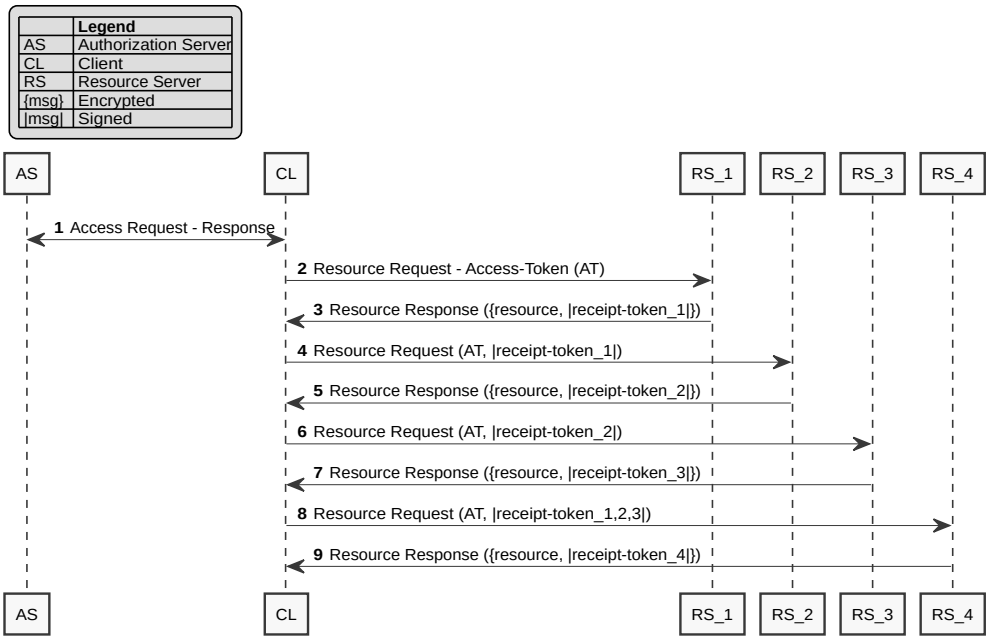



Figure 11.3: Clients (CL) using the receipt tokens to access resources from Resource Servers (RS)

As shown in Fig. 11.3, message 3 shows the first receipt-token$_1$ signed by the RS$_1$ and it is sent together with the resource in an encrypted response to the workflow participant initiating the request. If PAT profile is used, then the corresponding PoP key is used for encrypting the response. So, only the CL with the right key is able to decrypt the response. As you can note, to access RS$_4$'s resource the CL should present all receipt-tokens from all three previous RS. This brings additional security to the protected resource but at the same time it also brings additional overhead for the resource server because it must process all those receipt-token and therefore, should have appropriate key material of other resource servers. How this key material is transferred is not within the scope of this thesis. However, the decision to add such additional protection to the resources or not is left with the RO.

An example receipt tokens encoded in CBOR Web Token (CWT) are shown in the Listing 11.1. The advantages of having the receipt tokens are discussed below and are shown in the evaluation of WFAC and its attacker model Sec. 14.3.1.

```
CWT Header:
{
     "typ": "CWT", # type of token
     "alg": "..." # algorithm used
}
CWT Payload:
{
     issuer (iss)    : RS_1, # Resource Server
     subject (sub)   : Hash(AT), # AT is the identifier in case of PAT
     audience (aud)  : WF-a (RS_2, RS_4), # necessary workflow info, tasks
     CWT ID (cti)    : "receipt-token_RS_1", # arbitrary identifier
     expiration (exp): ..., # omitted for brevity
     issued-at (iat) : ... # omitted for brevity
}
```

Listing 11.1: An example Receipt Token format their contents encoded in CBOR Web Token (CWT) format

# Chapter 12

# Distributed Accountability and Access Control

The WFAC framework enforces the users by restricting them to perform tasks as specified in a particular order in the workflow. The WFAC framework uses a distributed blockchain network to achieve accountability and transparency. 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 [160]). A classic example of permissionless blockchain is bitcoin [105], 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 thesis.

To achieve distributed accountability, the workflow participants may publish workflow information in the blockchain. 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. A permissioned blockchain can be used to restrict the participants and the respective access control restrictions. For example, IBM's Hyperledger can be deployed as a permissioned blockchain where entities require permissions to access and publish information in the blockchain. When the workflow participant publishes the status of task (which needs to be published in the blockchain), the stakeholders will verify and approve the transactions in the blockchain. 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 and on the IoT devices and services. 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. A trusted application installed on the entity's handheld is used for enforcing the validity of the tokens generated and received. If the application is not trusted, then the IoT devices may also delegate these validation tasks to external services, for example, to check the blockchain for updates, or, to pull information tokens from an oracle, etc.

## 12.1 Secure Smart Contract Generation Framework

In Sec. 8, we presented various approaches to generate smart contracts. In this section, we present a method to create secure and safe smart contracts for platform-independent blockchain systems. Our approach presents a secure smart contract modeling tool using Petri Nets (PNs). Petri Nets allow us to visually model a process or workflow that represents one or more business logic. Once the Petri Nets are modeled and verified, our tool allows us to generate a smart contract template. The exported smart contract template allows the smart contract developers to extend the functionality before deploying it on the blockchain.

Our approach is not restricted to a particular blockchain technology. Our modeling tool can be extended to support any blockchain platform or smart contract programming language. For instance, now our tool is capable of producing a Solidity Smart Contract (SC) template that can be deployed in an Ethereum blockchain. The reason for selecting Solidity for the proof-of-concept (PoC) is solely for demonstration purposes and Solidity was not compared with other languages supported by Hyperledger Fabric, as described earlier, as our architecture is modular it can be extended to support other smart contract programming languages. The goal of our work is to help business owners, developers, and resource owners to create secure and safe smart contracts in a practitioner-friendly way. The modular software architecture of our tool can be extended to provide SC template for different blockchain systems.

In this section, we introduce our proposed *Petri Nets based Secure Smart Contract Generation Framework* that involves a multi-step process to generate *safe, secure, and human-understandable* smart contracts. Our approach focuses on security by design approach and it is based on Petri Nets.

The requirements of a SC generation should be as follows:

- A SC should be easy to understand and write, i.e., human-understandable, practitioner-friendly methods, and tools should be available to model the business logic or workflows.

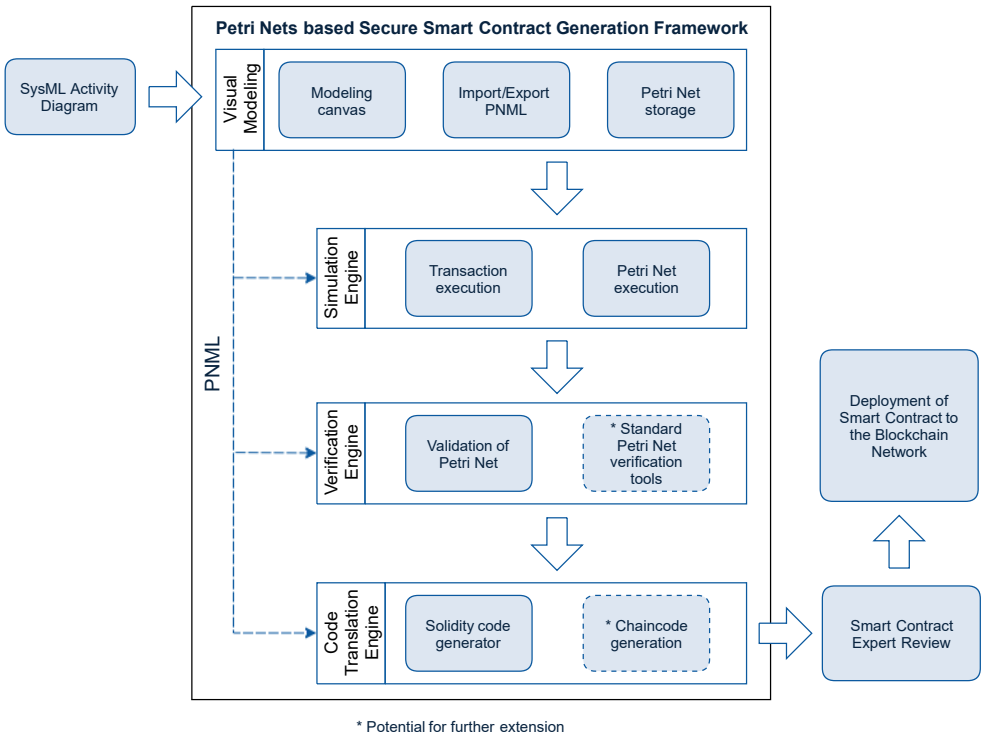



Figure 12.1: Modular architecture of our Petri Nets based Secure Smart Contract Generation Framework.

- A SC should be amenable to verification of process integrity, i.e., the smart contract should only allow the authorized actors and restrict them to perform actions with least-privilege principle. For more information on process integrity see [28].

- If necessary, the smart contract modeling language should support human interaction, for example, to approve or reject conditions specified in a SC.

We designed our framework based on the above mentioned SC requirements. Fig. 12.1 shows a modular architecture of our proposed framework which consists of following components: 1) a *Petri Net Visual Modeling* a graphical user interface (GUI) where user can model, import, export and store Petri Net workflows; 2) a *Simulation engine* that accepts Petri Net Markup Language (PNML) representation of the workflow and can execute each transition separately or a complete Petri Net; 3) a *verification engine* that preforms Petri Net validation and can be extended with standard Petri Net verification tools; 4) a *code translation engine* that generates the smart contract from the Petri Net model. In addition to the main components, our framework includes a SysML activity diagram modeling tool to support practitioner-friendly methods, and a smart contract expert review process to introduce a strict security auditing process within the framework before deploying the generated smart contract into the blockchain network.

The multi-step process of creating secure smart contracts starts with the stakeholders who identify the use case requirements and create the necessary business logic. To represent the business logic in a human-understandable way, we propose to use the SysML's activity diagram because it could be difficult to model complex business logic directly in Petri Nets. Open source SysML modeling tool "Modelio" [127] can be used to create activity diagrams. Later, the SysML activity diagram can be translated into Petri Nets directly using automated tools (see cite: [155]) or by a workflow expert via our Petri Net Visual Modeling GUI as shown in Fig. 12.2.

A Petri Nets model can be created from scratch or can be imported from other existing Petri Net models described in the PNML which is the standard XML format to exchange Petri Net models. After importing, the PN model can be modified, extended and stored according to the needs of the workflow expert.

During the modeling process, the user can simulate the Petri Net by executing each transition separately or by fast-forwarding through the complete Petri Net. This process allows us to test workflow logic already during the modeling process and allows for fast iteration.

The next step is a verification process of Petri Nets that is performed by the *Petri Net Verification Engine*. The Verification engine takes care of two main functions (validation and verification) while the user models the Petri Net workflow. The validation of Petri Net properties can help to optimize the business logic and to find and avoid errors at the modeling state itself. Additionally, the external Petri Net verification tools could be integrated that support PNML representation format.

Finally, the verified Petri Net model is provided to the *Code Translation Engine* which produces the desired smart contract template. This translation engine is designed modular and therefore it is blockchain platform agnostic. Currently, our tool supports the generation of Solidity smart contract which can be deployed in the Ethereum blockchain network but the modular design of the framework support further extensions of code translation engine to support other blockchain technologies. The generated smart contract can be a standalone contract or a part of a big contract consisting of many SCs.

Below, we present a detailed description of every step. Once the template is generated, it might be necessary for a smart contract developer to complete the smart contract with necessary business-logic details that were not modeled in the PN model. In addition, a security audit on the smart contract may be performed before deploying the SC on the blockchain.

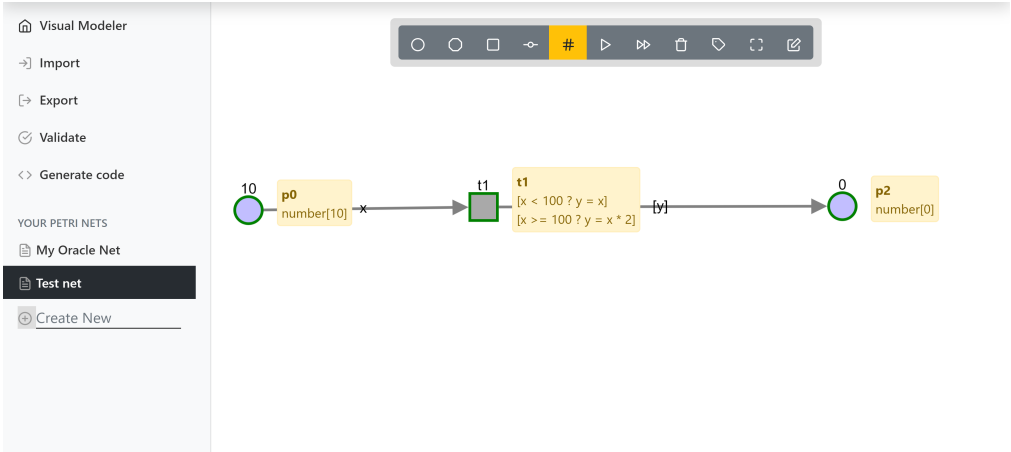



Figure 12.2: The user interface of Petri Nets based Secure Smart Contract Generation Framework featuring modeling canvas, toolbar, function menu and library of modeled Petri Nets.

### 12.1.1 Petri Net Workflow Visual Modeling

Petri Net Workflow Visual Modeling provides the user with a visual modeling graphic user interface (GUI), allowing to model Petri Net workflows through a canvas that guides the user with tips and information throughout the complete process – i.e., from modeling the Petri Net workflow until the generation of the smart contract. Our Petri Nets based Secure Smart Contract Generation Framework can be accessed via a web browser and does not require any server-side dependencies. Fig. 12.2 shows the user interface of the prototype which has four important elements: modeling canvas, toolbar, menu and workflow library. At the center of the application there is the interactive modeling canvas used to model the Petri Net workflows. The toolbar is located above the modeling canvas in grey color, and it contains tools to add new elements to the canvas (e.g., places, oracles, transitions), connect them with arcs and to edit, rename and delete them. As you can see in Fig. 12.2, on the left of the canvas, there is the menu with functions to import and export PNML representation of the modeled workflow, validate the workflow and generate smart contracts. Just below the menu, there is the library listing all stored Petri Net models that the user has created.

To ease the modeling process, GUI guides the user through the modeling process with visual cues and annotations, helping the user to understand and create the Petri Net workflow easily. There are three different element types that can be used in the modeling process. A most basic Petri Net workflow consists of at least two places a single transition, whereas the use of oracles is optional.

A place may represent a state of an actor in the workflow and it shall hold one or more tokens. These tokens are consumed by a transition and are moved to different places once the conditions of the transition are met and executed. Additionally,

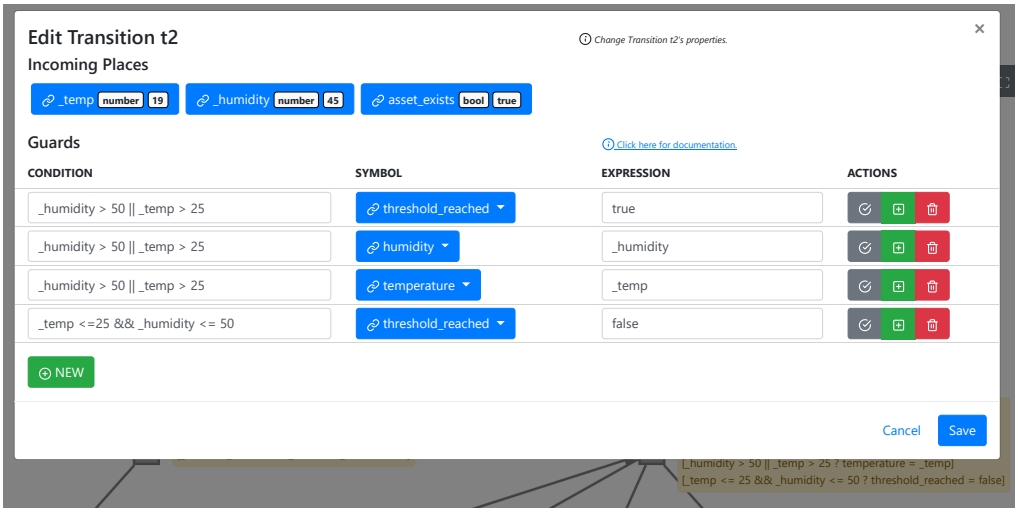



Figure 12.3: Transition edit view used to define guarded commands.

tokens can be of a different primitive data type (e.g., colored Petri Net tokens see [85]). In our framework, currently supported token can be of a type string, integer or a boolean, but the design is not limiting to only mentioned types.

The Oracles are a special type of places that represent external information that can be accessed within a Petri Net and is in the framework considered as an external parameter passed to the smart contract. Because the external parameters are tightly dependent on the use case the generated *execute* function can be modified by the smart contract developer in the review process in order to remove global variables representing oracles and pass them as variable values to the function according to the needs of the use case.

The transitions are extended with a guarded command language (see [150]) which is used to enforce the business logic conditions. Each transition contains one or more guarded commands. Each guarded command consists of a proposition and a statement. A proposition is a condition and if it is true, it will lead to the execution of the statement. The conditions are checked against the input tokes from the incoming places using algebraic and boolean expressions. Those conditional statements are written in the form of assignments, for example, $x = expr$ where $x$ must be named the same as one of the outgoing arcs of the transition. An expression ($expr$) can contain an algebraic expression that may include variables, constants, and algebraic operators. Figure 12.3 shows the *edit view* interface of a transition which is used to define the guarded commands. Every transition can contain multiple guarded commands. The visual modeling engine enforces that only the variables from the incoming places are allowed to be used in the expression ($expr$). The exit or output place of a transition is decided based on the execution of the statement and the transition may create new tokens, consume, or update existing input tokens and push it to the output place.

Additional functionality of the visual modeling engine is an option to save, import and export Petri Net workflows. This also increases the interoperability of the workflow development and allows the user to import basic Petri Net from other modeling systems. To facilitate import/export function, the PNML standard and its recognized exchange format are used. PNML exchange format is extended with additional XML tags to represent additional logic that was introduced in the form of guarded commands of the transitions, additional rules, and oracles. An example PNML is shown in the appendix listing A.1.

### 12.1.2 Petri Net Simulation Engine

Petri Net Simulation Engine is introduced to ease the modeling and to iterate on the workflow design during the modeling process. It consists of two main functionalities: (a) execution of a separate transition - a process which consumes the tokens from the input places of the transition based on the logic specified by the guarded commands and produces the tokens, and assigns them to the output places of the transition; (b) fast-forward of the complete Petri Net - a process which consecutively executes all transitions of the Petri Net based on the specified input parameters and guarded commands, and validates that the model does not result in a deadlock. The Simulation Engine interprets the PNML representation of the modeled Petri Net.

### 12.1.3 Petri Net Verification Engine

Petri Net Verification Engine supports the GUI by actively monitoring the Petri Nets visual modeling process and guides the user to correctly model the Petri Net by pointing out errors - an example of such error is shown in Fig. 12.5. Once the Petri Net model is completed without syntactic errors, then the verification engine supports validation (i.e., see validation of the PN model in Fig. 12.4). The verification of other Petri Net properties such as deadlocks, soundness, and liveness can be done with the help of external standard Petri Net tools such as CPN tools [85], WoPED [161], and YAWL [162]. The validation and verification of the PN model is a mandatory step and its role is to enforce Petri Net modeling restrictions by enforcing “validation” of modeled workflow prior to the generation of smart contract template.

All enforced modeling restrictions are displayed to the user during the modeling process through the visual cues providing the reasoning and instructions on how to resolve modeling mistakes. An example of the user interface informing the user of the successful validation is shown in Fig. 12.4, whereas Fig. 12.5 shows the response with errors that were identified during the validation process. Until the

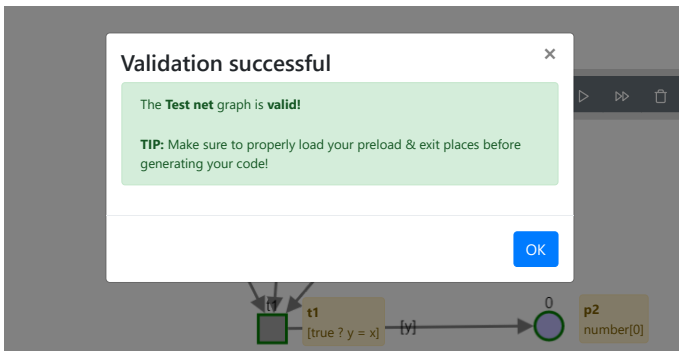



Figure 12.4: Feedback of successful validation step.

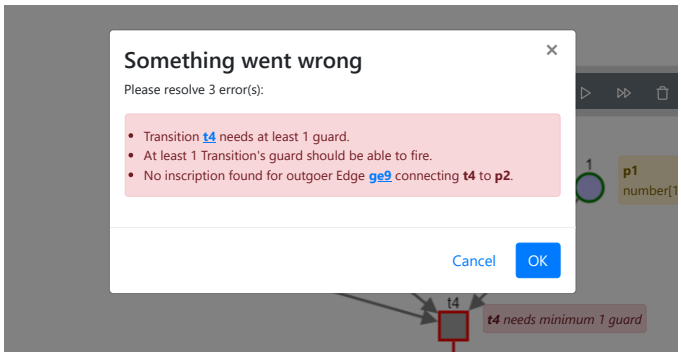



Figure 12.5: Feedback of validation step showing identified errors in the Petri Net model.

user does not resolve all errors, the proposed framework prevents the invocation of smart contract generation.

Petri Nets workflows are considered sound when it satisfies the following three requirements (see [31, 33, 147]): a) *an option to complete* you should be able to reach the end state or complete the workflow from any given state of the workflow; b) *proper completion* - when a workflow is completed no tokens (incomplete tasks) are left in the workflow; and c) *no dead transition* - every transition in the workflow can be executed by following the appropriate path. To facilitate the soundness property the following restrictions are enforced:

- There is one start place with outgoing arcs and one end place with incoming arcs.
- All transitions and places in the workflow should be along the path from start place to the end place.
- A transition must have at least one input and out place connected. Therefore all transitions and places need to be connected.
- Each transition must have at least one guarded command that evaluates to *true*. Note: having explicit else guard is, in general, good but not mandatory.

Arc annotations are used as a reference to the incoming and outgoing places via transitions.

Once the Petri Net model satisfies all conditions, the validation process is successful. Furthermore, to validate and verify other properties other Petri Net tools such as YAWL or CPN tools can be used. Finally, after the PN model is validated and verified, the smart contract generation step (icon generate code in the GUI) is enabled.

### 12.1.4 Petri Net to Smart Contract Translation Engine

Petri Net to Smart Contract Translation Engine helps to complete the final step of the multi-step process, i.e., to generate smart contract generation from the Petri Net workflows. This step is performed by the smart contract translation Engine that takes the PNML extended with guarded commands as an input and performs a translation of the Petri Net workflow into the smart contract code. In the current version of our tool, only translation to Solidity code is implemented but the engine is extendable, thus allowing us to translate to other smart contract types such as Hyperledger Fabric's chaincode.

The proposed framework takes an approach of mapping places, transitions, and guarded commands into smart contract code. A complete PN workflow is translated into a contract consisting of global variables representing places, functions that directly implement the execution logic of the transitions. The main execution function defines the procedure that is modeled by the Petri Net based on the conditioned executions of the state variables of places.

The places, oracles, and tokens are associated with the globally accessible information where they are translated into a pair of global variables representing a value and a state. The state variable represents whether the token was consumed from a place or not. If the token was consumed, the place is considered as disabled.

Each Place of PN when translated into a SC code is represented with the data structure:

- a struct will hold the value of tokens which can be of the specific type (i.e. *bool*, *string*, *uint*).
- a struct will hold the *enabled* or *consumed* state of the *bool* type.

A transition of a PN workflow is written using the guarded command language as proposed in [150], and each transition is translated to separate functions. Decoupling the main code from the functionality gives us several advantages such as re-usability, efficient analysis of functionality by testing a small piece of the code and easier modeling of the condition statements due to the use of the guarded commands. We map the guarded command language to the smart contract language's

logical and condition evaluation features such as Boolean and algebraic expressions and assignments.

The output of the function is mapped (or assigned) to the input place of next transition, and that can be achieved in two different ways:

- Approach (i): the main code invokes a function and assigns the return value (token) of the function to a variable representing the input place of the next transition.
- Approach (ii): builds on the premise that every transition of the PN can access only the places to which it is connected. The state of places is changed through assignments inside the function. In this approach, there is no need to return the results of the function to the main function.

The proposed framework implements a second approach. As the state of the place is changed from inside the function representing the transition, the translation engine ensures that the global variables representing tokens are not reused inside the functions to hold temporary states. The internal operations of a function first copy the global variable to a local variable and only if the expressions are valid, the global variable value is updated preserving the atomicity property. Therefore, we avoid any inconsistencies in the state of the same places.

In both approaches, we need a lock mechanism for writing/reading the value written to a place variable. To avoid race conditions that can erase a value from a place before it is being consumed, we propose to use mechanisms used in traditional databases, i.e., lock and release before writing and reading a value from such special places which are realized through the *enabled* and *consumed* state variable of the place.

Once the Petri Net is translated into a smart contract, a workflow expert reviews the generated SC code and publishes it in the blockchain.

### 12.1.5 Advantages of our Petri Nets based Secure Smart Contract Generation Framework

Our Petri Net workflows can be seen as high-level smart contracts that are amicable to formal verification, i.e., it is possible to check the workflow specific properties such as deadlock and workflow soundness properties. This supports avoiding such potential errors during the modeling phase of the workflow. Thus, translating validated and verified Petri Net models into a Solidity code will avoid those errors, therefore the generated smart contract template is considered secure and will protect the process integrity of the business logic. However, if the smart contract language has a language or platform specific problems or business logic errors then those problems will still exist in the generated smart contract. This holds true as

well for the scalability of blockchains. The process of modeling and generating smart contracts through the Petri Nets based Secure Smart Contract Generation Framework does not have an influence on the scalability characteristics of the blockchain platform.

In particular, the advantages of using our framework are:

- When domain specialists model the workflows using our PN based framework then this will minimize the chances of business logic errors during the modeling phase
- Workflow verification with the standard Petri Net tools can help to identify errors and prevent them from appearing in the smart contract.
- Simulation of PN workflow using the token-game at the modeling phase helps to understand the workflow intuitively before generating a smart contract.
- Platform independent modeling helps smart contract developers to design and deploy contracts in different blockchain platforms quickly by focusing on the modeling of business logic and not on the development process or on the programming languages.
- Smart contract generation without program development knowledge. This feature particularly helps stakeholders without programming or smart contract knowledge.

# Chapter 13

# WFAC Framework and Implementation

The chapter presents the WFAC framework, implementation and a demo use case to showcase the prototype implementation.

## 13.1 Workflow-Aware Access Control (WFAC) Framework

In this section, this thesis describes the architecture of the WFAC framework. This thesis has developed a WFAC framework with different modular components. First, the complete WFAC phases and architecture are described; second, detailed architecture description of each component involved in different phases is presented. The requirements presented in the use cases earlier are taken into consideration when designing the components of the framework. A prototype of the developed WFAC framework is implemented by this thesis. This chapter presents the architecture and insights to the implementation details whenever necessary.

Different phases involved in the WFAC procedure is shown in Fig. 13.1, they are:

1. *Negotiation Phase*: involves the process where the stakeholders, for example, business managers sit and negotiate the terms and conditions required for a specific process.

2. *Activity Diagram*: in this phase, the process activities are drawn in the form of a SysML activity diagram - this step introduces user-friendliness.

3. *Approval Phase*: the SysML activity diagram is approved by the involved stakeholders, if further improvements are necessary, they are added.

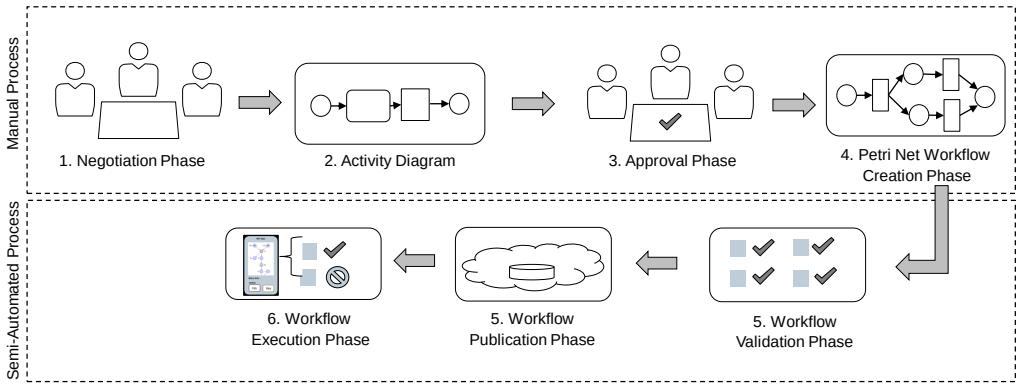



Figure 13.1: Different phases involved in WFAC

4. *Petri Net Workflow Creation Phase*: the SysML activity diagram is used as the reference to create the Petri Net workflows. In addition, these SysML could also be translated into Petri Net workflows using available tools and approaches.

5. *Workflow Validation Phase*: the Petri Net workflows can be validated using semi-automated tools that check workflow properties such as soundness and boundness. During this phase, several other steps such as who is authorized to execute the workflow, and other workflow Application Programming Interface (API) interfaces are developed and integrated, and also validated.

6. *Workflow Publication Phase*: after validation, the Petri Net workflows are published in a commonly accessible platform like a store, this thesis refers to it as a workflow store. For accountability and confidentiality purposes, a hash pointer of the workflow can also be published in a blockchain.

7. *Workflow Execution Phase*: this is the final phase of to use and enforce the WFAC procedure. It is important to ensure that the handheld capable of running the workflow application (capable of executing Petri Net workflows) is secure enough such that the workflow participant is not able to manipulate the workflow states or tokens to skip some steps of the workflow.

The WFAC framework when deployed shows how the workflow specified via Petri Nets can be executed and enforced.

### 13.1.1 WFAC Architecture

This thesis has developed the WFAC, and from the perspective of a resource owner it is shown in Fig. 13.2 which consists of the following five frameworks to support different phases presented in the previous section:

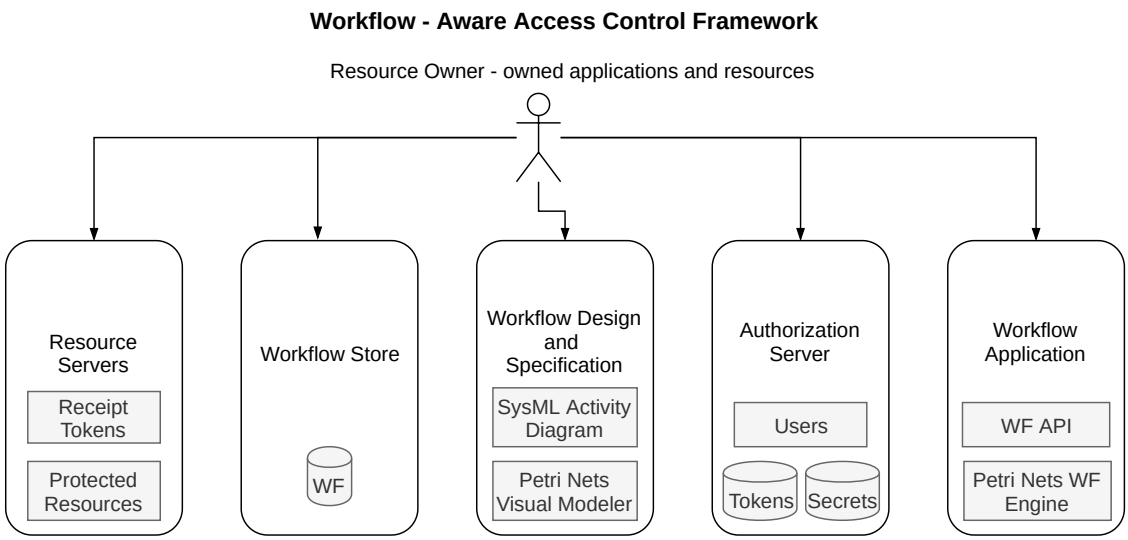



Figure 13.2: Workflow-Aware Access Control Architecture

- *Resource servers*: a resource server may have one or more resources that belong to a resource owner who wants to protect it. Several resource servers are usually involved in a distributed IoT use case where more than one resource owners are involved.

- *Petri Net Visual Modeling tool*: each resource owner wants to protect his own confidentiality and integrity goals, therefore, this thesis has developed a user friendly Petri Net workflow specification tool, also known as the visual modeling tool. This tool produces a Petri Net workflow.

- *Authorization Server*: a traditional OAuth server is extended to have workflow user management, and WFAC workflow specific features such as processing receipt tokens are integrated with it.

- *Workflow Store*: after Petri Net workflows are designed and validated, they are published in a workflow store for workflow participants to access. This workflow store ensures trust for the workflow participants that the workflows belong to the resource owner because they are published in resource owner approved workflow stores.

- *Workflow Application*: the application that is used to execute the Petri Net workflows, it is developed by the resource owner using secure development practices (SDL), it includes Petri Net workflow engine and workflow specific APIs that help to interact with other services and business logic.

All the above mentioned services and applications are owned by the resource owners and therefore are considered trusted.

### Actors

The two main actors involved in the WFAC are the resource owners and the workflow participants.

1. **Resource Owners**

   The RO is typically the owner of the resources i.e. RS therefore, RO is responsible for designing, creating, or specifying, and registering the Workflow (WF) before it can be used by the workflow participants.

   Some of the requirements of the resource owners are:

   - Each RO should be able to enforce workflow-aware access control restrictions on workflow participants accessing their RS.
   - RO can design their own workflows in collaboration with other resource owners.
   - RO can register and publish workflows in a Workflow store (WF-Store) and allow workflow participants to download workflows from it.
   - RO can register workflow participants, set permissions, and allow the workflow participants to execute their workflows. The authorization server framework supports and integrates this feature.
   - WP must only get access to resources by properly executing the workflow, not by bypassing the workflow or other means.

2. **Workflow Participants**

   The WPs are the entities that accept the workflow issued by the RO to access the R of RS. Some of the requirement of authorized WP are the following:

   - WP should be able to download the Workflow Application (WF-App) and download the necessary workflow from the WF-Store, and finally, execute the workflows.
   - WP should be able to recover from error conditions.

### Resource Server

In Fig. 13.3, the general architecture of a typical RS is shown. A RS contains one or more Resources (R). RSs can be constrained, therefore, they support only the minimum functionality such as validating access control tokens encoded as Java Web Token (JWT) or even precise Concise Binary Object Representation (CBOR) encoded CWT. Resource Servers may also produce and issue signed JWT or CWT tokens with predefined dynamic information.

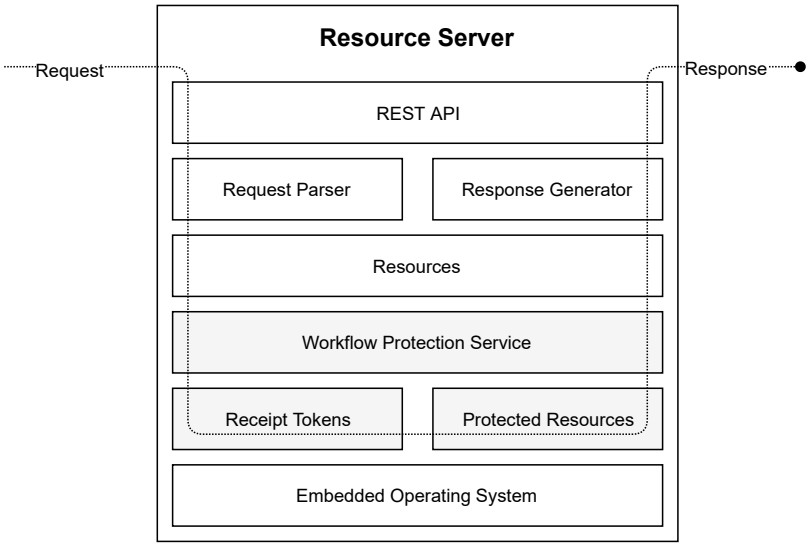



Figure 13.3: Resource Server - Architecture

In addition, to ensure that the workflow actions are committed by a workflow participant, the resource servers produce and issue signed *receipt tokens* - with some details encoded about the workflow participant, client device (if managed handheld), workflow action, issuer. Also, to access some resources, a resource server can be configured such it must check additional receipt tokens from previous workflow steps from the workflow participant or his/her client device in addition to the access token acquired from the authorization server.

### Petri Net Workflow Generation

In Fig. 13.4, the general architecture of Petri Net visual modeler and workflow generation architecture is shown. The RO can design and specify the Petri Net workflows using the Petri Net visual modeler tool, the tool can export the Petri Net workflows into an inter-exchangeable PNML format. This PNML file can be imported and executed in the WF-App.

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

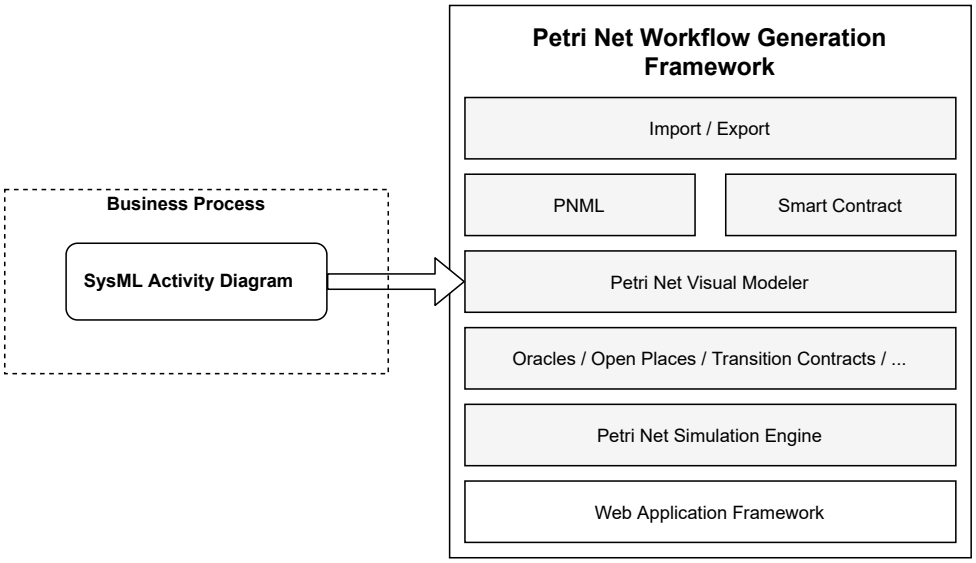



Figure 13.4: Petri Net Workflow Generation Framework - Architecture

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 [163, 164]. 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 analyzes the process or activity requirements, and verifies whether the designed workflow does the same as described.

### Workflow Store

The WF-Store acts as a distributed database of workflows published by different resource owners (ROs). The architecture of WF-Store is shown in Fig. 13.5. The workflow participants download the workflow from the workflow store and execute them. Only the resource owner can publish a workflow in the WF-Store, and this service could be similar to the traditional Smartphone application store, for example, the android play store. The resource owner provides necessary workflow attributes while publishing the workflow, the store creates a unique identifier for each workflow. This unique identifier can be used as a reference when the resource owner instructs the workflow participants to execute the workflow.

> **Example:**
>
> One example idea behind the workflow store is the following: for instance,

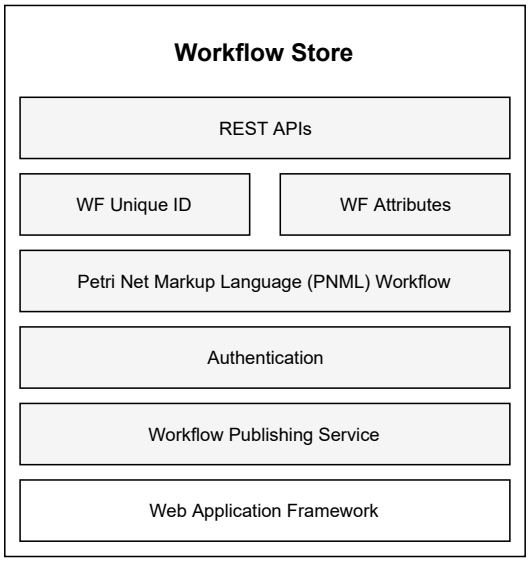



Figure 13.5: Workflow Store Architecture

> assume that there is a product (resource server), and the manufacturer (the resource owner) wants to enforce the customers (workflow participants) to obey a workflow for using their product devices. The workflow store provides the platform for the customers (the workflow participants) to download the workflow and start executing it with a generic workflow application.

### Authorization Server

The AS performs authorization decisions such as whether to grant a short-lived bearer access token to the Workflow Participant to access a particular resource while executing a workflow. In Fig. 13.6, the general architecture of AS is shown. How the authorization server authenticates the workflow participant or his/her client device is out of the scope of this thesis. Once the resource server successfully authenticates, he/she can register workflow by pointing towards the workflow identifier published in the workflow store, and manage which workflow participants should be allowed to get access tokens for which workflow. In addition, the resource owner specifies the scopes required to access resources, and receipt tokens that must be validated for each resource.

The main authorization decisions made by the AS specific to workflow-aware access control are the following:

- Is the WP authorized to execute the WF or not.
- Are the client-ID and client-secret provided by the WF-App is valid and sufficient to grant access to requested scopes.

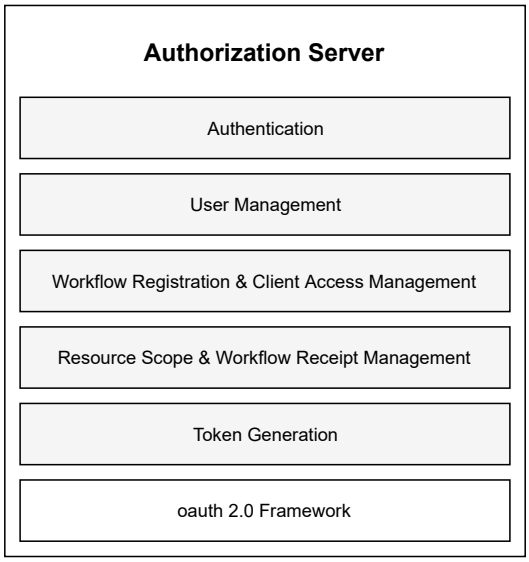



Figure 13.6: Authorization Server - Architecture

- Does the access request for a particular resource need additional history or receipt tokens or not - this is used to prove that a particular workflow action is completed by the workflow participant.

1. Revoking Workflow Rights

   Revoking a workflow participant to execute a workflow is an important requirement for a resource owner. The resource owner is allowed to delete the registered client to execute a particular workflow. This will prevent AS from issuing further new tokens to that particular client. However, already issued short-lived bearer tokens cannot be revoked - as the name suggests they are short-lived so it cannot be used after the expiry time.

### Workflow Application

The architecture of WF-App is presented in Fig. 13.7. The WF-App is developed and managed by the resource owner. The workflow application's main components are

- *Workflow Management & Execution GUI*: this helps the actor workflow participant to import workflows and start and stop executing it.
- *Authentication Service*: this service allows the workflow participant's client to authenticate with the authorization server, and download the necessary credentials to execute the workflow locally.
- *REST APIs*: interacts with the GUI and also with external resources and services, for example, with the resource servers and authorization servers.

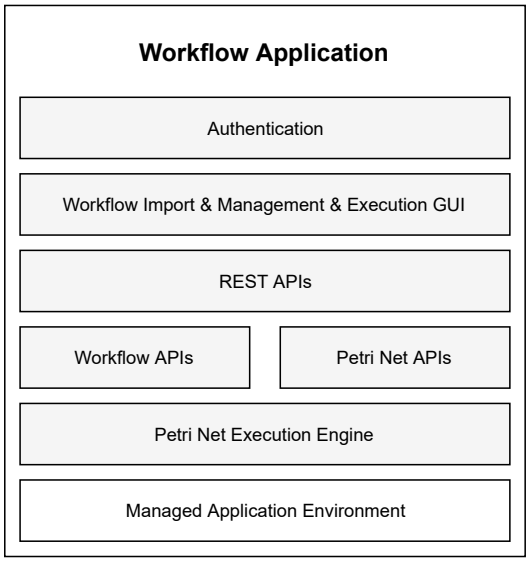



Figure 13.7: Workflow Application - Architecture

- *Workflow APIs*: workflow specific functions that are required to be used within the Petri Network workflow specification is provided via these inbuilt APIs. This component is modular such that external workflow specific functions can be loaded easily.
- *Petri Net APIs and Engine*: Petri Net workflow specification is actually executed with a Petri Net Engine and to interact with it, the Petri Net APIs are used.

So far, this thesis described the architectural aspects of the WFAC. In the next section, this thesis describes some aspects of the prototype implementation.

### 13.1.2 WFAC Framework Usage

This section presents how one can use the WFAC framework by presenting the flow of actions.

The following description corresponds to Fig. 13.8 describing the flow of actions of how one can use the proposed WFAC framework.

- 1(a): first, the Resource Owner (RO) designs the required workflow with the activity diagram of SysML, then the activity diagram is used to create the Petri Net Workflow using the visual modeler tool presented in Sec. 12.1.1. As described earlier, it is important to model the workflow to handle error-conditions. This process also includes the validation of the specified Petri Net workflow both using automated tools and expert review.

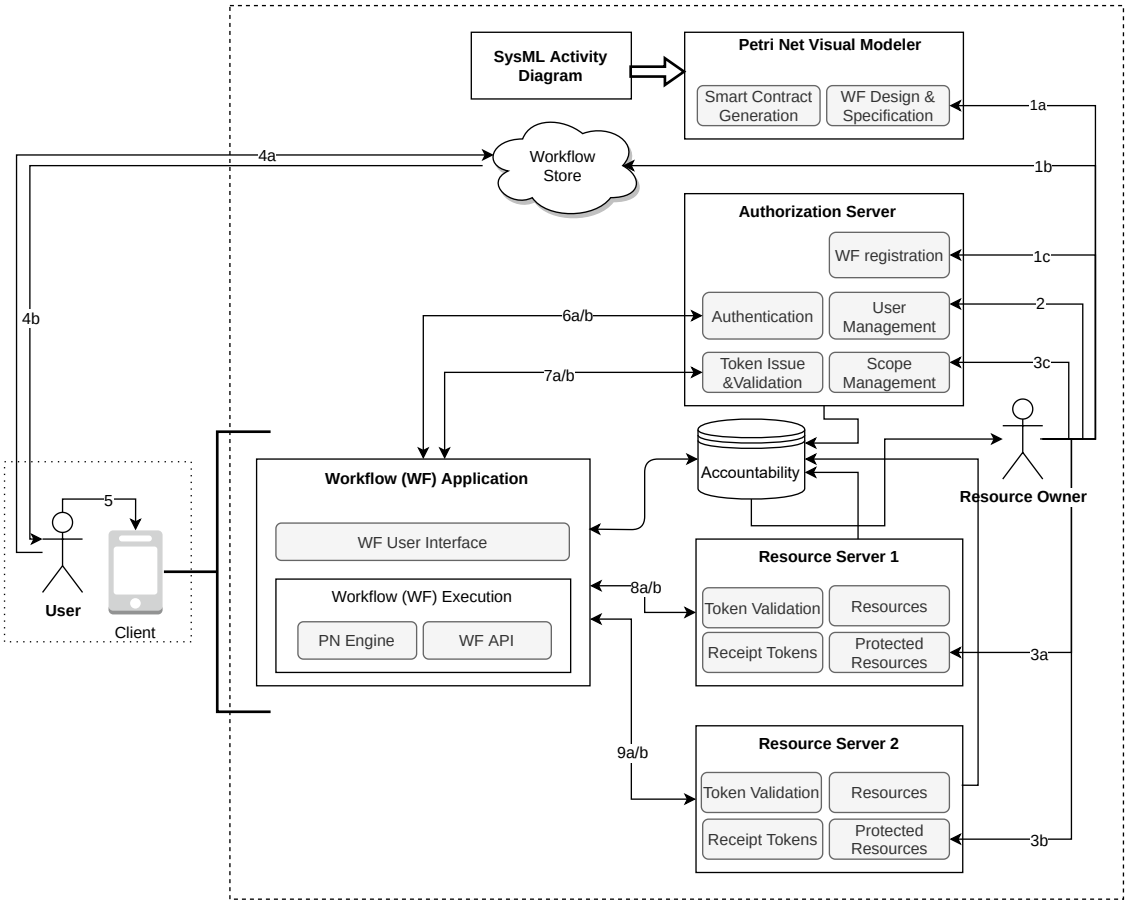



Figure 13.8: Workflow-Aware Access Control Architecture and its flow describing how one can use it.

- 1(b): RO then publishes the validated workflow in the workflow store.
- 1(c): RO registers the workflow with the Authorization Server (AS).
- 2: RO creates WP and assigns the workflow related permissions to the registered client. The AS creates a unique ID and secret for each registered workflow to a client. In OAuth terms, the unique-ID and secret are called the client-id and client-secret.
- 3(a/b): RO configures the resource servers and the AS with scopes, receipts that should be checked for each particular resource. In addition, RS is configured with information that should be inserted into the receipt token which will be issued to the client after successful workflow action completion.
- 3(c): during this phase, the RO also decides the access control permissions needed for each workflow step that the Workflow Participant (WP) or a user needs to execute the WF.
- 4(a/b): WP looks up and downloads the registered workflow.
- 5:WP imports the downloaded workflow or by simply providing the URL of the workflow.
- 6(a) The workflow participant successfully authenticates to an authorization server, as a result 6(b) a client-id and client-secret and other relevant tokens are pushed into the application. This connection is an end-to-end connection with the Workflow Application (WF-App) therefore, the workflow participant is not able to see this information. This secret material is later used to identify the client and the requests originating from the workflow application.
- 7(a/b): Furthermore, based on the implementation and use case specific requirements several access tokens that might be required to execute a workflow can also be pushed to the workflow device after successful authentication and authorization.
- 8(a): WP via the WF-App performs an access request to the resource and presents the token received from the AS
- 8(b): if the presented token is valid, the RS returns the valid resource with an additional receipt token signed by the RS - this receipt token proves that the WP has performed a particular workflow action
- 9(a): WP via the WF-App performs an access request to the resource and presents the token received from the AS + the receipt token received from the resource server 1 - in the previous step.
- 9(b): if the presented token and the receipt token are valid, then the Resource Server (RS) returns the valid resource. Based on the configuration, resource server 2 may also issue an additional receipt token signed by the RS2.

In addition, AS and RS can publish important workflow events into an immutable database, i.e., accountability database shown in Fig. 13.8. Depending on the use case, the WP with the WF-App credentials it receives from the AS, is able to publish workflow events e.g., in case of an error or usage or error-recovery tokens.

## 13.2 WFAC Implementation

The goal of the prototype is to demonstrate the feasibility of the WFAC framework. Therefore, the design choices that include software platform, frameworks, programming language, etc. were decided only to demonstrate the feasibility, and not to make a production-ready software application.

This thesis assumes that the communication interface between the WF-App, authorization servers, and other services are protected by standard security protocols such as TLS. DTLS together with the PAT profile is recommended to communicate with the constrained resource servers. Note: PAT profile can also be used without a DTLS communication. This thesis also assumes that the WFAC applications are developed using best security practices. The exploitation of implementation bugs or the underlying operating system to gain access to internal application services and the resulting attacks are not considered as part of this implementation.

As part of the WFAC framework implementation, the following web applications are developed:

- (a) An Authorization server web application with its functionalities described in Sec. 13.1.1. Each resource owner may have their own AS.
- (b) A workflow management application that integrates both the Petri Net visual modeler as presented in Sec. 12.1 and the workflow store (WF-Store) application as described in Sec. 13.1.1. Each resource owner may have their own WF-Store.
- (c) A resource server application with multiple protected resources. In the developed prototype, one can simulate multiple resources.
- (d) The workflow web/mobile application. The workflow application is critical for the WFAC framework, therefore it is explained in detail in the following Sec. 13.2.

The aforementioned web applications were developed using Python web application framework “FLASK” and several other python libraries. Node-js was used for developing the Petri Nets visual modeler.

### 13.2.1 Demo Use Case

Consider a simple use case where a building owner delegates installation or maintenance work to a contracting company. The RFC 7744 [165], 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 with the 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 Sec. 15.1.

Different phases involved in the WFAC framework are described in Fig. 13.1. As the use case is identified, a requirement elicitation is conducted with relevant stakeholders, with which requirements are identified. A business manager or software engineer might use the traditional UML based modeling approach to model the first activity diagrams. This thesis adopts the SysML activity diagram and recommends it to model the activity diagram. Next, 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. After that, a Petri Net library (e.g., see Sec. 13.2.2) can be used for implementing Petri Net functionalities and enforce the conditions written within them. 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 WF-Store or implemented as smart contracts and then published into 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.

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. 13.9 to execute the WF. The user or workflow participant executing the workflow needs to authenticate via the App (i.e., to prove that the user can execute the workflow) to the Authorization server. After successful authentication, the WF-App receives the client-id and client-secret from AS which is used for further requests to AS.

The building automation devices use the standard ACE-OAuth [47] protocol to validate the token that it receives, and if the tokens are valid, then access to a 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. 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.

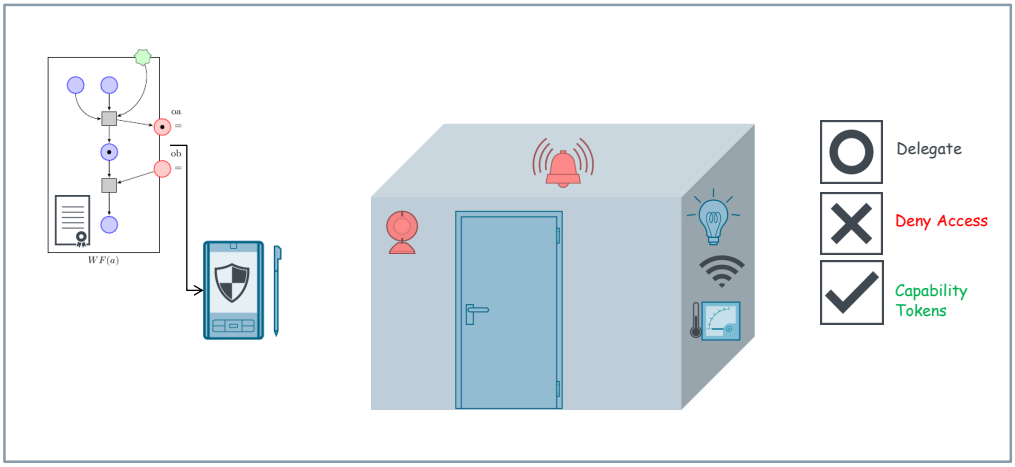



Figure 13.9: Building Automation - Petri Net Workflow Enforcement - access denied or granted based on the workflow specification.

### 13.2.2 Petri Nets Library

The Python library called "SNAKES", presented in [166] by Pommereau, is used to execute the Petri Nets workflows written in PNML. The SNAKES library is extended to support different types of Petri Net places such as oracles and open places and to evaluate transitions with external conditions and produce the necessary tokens that will be required for subsequent transitions. In the current implementation, the transition contracts are expressed with limited features of SNAKES library's arc notations, expressions that can use native python functions to evaluate input tokens and produce required tokens. 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 are needed.

Assume that different stakeholders are providing their services as Representational State Transfer (REST) based web services through the WFAC services such as WF-Store and supported via their authorization servers. The workflows are created by workflow experts, approved by the stakeholders, and made available via their own workflow repository WF-Store. A participant can download the WF-App and the required workflow from the respective resource owner's WF-Store repository, and then he may start executing the workflow. The WF-App front-end provides the communication interface between different external services and the RS. standard security protocols are used to protect the communication channel. How participants authenticate with the Authorization server is out of scope here. The enforcement of the Petri Net workflows is integrated with the WF-App.


Figure 13.10: The demo building maintenance use case that shows 3 steps, after completing each step, the workflow participant receives a receipt token. In this figure, one could see that the tokens are represented in different shapes e.g., circle, square, and triangle. The collected tokens are further used in the next request.

### 13.2.3 Workflow Application

The workflow application is built to execute workflows specified as Petri Nets, and it can handle oracle, timeout transitions, and other features introduced in this work. Weber et al., [167] introduced PNML which is based on XML, and in this work, we use PNML to express the Petri Net workflows. The application is created by the resource owner RO therefore, it is trusted and is allowed to run in a certified device. The workflows being executed within the WF-App is recommended to run on a certified device that contains a protected execution environment such as the Trusted Execution Environment (TEE) to do some sensitive operation involving the secret material. Thus, we assume that the participants are not able to extract or modify any secrets from the workflow. This certification ensures that the secrets enclosed within the WF-App cannot be easily extracted by the CL. In Fig. 13.7, the general architecture of WF-App is shown.

The implementation uses the ACE-OAuth protocol to create JWT tokens [168]. These proof-of-possession tokens are used by the workflow participant to prove to the resource server that the workflow participant 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 Fig. 13.10 is showing the demo use case, where a workflow participant must execute the workflow actions to gather the receipt tokens, with which the access to a final resource (entrance to a room with a smart door) is granted.

The workflow application controls the workflow execution step such as the following:

- Whether the workflow (WF) is executed properly or not?

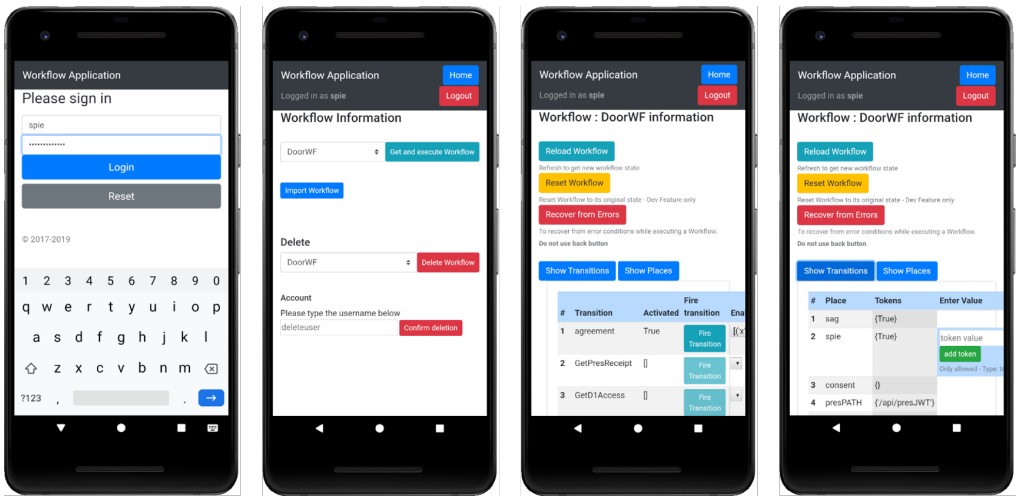



Figure 13.11: The prototype of the WFAC workflow application installed on an Android smartphone.

  - Does the WP need to access this a resource R in RS to execute the next step in the WF?
  - Does the response from previous WF step is correct or not by validating the receipt tokens? This information is used to enable the next state in the WF. For example, the response from the resource request from a RS may include a signed JWT with information such as workflow execution state, next accessible resources, etc.

- The client-id and client-secret information are used by the WF-App to authenticate itself to the AS; in addition, when requesting access tokens the WF-App will also request the required scopes. Note: how this client-id and client-secret is provided to the WF-App is out of the scope of this thesis.

The prototype of a workflow application typically installed in a handheld or a mobile device (e.g., an Android smartphone) is shown in Fig. 13.11. This prototype WF-App is developed as part of this thesis to evaluate the the WFAC framework. However, this application can also run as a web application. Python application development framework 'Flask' is used to build this workflow application. This WF-App is used by the workflow participant to execute the workflow and access protected resources.

Figure 13.11 shows four views of the workflow application installed in the form of a mobile application (e.g., in Android smartphone) in a handheld. The first view of Fig. 13.11 shows the login page of WF-App using which the user presents the credentials to the AS. If the user is allowed to execute the workflow, then the AS presents appropriate workflow application credentials (e.g., OAuth client-credentials with limited scopes) to execute the workflow. The credentials are transported via TLS and stored within the application in such a way that they are not

accessed or tampered by the WP or other applications installed in the handheld. The second view from the left of Fig. 13.11 shows that the user *sag* logged in, therefore, can import or delete existing workflows. The third view from the left of Fig. 13.11 shows the current status of DoorWF workflow for the user *sag*. Also, one can notice that the agreement is enabled (activated) because *sag* and *spie* both have agreed to the Petri Net workflow by presenting a 'True' token in the appropriate places. The WF-App also shows options (via show buttons) to view transitions and places, recover from errors, and view the workflow state as the Petri Nets. The WF-App application includes an option for the users to insert tokens in authorized places, for instance, in the fourth view from left in Fig. 13.11, as *sag* user is logged in, the user is able to insert tokens into place *sag*. This prototype version also includes development only features such as reset workflow. Additionally, the workflow application also allows the user to see the workflow state in the form of Petri Net workflows.

# Part IV

# Evaluation

# Chapter 14

# Security Analysis of Workflow-Aware Access Control Framework

This chapter evaluates the security of the proposed WFAC framework. To perform this security analysis, this thesis has used both the practical approach in Sec. 14.1 and Sec. 14.2 and the theoretical approach in Sec. 14.3 respectively. First, the assumptions considered when developing the WFAC framework are presented. Second, the results of the WFAC framework's threat model and web security analysis are discussed. Third, attacker models of WFAC are discussed. Finally, the countermeasures and recommendations are presented to improve the security of WFAC.

This thesis assumes that the resource owner and associated entities are trusted and secure. Trusted entities are assumed not to collude with an adversary or attack the system themselves. The WFAC frameworks and its applications deployed on the servers are trusted. Therefore, the authorization server, workflow-store, workflow modeling systems, workflow experts, and their administrators with special privileges are also trusted.

In the performed security analysis, common security issues such as code bugs and vulnerabilities that exist in the different software or operating systems are not considered. It is assumed that appropriate security measures are taken during software development.

## 14.1 Threat Modeling

This thesis has used Microsoft's threat modeling tool (see [169]) to analyze potential threats present in the WFAC framework. Microsoft's secure development lifecycle (SDL) is considered the industry's best practice for secure software development.

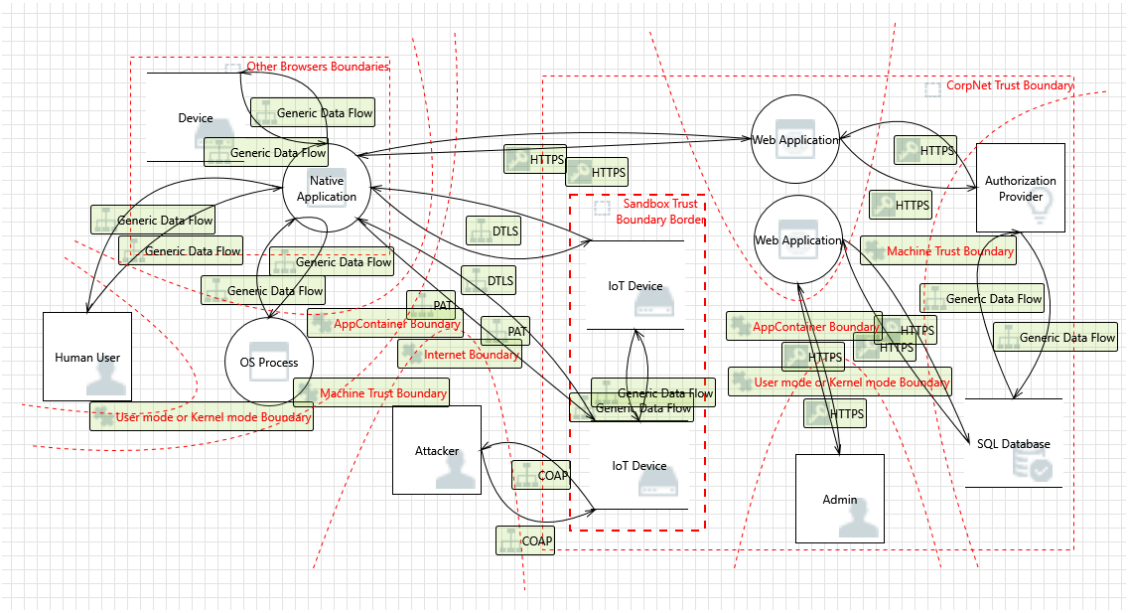



Figure 14.1: Microsoft Threat Modeling Tool - WFAC architecture threat model diagram

One of the twelve recommended security practices of SDL (see [170]) is *threat modeling*. Microsoft's threat modeling focuses on finding and thwarting threats related to Spoofing, Tampering, Repudiation, Information Disclosure, Denial-of-Service, and Elevation of Privilege (STRIDE) [171]). STRIDE is a proven and effective approach to detect different threats (see [172, 173]) that exist in common software applications.

Figure 14.1 shows the WFAC deployment architecture drawn and analyzed using Microsoft's Threat Modeling tool. The tool provides insight into potential threats and also recommendations to avoid those threats.

The tool generates a generic report which states the specific problem and the countermeasures that can be applied to solve the problem. The tool also highlights the problem by showing only a snapshot of the particular part of the complete WFAC architecture threat model diagram. For example, in Fig. 14.2 the tool highlights the Structured Query Language (SQL) injection vulnerability between a SQL database and the authorization server.

As a result, in total, the tool reported 149 possible threats that can be classified into one of the STRIDE categories, i.e., classified into 19 in Spoofing, 20 in Tampering, 24 in Repudiation, 17 in Information Disclosure, 40 in Denial-of-Service, and 29 in Elevation of Privileges Fig. 14.3 shows the number of threats. The identified threats do not mean that they exist in the developed WFAC framework, rather it shows that these threats exist, and proper measures must be taken to resolve them.

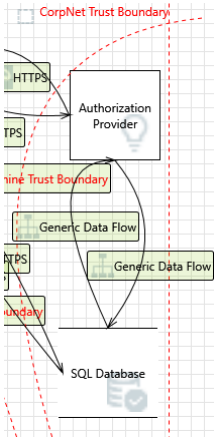



Figure 14.2: Microsoft Threat Modeling Tool highlighting SQL injection vulnerability present between the Authorization Server and the SQL database, for example, with credentials

Most of the spoofing related threats can be resolved by implementing standard source and destination authentication mechanisms for properly identifying processes, storage, or databases. For the mentioned purposes, Public Key Infrastructure (PKI) based certificate authentication methods are usually used. Similarly, tampering related threats can be resolved by using anti-replay mechanisms, sanitizing input and output data, integrity measures, logging, using proper pointer or memory access management, etc. To fix repudiation related threats issues, accountability measures that include logging of actions committed should be integrated. To resolve information disclosure related threats proper access control and accountability measures must be used. Similarly, data replication, load balancers, replicated servers, crash detection, auto start processes are some of the passive approaches that could be used to tackle Denial-of-Service (DoS) related threats. Stopping a threat before it occurs is an alternative and active approach to restrict Denial-of-Service (DoS) attacks, in such a situation, a legitimate user may be denied access to services. The least privilege mechanism must for enforced for processes and threads to perform updates and patches to tackle privilege escalation related attacks. Disabling all unnecessary capabilities and features will prevent some threats.

## 14.2 Web Security Analysis

This thesis has developed a WFAC framework prototype that includes different web applications: (a) an authorization server, (b) workflow application, (c) resource server, and (d) a workflow design and specification application based on Petri Nets. The workflow application is also ported to a handheld, for instance, an Android application. This thesis has used one of the popular web application


Figure 14.3: Number of different STRIDE threats identified by the Microsoft Threat Modeling tool on the WFAC threat model diagram

Table 14.1: Google Web Security Scanner Results of WFAC framework performed in December 2019.

| Application | URLs tested | Duration | Vulnerabilities Found |
|---|---|---|---|
| Authorization Server | 318 | 29 min 32 sec | 0 |
| Workflow Application | 457 | 1 hr 37 min | 0 |
| Workflow Management | 191 | 15 min 42 sec | 0 |
| Resource Server | 66 | 8 min 23 sec | 0 |

frameworks based on a python programming language called "Flask" [1] to develop the aforementioned web applications.

The OWASP foundation publishes the top 10 vulnerabilities found on web application security [2] to create awareness amount the developer's community. To test whether the developed web applications contain known web application security vulnerabilities this thesis has used Google Cloud's state-of-the-art Web Security Scanner [3] to automatically test the known vulnerabilities as follows:

- Cross-site scripting (XSS)
- Flash injection
- Mixed-content
- Clear text passwords
- Usage of insecure JavaScript libraries

The Google Web Security scanner is not intended to replace manual code inspection or other types of evaluation, and it does not guarantee that the applications are free of any vulnerabilities. Table 14.1 shows the scan results performed on 5th December 2019 of four different web applications. The Google Web Security Scanner was provided with login credentials for applications that have a login interface. Therefore, the internal state changes of the web applications were also tested. Note: as Google's Web Security Scanner is constantly improving, the scan results may differ from the time this scan was performed.

This thesis does not consider other types of testing approaches such as application testing such as blackbox, whitebox, and penetration tests, or the code analysis techniques as Static vs. Dynamic Analysis of the WFAC, which are also popular methods to detect and fix vulnerable bugs in software applications used. However, as the intention of this thesis is to develop a proof-of-concept and not a production ready software application. Therefore, this thesis has not performed such in-depth production quality software testing. However, when this WFAC is used in

[1] `https://www.fullstackpython.com/flask.html`
[2] `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`
[3] `https://cloud.google.com/security-scanner/docs/overview`

production, the thesis recommends professional software development and testing approaches to reduce bugs or other vulnerabilities.

## 14.3 Attackers

An attacker can be classified based on his capabilities, motivation, skills, level of access, and intentions. A generic attacker taxonomy is presented in Fig. 14.4. The taxonomy provides a broad overview and helps to select the attacker models for the WFAC.

### 14.3.1 Attacker Models

The Dolev-Yao attacker model is considered to be one of the strongest attacker models for analyzing the cryptographic protocols, in particular, it considers an active network attacker who impersonates to be a legitimate user altering the communication for his/her own benefit (see [174]), but for the purposes to analyze the proposed WFAC system more than a network attacker is required. This thesis has chosen three different attacker models from the taxonomy presented earlier.

In each attacker model, the assumptions of the WFAC are the same, and only the level of access the attacker has to the components of WFAC differs. This thesis assumes that the attacker has limited resources, i.e., the attacker is not state-sponsored. An insider such as the administrator who has extreme capabilities is not considered an attacker in the chosen attacker models. This thesis assumes the resource owners and associated entities as trusted. This thesis assumes that the workflow participants executing the workflow via the trusted workflow application may act maliciously. Therefore, the main focus of the chosen attacker models is to investigate how a malicious workflow participant may attack the overall system.

The assumptions presented earlier are used to generalize the attacker's skills, capability, attack type, and motivation. In the following, they are described briefly:

- *skills*: professional, i.e., with above average skills to attack the WF-App or the IoT devices.
- *capability*: a lone attacker or a small dedicated team.
- *attack type*: both passive attacks such as monitoring and active attack such as tampering or altering the input data to devices and services
- *motivation*: usually financial, but in most cases, the motivation depends on individual use cases of the workflow and the resources involved.

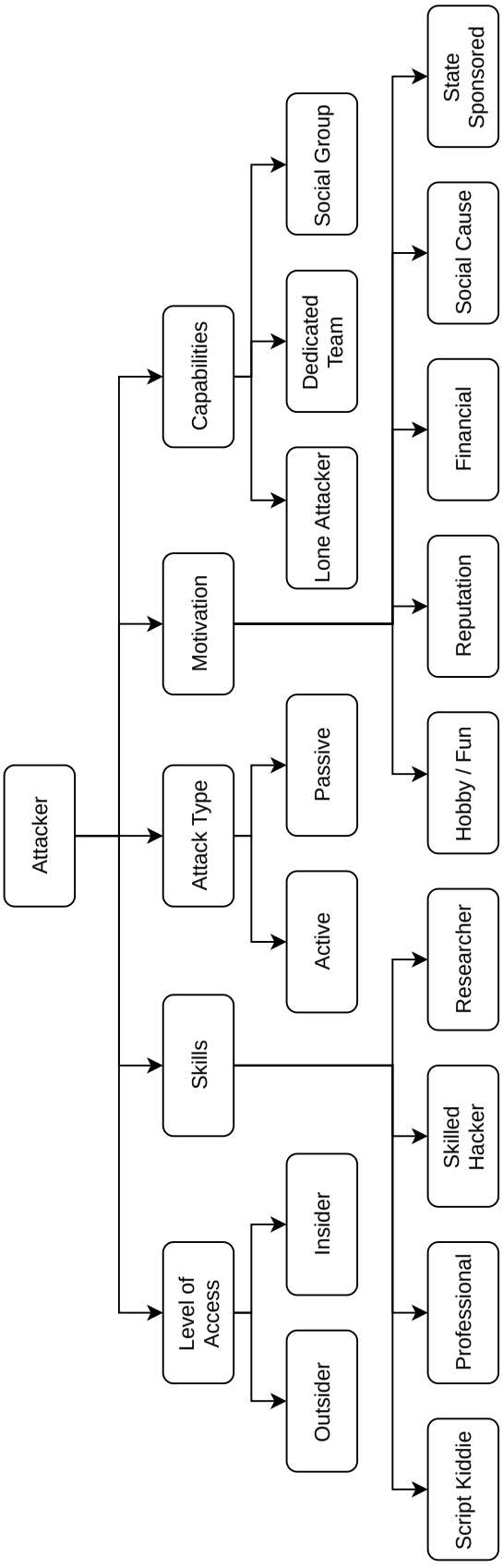



Figure 14.4: A Generic Attacker Taxonomy

The three chosen attacker models are differentiated based on the level of access the attacker has to the WFAC or its components.

- *Attacker-1 (A1)* has no access to the workflow application, handhelds, or any workflow information. Therefore, the attacker can, at most, access the IoT devices or services exposed physically or virtually via the Internet, respectively.
- *Attacker-2 (A2)* has partial access to the workflow execution, i.e., access to the handhelds with the workflow information but does not have any workflow related credentials. One can assume this scenario with the stolen handheld.
- *Attacker-2 (A3)* has complete access to the handheld, workflow application, and also holds legitimate credentials to execute a workflow. You may think this attacker as a legitimate workflow participant who acts maliciously. This attacker is the most powerful out of the chosen three attackers.

Now, this thesis discusses the attack scenarios with the chosen attackers.

Irrespective of how good the protection mechanisms are, the attacker will try to find a way to compromise the system until he/she is successful - this is based on the fact that no computer system is 100% secure. This is one of the reasons why an attacker with unlimited resources is not considered.

### 14.3.2 Attacker-1

The attacker-1 has three possibilities to attack the IoT devices and services protected with the WFAC framework.

- First, the attacker can brute force to get access to the device via the Internet, but as the devices are protected using our WFAC, the attacker cannot possess the workflow related tokens, in particular, *receipt tokens* without executing the workflow. Therefore, this attack cannot succeed without having workflow credentials.
- Second, the attacker may use a vulnerability to gain access via the Internet or with physical access to one of the IoT devices. Assume that the attacker has gained access to that particular IoT. Even in this scenario, the attacker cannot perform the lateral movement to gain access to other internal IoT devices because the attacker does not have workflow knowledge or the ability to execute the workflow.
- Third, the attacker can eavesdrop on non-encrypted channels, which will allow the attacker to capture short-lived tokens and replay them to resource servers as authentic requests. This attack affects constrained IoT devices that may not have the capability to encrypt the communication channels. But even

those devices will be using the PAT profile for OAuth 2.0. - introduced in this thesis for protecting constrained devices, as well as to protect the identity of the workflow participant. PAT uses proof-of-possession key to encrypt the responses. Therefore, the attacker may not be able to read the response from IoT devices. However, this attacker can perform denial-of-service or battery drain attacks.

### 14.3.3 Attacker-2

This attacker-2 has access to a handheld with the workflow application but does not have the credentials to access the handheld or the workflow - usually, the workflow participant needs to authenticate against the authorization server before starting to execute the workflow, and in return, some temporary workflow credentials are transferred to the handheld.

This thesis assumes that the workflow application does not have any static or long-lived workflow related secrets within the handheld. At the time of workflow execution, a workflow participant has access to the short-lived access tokens, and receipt tokens. If the attacker is able to break into the handheld and extract some of those credentials and tokens before they expire, then the attacker can access some resources for a short amount of time.

As these credentials are temporary, with the help of an authorization server, some of these credentials can be revoked. Therefore, further retrieval of tokens by the attacker can be prevented. Also, this problem can be thwarted with the use of hardware and software integrity mechanisms provided by the TEE or Trusted Platform Module (TPM). For example, if the mobile application detects any unauthorized changes to the approved handheld models, then the adversary has little chance to extract any valuable information.

### 14.3.4 Attacker-3

This attacker-3 is the most interesting attacker model who has access to a handheld, workflow application, credentials to authenticate against the authorization server, and legitimate access to a workflow. Let us assume that the attacker has access to Workflow-1 (WF-1). Now, the attacker may try to compromise other IoT devices or service which are not part of the WF-1, i.e., that the attacker can try to access devices or resources that belong to Workflow-2 (WF-2), etc. To do this, the attacker has two options: (a) compromising an IoT device and extract the secrets, and with those secrets may try to access WF-2 resource; (b) as a legitimate workflow participant, the attacker can try to misuse the error handling capabilities supported in WFAC. In the following, we discuss those aspects in detail.

### Compromising an IoT Device

Assume that the attacker has authorized access to workflow-1 (WF-1). Using WF-1, one can access the following devices *a, b1, and c1* in a sequence. Using Workflow-2 (WF-2), one can access the following devices *a, b1, and c2*. Let us assume that the attacker wants to access a confidential resource from an unauthorized workflow, in this case, a resource from device c2 of WF-2.

**Goal**: the attacker wants to access an unauthorized workflow resource by compromising one of the devices that belong to his authorized workflow.

Assume that the attacker has compromised one of the IoT devices of WF-1 that bridges the two workflows, for instance, device *b1* is compromised and thus with credentials from b1, the attacker could create a fake receipt token *RcpTkn-b1:(for WF-2)* - indicating that previous workflow WF-2 steps have been completed properly. The receipt token (*RcpTkn-b1*) alone from device b1 is sufficient to access any resource in c1 but not sufficient to access any resource in c2 because c2 is configured to process all previous receipt tokens, i.e., *RcpTkn-a1, RcpTkn-b1*. This configuration introduces additional overhead for the resource server c2 because it has to process all additional tokens but provides additional security for the resources. This configuration decision and the trade-off between security and performance are left to the resource owners.

In the assumed example case, the attacker has the ability to create fake receipt tokens from device b1, for example, *RcpTkb-b1:(for WF-2)* but cannot do the same for creating *RcpTkn-a1:(for WF-2)* - because it is not compromised. The attacker's request to access a resource from c2 fails in this case because the *Rcptkn-a1:(for WF-1)* is issued for completing workflow *WF-1*, and therefore, cannot be used as combined with *RcpTkn-b1:( for WF-2)*.

Given the integration of strong accountability measures, such action can be configured to trigger an automatic action against such malicious workflow participants. The receipt tokens are introduced in the WFAC framework to protect attacks from such compromised IoT devices.

### Misusing Dynamic Workflows

Dynamic workflows are introduced in the proposed system to handle error conditions and exceptions. If the use of dynamic workflows is allowed without proper accountability and additional access control restrictions, then it could be exploited by an attacker to bypass the workflow. The attacker should have a valid workflow execution contract (which could have been stolen from a legitimate client) and convince the administrator to provide a dynamic workflow (e.g., to overcome specific restrictions made available in the original workflow) to exploit this vulnerability successfully.

A general approach to tackle this issue is the following, (a) restrict employees with privileges to issue dynamic workflows, (b) compulsory training to those privileged employees to identify misbehaving entities (workflow participants) or social engineering attacks, (c) enforce strong accountability measures to collect information and cross checking whether the request is legitimate. Also, every action related to dynamic workflow, prior errors, and exceptions must be properly documented for auditing purposes. The next version of the workflow may consider the exceptions that occurred earlier, if they are frequent, then it is usually a good idea to incorporate them into the main workflow itself, such that dynamic workflows may not be invoked frequently.

## 14.4 Recommendations

It is recommended to follow best security practices and guidelines to avoid common threats and vulnerabilities. For instance, it is recommended to follow a secure development life cycle (SDL) that includes threat analysis, secure coding guidelines, and both hardware and software hardening guidelines. This thesis recommends that the workflow application be run on devices with hardware integrity protection mechanisms such as TEE and TPM. The hardware integrity mechanisms check the integrity of the underlying operating systems and applications, and whenever the integrity checks fail, appropriate prevention mechanisms such as boot prevention or data deletion are triggered.

# Chapter 15

# Use Cases

This chapter presents the concrete use cases that were used to collect the generic challenges and requirements of emergent IoT applications. The challenges and requirements helped to build the proposed WFAC framework presented in Part III. In this section, after describing each use case, the proposed WFAC framework is evaluated, i.e., by first presenting the SysML activity diagram, then describing the high-level Petri Net workflow with additional sub-workflows and explaining how each actor interact and execute the workflows. Finally, how this approach can be used for generic use cases is discussed. The four use cases presented and evaluated in this thesis are the following:

1. Building Automation
2. Connected Mobility Lab
3. Car Sharing
4. Supply Chain

Each use case can be slightly different in terms of its challenges, requirements, owners, resources, IoT devices, and their capabilities, and the involved workflow participants.

## 15.1 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

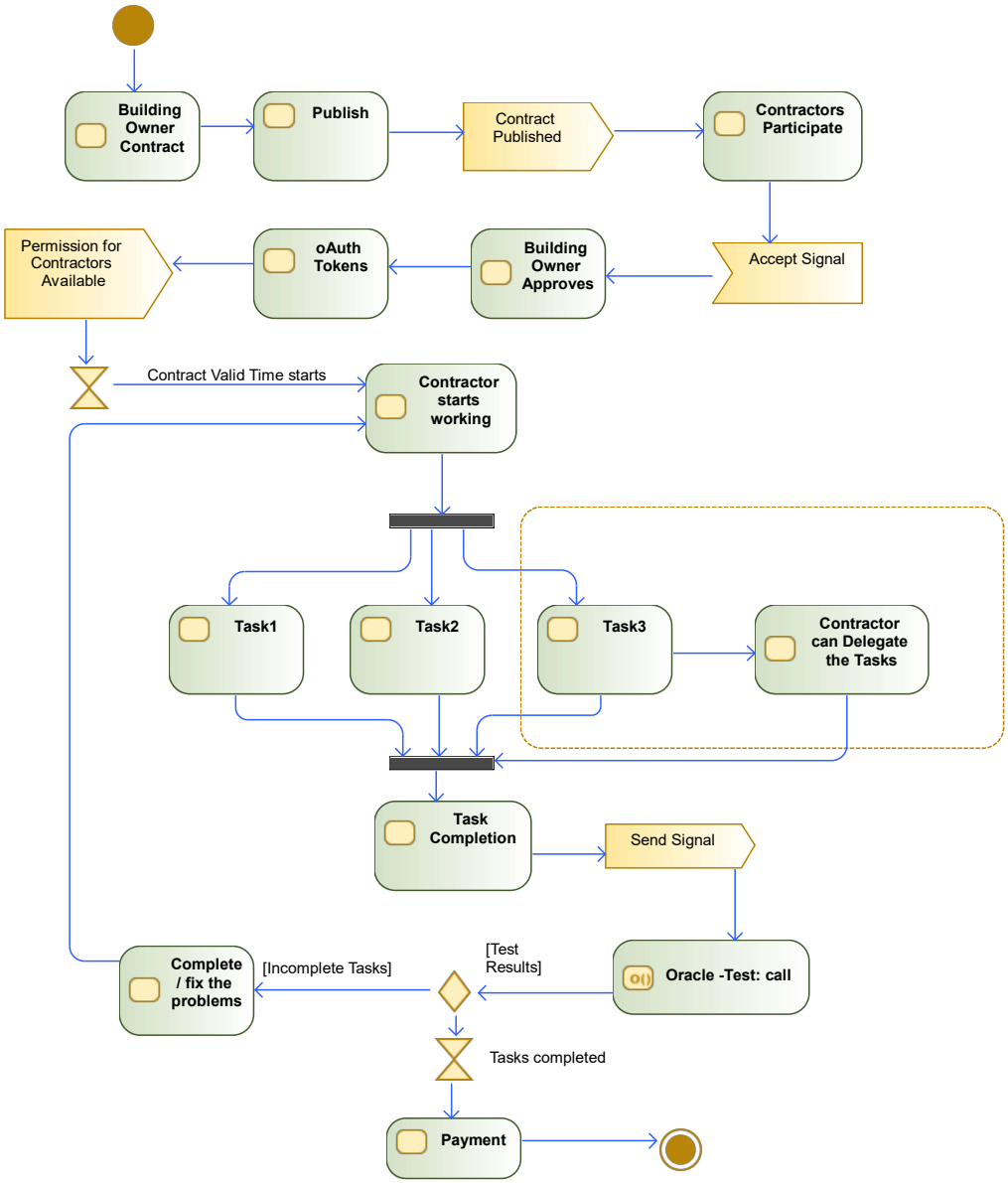



Figure 15.1: SysML Activity Diagram of Building Automation

on the devices. Usually, the building owner delegates the installation or maintenance work to a contracting company. The RFC 7744 [165], provides a summary of authorization problems that emerge during the device life-cycle (commissioning, maintenance, recommissioning, decommissioning). In addition to the authorization problems, building owners may wish to ensure that only products with a certain provenance or quality are used for installation and that the process complies with standard operating procedures.

The building owner also wants the contractor to obey the conditions agreed. The conditions also include some penalty if the contractor fails to satisfy the agreed

condition. The conditions agreed, including the penalty, are specified with the proposed Petri Net workflows.

For instance, the building owner

- wants to automatically enforce the conditions agreed with the contractor including any penalty if agreed conditions are not met,
- 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,
- wants to control authorization permissions given to the contractors enforcing fine-grained access control, i.e., the least privilege principle,

First, the requirements elicitation is conducted to gather the requirements. Next, a SysML activity diagram is created as shown in Fig. 15.1. If multiple different building owners are involved, then the activity diagram is discussed and agreed with everyone of them. Next, the building owner responsible for creating the workflow, with the help of workflow experts, creates the Petri Net workflow (*WF-BA*). The partial Petri Net workflow is shown in Fig. 15.2. The workflow WF-BA is then published in a workflow store. The workflow application certified by the building owner is downloaded and executed by the contractor (workflow participant).

Below, the steps involved in the workflow are explained:

- Once the workflow is published, the workflow participants (contractors) evaluate the workflow and decide whether to participate or not. If a contractor is interested, then he/she signs the token using his/her private key, this signed-token is placed in the place (CO) in Fig. 15.2.
- Next, let us assume that the building owner selects one of the contractors based on the 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 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).

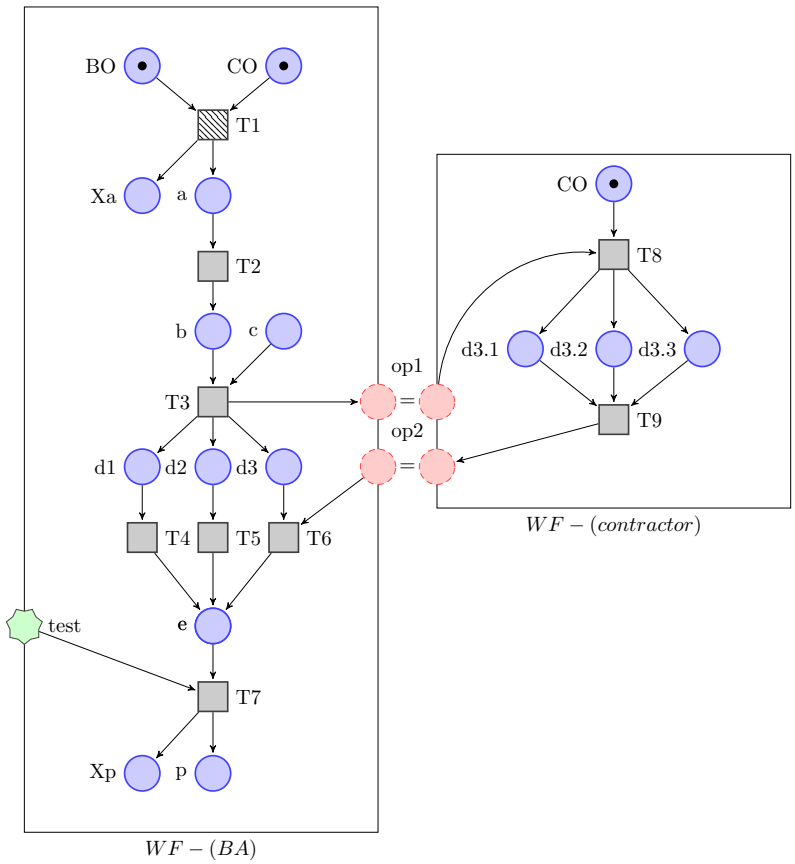



Figure 15.2: Open PN workflow (WF-BA) of Building Automation

- 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 give 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 Fig. 15.2 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 test 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 particular error conditions). The open places introduced in the main workflow 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. 15.2) to allow open places from authorized participants. Figure 15.2 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, a distributed workflow management system is realized. This use case shows how one can execute and enforce a workflow in a distributed way.

## 15.2 Connected Mobility Lab (CML)

The Connected Mobility Lab (CML) is a public funded project that envisioned integrating the services from different stakeholders – such as mobility, financial, and IT services – providing a comprehensive mobility solution by seamlessly exchanging data and analytics (see [175]). 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. 15.3.

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 payment is done differently by each mobility service provider. It is essential to guarantee the process integrity of such processes defined by each service provider. Therefore, Workflow-Aware (or Workflow-Driven) Access Control (WFAC) is required with a high-level workflow specification language.

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

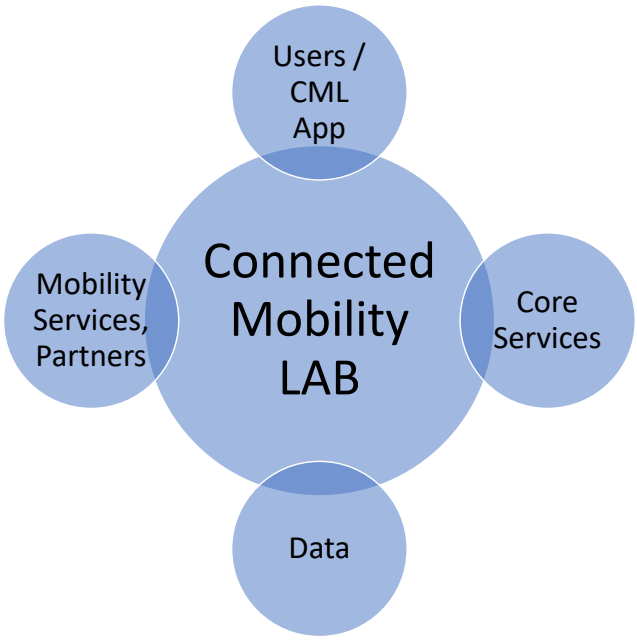



Figure 15.3: The Connected Mobility Lab (CML) offers a comprehensive mobility service by integrating different mobility service provides, partners using its core services and CML App.

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 particular 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 costs after a successful trip should be automated.
- 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 activity diagram. The open-source

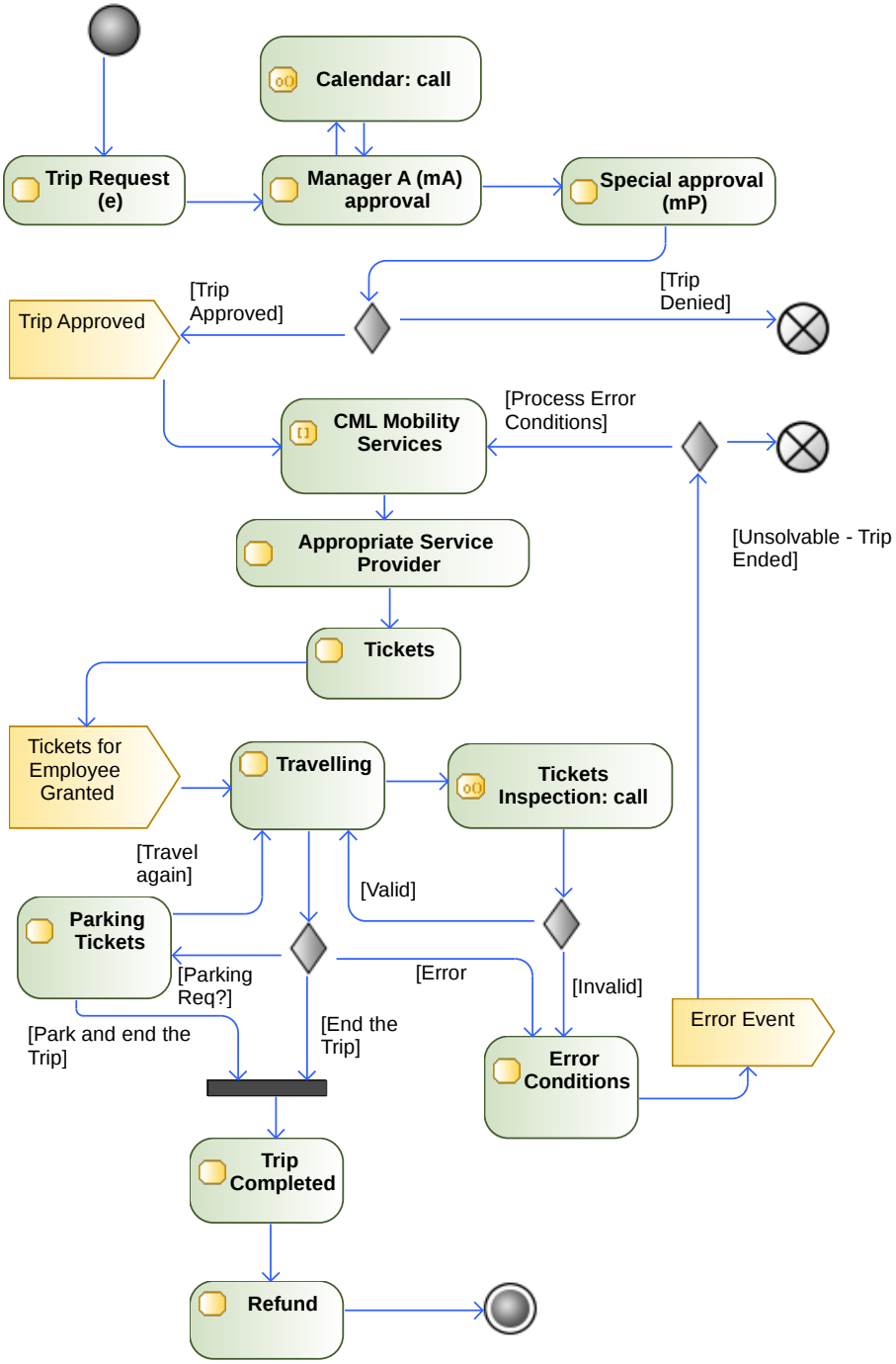



Figure 15.4: SysML activity diagram of the CML business mobility use case

modeling tool "Modelio", for example, can be used to create a SysML activity diagram. Figure 15.4 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

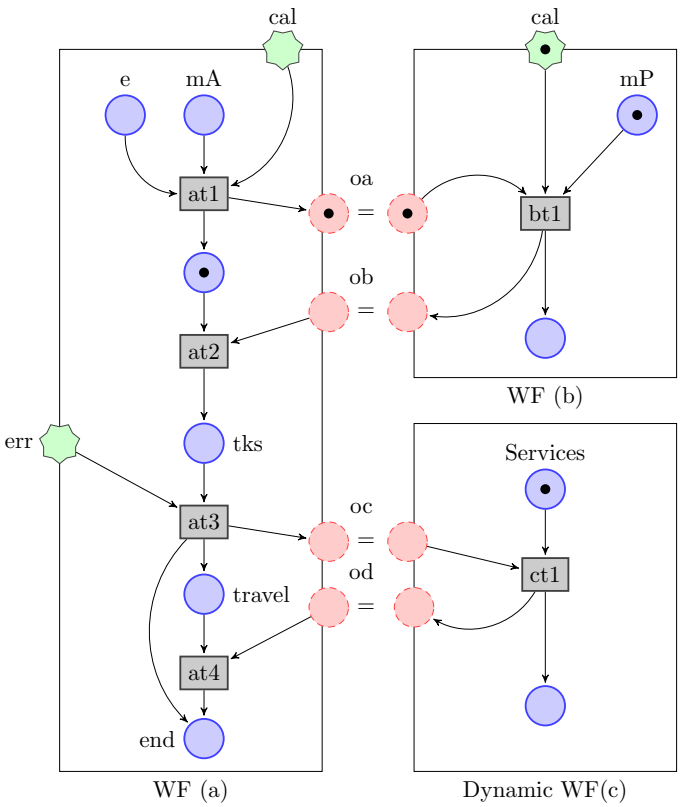



Figure 15.5: Petri Net workflows of the business mobility use case

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. 15.5. 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.
- The WFs (a), (b), and (c) as shown in Fig. 15.5 are the resulting Petri Net workflows created by the workflow experts and are available in the central CML repository or the workflow 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 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. 15.5: 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. 15.5. 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. 15.5, 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. 15.5.
- 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. 15.5.
- Alternatively, if this trip does not 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 the next transition at2 e.g., OAuth token with a secret with which that transition at2 does not need a token from the open place (ob). Therefore, the transition at2 fires only with its normal input place. Similarly, it is possible to execute WF (a) without invoking WFs (b and c). This scenario describes that was no need for a particular 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 particular 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 particular 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 human intervention is invoked. Thus, new tickets are delivered via the open place (od). Note: figure 15.5 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), reimbursements can be automated, i.e., after a successful trip a waiting time is introduced using timeout transition and 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 their employees. The

companies could restrict its employees from using transportation services 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.

## 15.3 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 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. 15.6.

We translate the SysML activity diagram of DriveNow car hire process into our Petri Network workflows as shown in Fig. 15.7.

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).

- Method(a): the customer finds a DriveNow car in the street with Green LED blinking on the 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 minutes 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 minutes. The customer should open the reserved car within 15 minutes; 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. 15.7. The Transition (at1) process this token in place (cus), availability of the car (with inbuilt car information) in place (car) and opens the door.

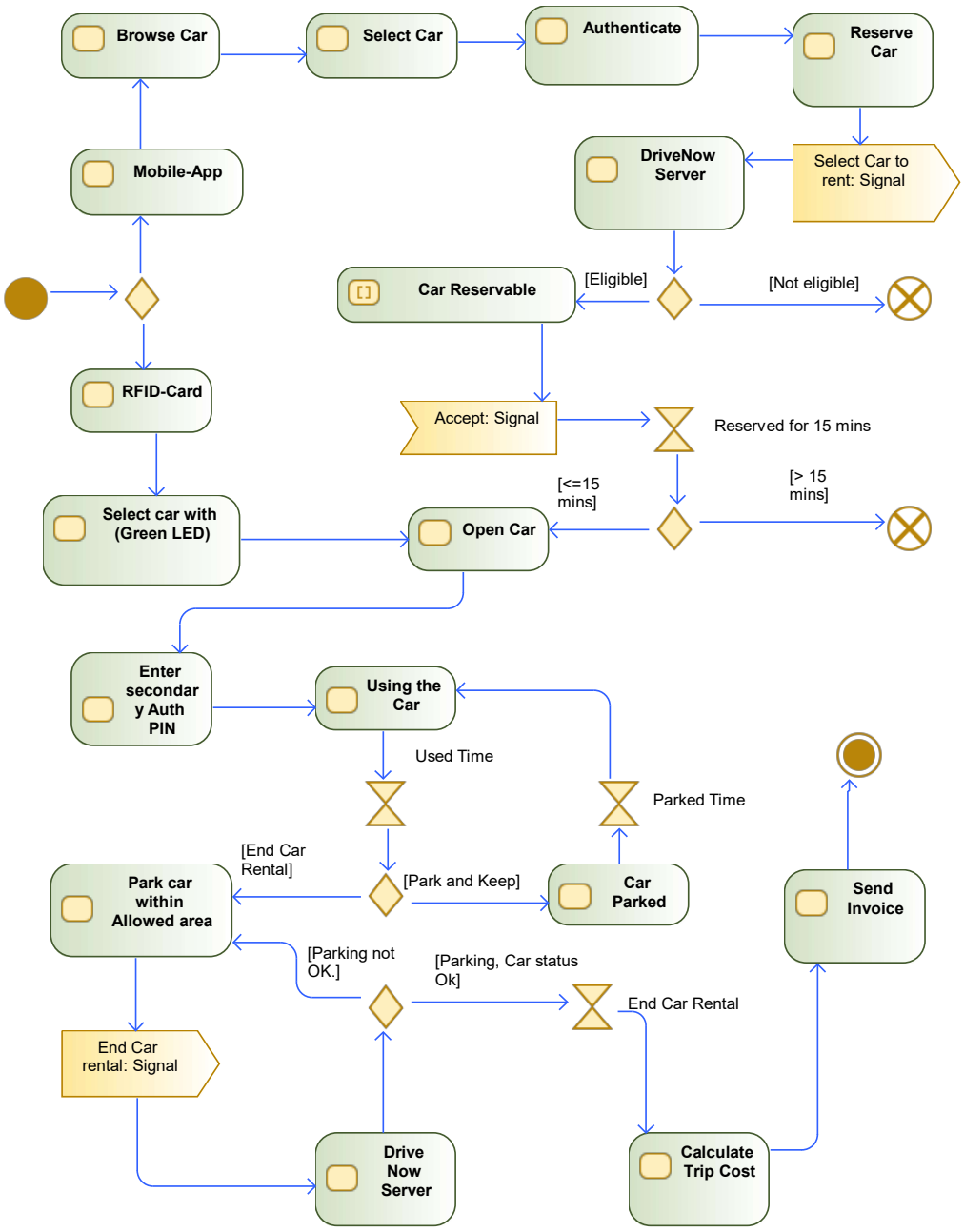



Figure 15.6: SysML Activity Diagram of DriveNow Car sharing platform

- 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

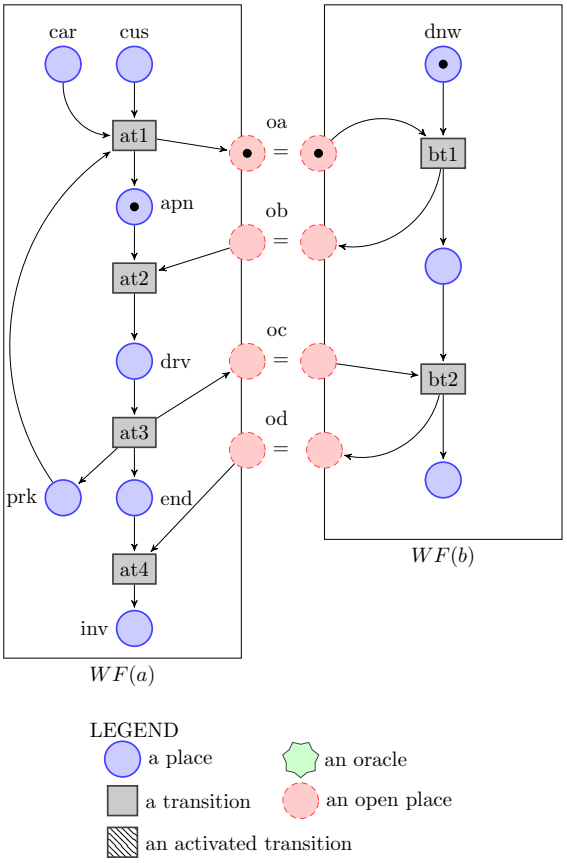



Figure 15.7: Petri Net workflow specification of DriveNow use case

dashboard, i.e., it is not possible to end of 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 (geo-fenced 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 15.7 shows the Petri Net workflow of DriveNow car sharing 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.

## 15.4 Supply Chain

This particular use case is used to evaluate of Petri Nets based Secure Smart Contract Generation Framework. This section presents a supply chain use case and the advantages of using the blockchain approach.

A supply chain is a network of different organizations involved in the process of moving products from one or more suppliers to a customer. The products can be raw materials, small parts, natural resources (e.g., minerals), food, finished products, or even a service. A few examples of involved parties and stakeholders are the manufactures, customers or consumers, logistic companies, distributors, insurance providers, regulatory bodies, and so on.

If the participants of the supply chain are enforced to record essential data on the blockchain, then the following features of blockchain reduce the chances of fraud in a supply chain process. The blockchain technology provides a single source of truth, a transparent auditability of the historical events, and proof of provenance by storing data in an immutable way. The main focus of any supply chain use case is to have an audit trail of products to track provenance. The blockchain ensures a correct and verifiable audit trail throughout the lifetime of the product.

As the use case is dynamic and depends on the domain type of the product, it is important to generate a smart contract that fits the use case. These domain-specific workflows can be modeled by domain specialists without the knowledge of smart contract development. Our proposed smart contract modeling and generation framework is designed with this focus and therefore brings all the advantages described in the section 12.1. As described earlier, after modeling, the developer or smart contract expert reviews the generated smart contract and deploys it to

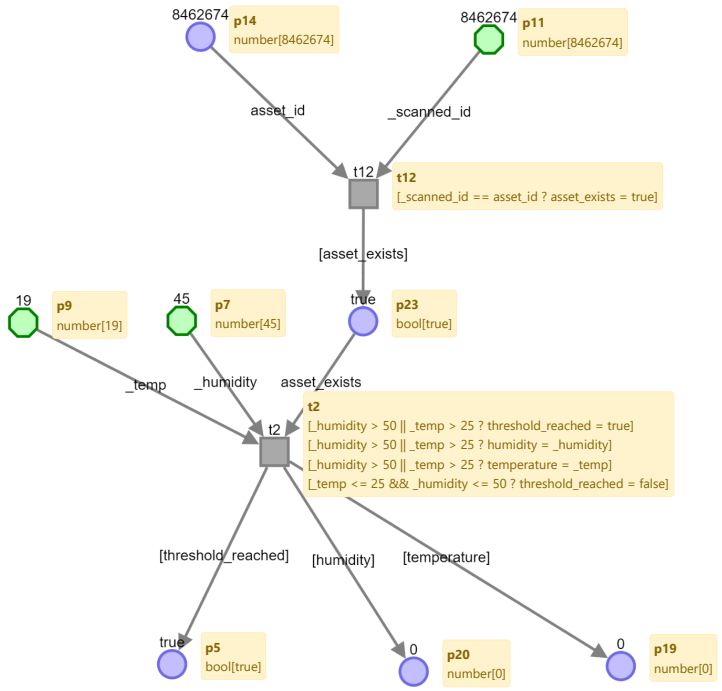



Figure 15.8: An example Petri Net workflow that shows a part of the supply chain use case.

the blockchain network. In this step, before deploying the smart contract code, the smart contract developer needs to perform a requirement analysis of essential data that is required to on the blockchain, and define the scope of the methods (i.e., which methods should be public, private, etc.).

Our approach allows the domain specialists to simulate and test the workflow or business requirements prior to the code generation part, this activity provides the confidence that the smart contract is modeled as required. Our approach can support different blockchain technologies with one standard modeling approach, thus it minimizes the development cost, testing, and verification efforts. In particular, with our approach, we improve security by minimizing the errors in the design stage of smart contract development (correctness-by-design).

For simplicity reasons, we modeled only a part of the supply chain use case with desirable features and actions involved. An example of a part of a supply chain use case is shown in Fig. 15.8. The modeled Petri Net workflow consists of three Oracle places, five normal places, and two transitions. The Oracle places (*p11,p7,p9*) represent some external information that is sent by different equipment and sensors such as a scanner, a temperature sensor, and a humidity sensor. The first transition (*t12*) checks if the scanned asset ID (*_scanned_id*) from the Oracle place (*p11*) equals to the already registered *asset_id* from the place (*p14*). If transition *t12*

conditions evaluate true, then it passes a token (*asset_exists*) to the input place (*p23*) of the second transition (*t2*). The transition (*t2*) has two additional input oracles named *_humidity* and *_temperature* besides the input place (*p23*). The guarded commands of the transaction *t2* checks if the humidity value exceeds the threshold of 50 or the temperature value exceeds 25, if this evaluates true, then the *threshold_reached* boolean variable is set to true and the temperature and humidity variables are set accordingly. If the temperature value is less than or equal to 25 and the humidity value is less than equal to 50 then the *threshold_reached* variable is set to false. In either case, all output places (*p5, p20, p19*) of transition *t2* receives a token.

The Petri Net visual modeling tool allows assigning the tokens to the starting places and oracles, which makes it possible to simulate the workflow with the arbitrary tokens and fire the transactions prior to the smart contract generation step. That way each transition can be tested with different token values to cover the complete condition space set by the guarded commands. The user can fire each transition separately, or fast forwards through the complete Petri Net model and execute all transitions.

The generated solidity code from the PN workflow shown in Fig. 15.8 is presented in the appendix listing A.2. The generated smart contract follows the approach that was explained in the chapter 12.1.4. The code is separated into two types of functions: one is the main *execute* function which represents the execution flow of the transitions; and, the other is the normal function which encapsulates the guarded commands and assignments written in the transitions.

The *execute* function represents the execution flow of the workflow through the conditioned execution of the functions representing transitions. These functions are executed only if all incoming places of each transition are enabled, which means that they contain tokens. Next, the incoming places of the first transition are enabled to start the execution at the beginning of the *execute* function.

The normal functions of the transitions have limited internal scope for the execution - this prevents the functions from being publicly executable, and the functions being called within the contract itself or any derived contracts. The body of the normal function starts with the definition of local variables which are created from input and output places connected to the transition. In the case of input places, the variable of the incoming input token is assigned to the local variable whereas, in case of the outgoing tokens, the variables are initialized to the default value of the output place.

Below, we explain how guarded command conditions written in the transitions are translated into the code. Every translation that holds guarded commands execute the following three steps:

- *Consume*: Each condition first starts by *consuming* the tokens of all addressed places of the guarded command and therefore disabling the place. This is done by assigning the token state variable to *false*.
- *Assign*: Next, the logic of guarded commands is translated into the code, resulting in the list of assignments to the local variables of the tokens.
- *Produce*: The last step is to produce tokens for outgoing places. This process is done by first assigning the token values of the local variables to the global ones followed by assigning the state variables of tokens to *true* and therefore enabling the outgoing places.

## 15.5 Use Case Discussion

A use case and its requirements, entities involved and their relationship with IoT devices and external services can influence what features of the WFAC are necessary, suitable, and how they can be used. In this section, how the use case factors such as IoT device constraints and handheld security influence the WFAC framework and how one can implement the WFAC in a generic distributed IoT applications are presented.

### 15.5.1 IoT devices and their capabilities

Depending on the use case, thee IoT devices used can be either constrained devices or powerful. If the IoT device is constrained the introduced PAT profile and receipt token concepts are necessary, otherwise, if the IoT device is powerful, then only the receipt token concept of the WFAC is necessary.

- Even the considered constrained IoT devices should be able to perform primitive cryptographic operations including hash, encryption, decryption, and token validation. In addition, they can maintain a partial workflow state and pass workflow information to other entities in the form of receipt tokens. For instance, constrained IoT devices may benefit from other powerful devices such as a border router or a secure handheld (e.g., with workflow application) that support to retrieve and publish information to external services. These powerful devices act as a proxy between constrained devices and the external devices, therefore, they must be able to authenticate against each other. Thus, entities that maintain and pass the workflow state to each other must also be secure. To support such constrained IoT devices and to protect the client identity information, the PAT profile is introduced in the WFAC framework.

- A powerful IoT device can support transport layer or application layer encryption, have unlimited power resources, and have direct communication with external services without requiring a proxy. In this situation, introducing the receipt tokens concept from the WFAC is sufficient to prove to other IoT devices or services that the workflow participant has completed the previous workflow task. When the transport layer or application layer encryption is available to protect end-to-end communication, the introduced PAT profile is not necessary. Note: The motivation of the WFAC framework is to support constrained IoT devices with capabilities such as performing basic cryptographic operations as described above and not the devices that cannot perform any cryptographic operations.

### 15.5.2 Handhelds and their capabilities

The handhelds that run the workflow application can be either trusted (i.e., the handheld is developed by the resource owner or certified such that the secrets, tokens, workflow state information cannot be tampered) or not trusted (i.e., the handheld is not trusted and owned by the workflow participant, therefore, the application and information stored can be tampered).

- Handhelds that run the workflow application can be secured by leveraging hardware security approaches such as TEE. TEE guarantees that the cryptographic operations and keys are protected in a secure enclave and cannot be tampered easily. With this assumption, the workflow application may use TEE to protect keys with which it could create short lived access tokens such as OAuth tokens for the workflow participant. This allows the resource owners to model workflows that can work without having continuous online access for retrieving access tokens for authorization servers. Also, the workflow information, i.e., actions committed by the workflow participant and receipt tokens of IoT devices can be processed and stored in the handheld before being sent to other entities. For instance, the workflow application in a handheld could maintain the local workflow execution state of the workflow participant, then after finishing one or more tasks, aggregated information can be published via an oracle. Oracles allow participants to create, read, and update particular resources involved in the workflow. The workflow participants can query the oracle to receive the current state of that workflow information whenever necessary.
- If the handhelds are not trusted (i.e., the workflow info, secret or tokens can be tampered), then the involved IoT devices must be secure and cannot be tampered. In addition, the IoT devices must have the following capabilities: support PAT profile and/or transport layer or application layer encryption, the ability to handle receipt tokens and parse them, communication capabilities to external services through which the receipt tokens and workflow state

information can be published and synchronized locally (i.e., local IoT device to device communication via the secure border router or master device) or globally (e.g., communication with external service such as an oracle through which all IoT device can query and synchronize information).

### 15.5.3 Distributed and decentralized ledger

This thesis focuses on decentralized and distributed IoT applications where the owners may not trust each other. Therefore, a distributed, decentralized, and an immutable ledger such as a blockchain can be used to record workflow events and enable traceability of workflow actions committed by each participant. An oracle can be used to query, publish, and updated information in the blockchain. In a typical blockchain system, transactions consisting of one or more workflow states are proposed by participating entities, and later, the transactions are approved by other participating entities depending on the underlying consensus mechanism. Thus, accountability and auditing can be guaranteed where no entities could trust each other.

# Part V

# Synopsis

# Chapter 16

# Conclusion

Over the recent years, IoT devices are used in many different applications, including the critical infrastructure. Attacks on such IoT devices are increasing and also getting more complex. Therefore, the shortcomings in IoT technologies motivated us to develop security mechanisms to stop adversaries from using IoT devices for malicious purposes. On the other hand, the IoT device owners want to enforce their own rules and conditions on entities that want to access the exposed IoT services. This thesis develops an access control framework that enforces workflows (representing owners' rules and conditions) on entities that want to access distributed resources such as in IoT. The introduced WFAC framework can be integrated with user-friendly and practitioner-friend tools, interoperable IoT protocols, and WFAC reduces the possible ways an attacker can compromise a system or resource protected with WFAC.

This thesis investigates the use cases of IoT devices and applications to identify the requirements and problems that exist in the IoT. The research goals and scope of this thesis are formulated based on the concrete use cases and requirements from the stakeholders. Using the introduced WFAC framework, one can grant access control permissions to entities in a more fine-grained form: "You are allowed to execute this task in this workflow at this moment" instead of "You are authorized to access this service during this period of time (temporal) or because you are an admin (role), or because you possesses certain attributes". The permission to execute a step in a workflow depends on having executed the required previous steps (i.e., based on the history) of the workflow.

The contributions of this thesis are summarized in the following three different areas:

**Workflow specification and enforcement:** This thesis presents a Petri Net based workflow specification and enforcement framework by extending and adapting the Petri Nets to support and secure emergent IoT applications. This thesis demonstrates how the proposed method can solve different use cases and guarantee workflow integrity, where workflows are enforced in a distributed IoT environment. Besides, Petri Nets have simple semantics, PNs are easy to understand and do not

need any prior mathematical background in contrast to other approaches such as automata. The WFAC framework can be integrated with other practitioner-friendly tools such as SysML. The thesis shows how the workflow specified in Petri Nets can manage error situations by exchanging information via open Petri Net places - therefore, this contribution satisfies the following Research Questions (RQ): RQ 1, RQ 2, RQ 4, and RQ 3 for modeling, specification, and enforcement of workflows including error-recovery features in distributed IoT environment.

**IoT workflow services:** This thesis introduces a lightweight workflow service to support the workflow execution and to trace the history of actions committed by the workflow participant in a distributed environment. The IoT device has a workflow service that provides entities with a signed token (also known as the *receipt tokens*) that certifies the workflow action completed by the entity. Thus, receipt token provides an additional level of assurance to the entities that previous workflow action has been completed successfully. The next IoT device or service can verify that the participating entity has completed the relevant workflow action. In addition, this thesis presented how privacy-enhancing tokens (PAT) can be used in situations, where IoT devices might not use communication security protocols such as DTLS. Therefore, this contribution satisfies the following Research Question (RQ): RQ 1 and RQ 2 in a distributed IoT environment.

**Blockchain and smart contracts:** WFAC framework includes a tool that translates the Petri Net workflows into a *Smart Contract* template. The generated smart contract templates can be deployed in a blockchain with little additional effort. Smart contracts can guarantee workflow execution as defined and publish the information of actions committed by the entities on the immutable ledger provided by the blockchain. Thus, one can achieve accountability in an untrusted, distributed, and decentralized environment. In addition, the WFAC allows using an accountability database such as a blockchain, for recording important workflow events via the AS, RS, and WF-App - which could be later used for auditing purposes. Therefore, this contribution satisfies the following Research Question (RQ): RQ 5 and RQ 2, where entities do not have to trust one central trusted entity to ensure workflow integrity and accountability.

To summarize, this thesis presents the WFAC framework for a distributed environment that supports the following:

- To specify workflows as Petri Nets, which are amenable to formal verification that can be created in a stepwise manner with the help of practitioner friendly and user friendly tools such as SysML - satisfies RQ 1 and RQ 3.
- To restrict an entity from using an application/service/resource by enforcing a prescribed workflow. This workflow includes enforces fine-grained authorization constraints based on the *least privilege* and *need to access* principle. The introduced method is applicable in the IoT environment via OAuth tokens and the introduced receipt tokens. In addition, WFAC supports a workflow

application that allows entities participating in a workflow to have a choice, for example, to accept (or reject) "contracts" or conditions and thus, enforces the integrity of Workflow - satisfies RQ 1 and RQ 2.

- A method to handle error conditions by supporting the creation of dynamic workflows - satisfies RQ 4.
- To enable distributed accountability while executing the workflow, i.e., actions executed by entities executing the workflow are recorded in an immutable database. In addition, WFAC includes a framework to generate Petri Net based smart contracts that can be natively deployed on a blockchain platform - satisfies RQ 5.

## 16.1 Discussion

This thesis has contributed to the existing research by introducing the WFAC framework. However, the approach introduced also has some limitations. The limitations of the proposed framework are discussed below:

**Deadlocks when merging workflows**: Consider three small individual processes a, b, and c designed and verified for Petri Nets properties. With the help of open Petri Nets, it is possible to create interfaces between those three different processes a, b, and c. Therefore, the concept of high-level Petri Nets helps to create a main workflow consisting of one or more sub-workflows. When two or more sub-workflows are merged without proper validation, then it is possible to have deadlocks. 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 16.1 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 essential to validate and verify the properties before merging sub-workflows with the main workflow.

**Design flaws and resulting errors in Petri Nets**: A Petri Net analysis engine cannot detect a design problem or a flaw introduced by the process or the use case 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, then this significant design flaw cannot be detected by the Petri Net analysis tools themselves. Therefore, designing workflows using Petri Nets does not guarantee error (e.g., deadlock-free) free workflows. The four-eyes principle can be used to verify such problems in the process using another expert. The errors or problems in the process design can be reduced when it is reviewed by several experts. Once the process is designed without obvious design

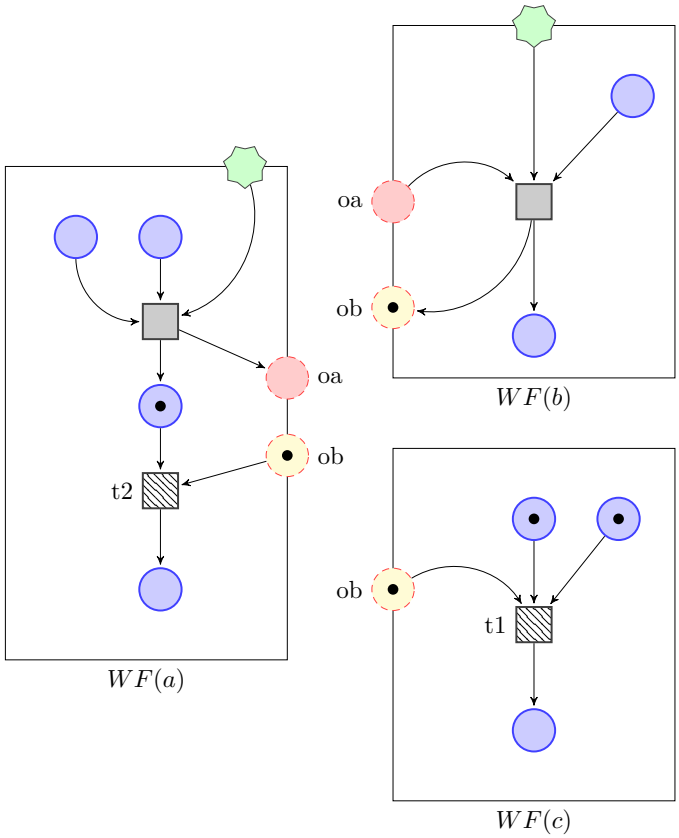



Figure 16.1: The token in open place *ob* of WF (b) can be consumed either by *t2* of WF (a) or *t1* of WF (c). This prevents either WFs (a or c) to proceed forward.

flaws, then it can be evaluated with Petri Nets analysis tools for properties such as deadlock-freeness.

**Battery Exhaustion Problem:** The handhelds and the IoT devices involved in the workflow execution usually rely on batteries for their power supply. For example, a handheld is expected to work for a short period of time without a connection to some external services (e.g., a service that synchronizes the workflow states to an accountability server). In such situations, if the battery-operated handheld loses power, then the workflow state or actions committed that are not persistent in the handheld memory are lost. The battery exhaustion problem is not a limitation to the proposed method but is a common issue of systems that operates on battery power without connectivity to external systems, thus, without the power and persistent data storage, the current status and related actions are lost if they are not synchronized to trusted external entities. If the workflow states are stored temporarily in the handhelds, then appropriate security mechanisms must be deployed to protect the confidentiality and integrity of workflow related secrets and state information.

**Optimization issues in smart contract generation framework**: As the framework was developed to adapt to any blockchain technology, the smart contracts generated are not optimized for a blockchain platform. For example, in Ethereum blockchain, a platform specific value known as *gas* is required to run a smart contract. The gas value has some monetary value and is introduced to protect the abuse of computational resources in a distributed environment. The smart contract code (Ethereum based solidity) presented in the appendix listing A.2 is not optimized to minimize the consumption of gas as this was not a focus of this thesis. The generated smart contract is only a template; appropriate business-logic conditions must be developed by a developer before deploying the smart contract. This process could introduce additional coding language specific errors. This topic is not within the scope of this work. This thesis recommends the integration of smart contract vulnerability analysis tools coupled with manual expert reviews before deploying them on the blockchain.

**Regulatory Compliance with Laws**: The WFAC framework proposes the use of a blockchain platform to record workflow related information transparently (e.g., approvals, actions committed by the workflow participant). Traditionally, when a piece of data is recorded in a blockchain, then because of its inherent properties such as immutability, this data stays in the blockchain forever and cannot be deleted or erased. In contrast, privacy regulations such as GDPR demand the responsible data controllers and data processors to comply with strict rules such as "*right to be forgotten*" - which states that the data owner has the right to inform the data controller to delete all data related to the data owner. Moreover, in many permissionless blockchain architectures, there is no single legal entity (e.g., data controller) who is responsible for complying with such laws. A recent study (see [42]) submitted to the EU Parliament discusses these issues carefully and proposes three different policy recommendations to address this issue. Also, permissioned blockchain platforms such as Hyperledger Fabric are working towards addressing some of these issues e.g., in Hyperledger Fabric v2.0 [1]. This is still an open issue and has to be researched further. This thesis did not focus on this topic; however, it recognizes this as an important issue that should be solved as future work.

## 16.2 Future Work

Although this thesis has presented a WFAC framework, there are few open questions that could be investigated as future work. This section presents potential research directions.

[1] `https://hyperledger-fabric.readthedocs.io/en/release-2.0/private-data/private-data.html`

### 16.2.1 Smart Contract Generation

Current implementation supports only Ethereum based smart contract language, i.e., the Solidity. In the future, the tool can be extended to support other blockchain platforms such as Hyperledger Fabric. Also, the Solidity smart contracts generated from Petri Nets are not optimized. For instance, the Solidity smart contracts may consume gas value based on the resources used within the smart contract. The gas units contain a certain monetary value in the real blockchain network. Therefore, Smart Contracts must be optimized to consume less gas before deploying them. One of the approaches to minimize gas consumption can be further work by extending the introduced smart contract generation module. For example, one can use bit arrays to store state variables of tokens instead of separate variables for each token. For example, Garcia et al., in [135] explains a type of optimization concept with its advantages and limitations.

### 16.2.2 EU Funded Research Projects

As a continuation of this thesis, to further develop the research concepts and to address some of the open issues discussed earlier, the author of this thesis is involved in two EU funded H2020 research projects: (a) Cyber Security for Europe (CyberSec4Europe [2]) and A COmprehensive cyber-intelligence framework for resilient coLLABorative manufacturing Systems (COLLABS [3]). In particular, the aforementioned projects focus on industrial scenarios involving supply chain and manufacturing use cases. The topics include investigation of what information must be stored in the blockchain (on-chain or off-chain) to provide the following properties: transparency, auditability, and traceability while protecting the privacy and confidential information of the involved stakeholders. One of the main focus involves the specification of the information model, the data structure of the workflow events published as transactions. This includes information stored on-chain (i.e., the blockchain) and off-chain (e.g., a private local database storing confidential information and publishing a hash of the data object on the main blockchain).

### 16.2.3 Local Reasoner

This thesis has introduced a lightweight IoT workflow service as part of the IoT devices to evaluate whether the workflow participant has performed the corresponding workflow specific action or not. The IoT device should be aware of the workflow and its tasks or actions that must be conducted by the workflow participant to

[2] https://cybersec4europe.eu/
[3] https://www.collabs-project.eu/

access its service. In existing implementation, the rules are static. As future work, to enable dynamic rules, it is crucial to have a rule processing engine.

A local reasoner is an envisioned rule processing component within an IoT device that can reason from the information that is passed to the device and other existing *Trust* rules. As a result of the evaluation, the reasoner should be able to either grant or deny access to the resources for the workflow participant.

# Bibliography


[1] Butler W. Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Trans. Comput. Syst.*, 10(4):265–310, 1992.

[2] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[3] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig. Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth). Technical report, IETF, 2018.

[4] W. Trappe, R. Howard, and R. S. Moore. Low-energy security: Limits and opportunities in the internet of things. *IEEE Security Privacy*, 13(1):14–21, Jan 2015.

[5] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao. A survey on security and privacy issues in internet-of-things. *IEEE Internet of Things Journal*, 4(5):1250–1258, Oct 2017.

[6] ENISA. Baseline security recommendations for iot in the context of critical information infrastructures. Technical report, European Union Agency For Network And Information Security (ENISA), November 2017.

[7] I. Stellios, P. Kotzanikolaou, M. Psarakis, C. Alcaraz, and J. Lopez. A survey of iot-enabled cyberattacks: Assessing attack paths to critical infrastructures and services. *IEEE Communications Surveys Tutorials*, 20(4):3453–3495, Fourthquarter 2018.

[8] Manos Antonakakis, Tim April, Michael Bailey, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, Yi Zhou, Manos Antonakakis Tim April, Matthew Bernhard Elie Bursztein, Jaime J Cochran Zakir Durumeric Alex Halderman Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever Zane Ma, Joshua Mason, and Nick Sullivan Kurt Thomas. Understanding the Mirai Botnet. In *USENIX Security Symposium*, pages 1092–1110. 26th USENIX Security Symposium, 2017.

[9] A. Sadeghi, C. Wachsmann, and M. Waidner. Security and privacy challenges in industrial internet of things. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2015.

[10] Elaine Barker, Miles Smid, and Dennis Branstad. A profile for u.s. federal cryptographic key management systems. *NIST Special Publication*, 800(152), 2015.

[11] Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley Professional, 2018.

[12] Kenneth J Biba. Integrity considerations for secure computer systems. Technical report, MITRE CORP BEDFORD MA, 1977.

[13] David D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 27-29, 1987*, pages 184–195, 1987.

[14] D Elliott Bell and Leonard J La Padula. Secure computer system: Unified exposition and multics interpretation. Technical report, MITRE CORP BEDFORD MA, 1976.

[15] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, February 2000.

[16] David A. Basin, Vincent Jugé, Felix Klaedtke, and Eugen Zalinescu. Enforceable security policies revisited. In *Principles of Security and Trust - First International Conference, POST 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012, Proceedings*, pages 309–328, 2012.

[17] Richard Gay, Heiko Mantel, and Barbara Sprick. Service automata. In *Formal Aspects of Security and Trust - 8th International Workshop, FAST 2011, Leuven, Belgium, September 12-14, 2011. Revised Selected Papers*, pages 148–163, 2011.

[18] David A. Basin, Samuel J. Burri, and Günter Karjoth. Obstruction-free authorization enforcement: Aligning security and business objectives. *Journal of Computer Security*, 22(5):661–698, 2014.

[19] Lakshya Tandon, Philip W. L. Fong, and Reihaneh Safavi-Naini. HCAP: A history-based capability system for iot devices. In *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies, SACMAT 2018, Indianapolis, IN, USA, June 13-15, 2018*, pages 247–258, 2018.

[20] Alessandro Aldini and Marco Bernardo. On the usability of process algebra: An architectural view. *Theoretical Computer Science*, 335(2):281 – 329, 2005.

[21] Vijayalakshmi Atluri and Wei-kuang Huang. A petri net based safety analysis of workflow authorization models. *Journal of Computer Security*, 8(2/3):209–240, 2000.

[22] Elisa Bertino, Elena Ferrari, and Vijay Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1):65–104, 1999.

[23] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. Blockchain based access control. In Lydia Y. Chen and Hans P. Reiser, editors, *Distributed Applications and Interoperable Systems*, pages 206–220, Cham, 2017. Springer International Publishing.

[24] Yuanyu Zhang, Shoji Kasahara, Yulong Shen, Xiaohong Jiang, and Jianxiong Wan. Smart contract-based access control for the internet of things. *IEEE Internet of Things Journal*, 6(2):1594–1605, 2018.

[25] Oscar Novo. Blockchain meets iot: An architecture for scalable access management in iot. *IEEE Internet of Things Journal*, 5(2):1184–1195, 2018.

[26] Prabhakaran Kasinathan and Jorge Cuellar. Securing the integrity of workflows in iot. In *Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks, EWSN 2018. Madrid, Spain, February 14-16, 2018*, pages 252–257, 2018.

[27] Prabhakaran Kasinathan and Jorge Cuellar. Workflow-aware security of integrated mobility services. In *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II*, pages 3–19, 2018.

[28] Prabhakaran Kasinathan and Jorge Cuellar. Securing emergent iot applications. In *Engineering Trustworthy Software Systems: 4th International School, SETSS 2018, Chongqing, China, April 7–12, 2018, Tutorial Lectures*, pages 99–147, Cham, 2019. Springer International Publishing.

[29] Nejc Zupan, Prabhakaran Kasinathan, Jorge Cuellar, and Markus Sauer. Secure smart contract generation based on petri nets. In *Blockchain Technologies for Industry 4.0*, Cham, 2019. In Press: Springer Singapore Publishing.

[30] Jorge Cuellar, Prabhakaran Kasinathan, and Daniel Calvo. Privacy-Enhanced-Tokens (PAT) profile for ACE. Internet-Draft draft-cuellar-ace-pat-priv-enhanced-authz-tokens-06, Internet Engineering Task Force, January 2018. Work in Progress.

[31] W. M P van der Aalst. Verification of workflow nets. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1248, pages 407–426. Springer, Berlin, Heidelberg, 1997.

[32] Shoichi Morimoto. A Survey of Formal Verification for Business Process Modeling. In *International Conference on Computational Science*, pages 514–522. Springer, Berlin, Heidelberg, 2008.

[33] W. M. P. van der Aalst. The application of Petri nets to workflow management. *Journal of Circuits, Systems and Computers*, 08(01):21–66, feb 1998.

[34] Wolfgang Reisig. *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013.

[35] W. M. P. van der Aalst. *Three Good Reasons for Using a Petri-Net-Based Workflow Management System*, pages 161–182. Springer US, Boston, MA, 1998.

[36] Hejiao Huang and Hélène Kirchner. Formal specification and verification of modular security policy based on colored petri nets. *IEEE Trans. Dependable Sec. Comput.*, 8(6):852–865, 2011.

[37] Óscar R. Ribeiro and João M. Fernandes. Translating synchronous petri nets into PROMELA for verifying behavioural properties. In *IEEE Second International Symposium on Industrial Embedded Systems, SIES 2007, Hotel Costa da Caparica, Lisbon, Portugal, July 4-6, 2007*, pages 266–273, 2007.

[38] A. Calà, A. Lüder, J. Vollmar, and M. Foehr. Evaluation of migration scenarios towards cyber-physical production systems using sysml. In *2017 IEEE International Systems Engineering Symposium (ISSE)*, pages 1–5, Oct 2017.

[39] Birgit Vogel-Heuser. Usability experiments to evaluate uml/sysml-based model driven software engineering notations for logic control in manufacturing automation. *Journal of Software Engineering and Applications*, 7(11):943, 2014.

[40] Daniel Schütz, Martin Obermeier, and Birgit Vogel-Heuser. Sysml-based approach for automation software development - explorative usability evaluation of the provided notation. In *Design, User Experience, and Usability. Web, Mobile, and Product Design - Second International Conference, DUXU 2013, Held as Part of HCI International 2013, Las Vegas, NV, USA, July 21-26, 2013, Proceedings, Part IV*, pages 568–574, 2013.

[41] Carsten Bormann, Mehmet Ersue, and Ari Keranen. Terminology for Constrained-Node Networks. Technical report, IETF, may 2014.

[42] Michèle Finck. Blockchain and the general data protection regulation. Panel for the Future of Science and Technology - Study PE 634.445, European Parliamentary Research Service Scientific Foresight Unit (STOA), July 2019.

[43] Ahmad-Reza Sadeghi, Christian Wachsmann, and Michael Waidner. Security and privacy challenges in industrial internet of things. In *Proceedings of the 52nd Annual Design Automation Conference on - DAC '15*, pages 1–6, New York, New York, USA, 2015. ACM Press.

[44] Rolf H. Weber. Internet of Things – New security and privacy challenges. *Computer Law & Security Review*, 26(1):23–30, jan 2010.

[45] S. Sicari, A. Rizzardi, L.A. Grieco, and A. Coen-Porisini. Security, privacy and trust in Internet of Things: The road ahead. *Computer Networks*, 76:146–164, jan 2015.

[46] Daniel Miessler, Craig Smith, and Jason Haddix. OWASP Internet of Things Top Ten Project. Accessed on December 2017, 2014.

[47] IETF ACE Working Group. Authentication and Authorization for Constrained Environments (ACE). Accessed on December 2017, 2017.

[48] Dick Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, October 2012.

[49] S. Gerdes, O. Bergmann, C. Bormann, G. Selander, and L. Seitz. Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE). Accessed on March 2018, 2018.

[50] R.R. Schaller. Moore's law: past, present and future. *IEEE Spectrum*, 34(6):52–59, jun 1997.

[51] Ravi S. Sandhu and Pierangela Samarati. Access Control: Principles and Practice. *IEEE Communications Magazine*, 32(9):40–48, sep 1994.

[52] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal methods: Practice and experience. *ACM Computing Surveys*, 41(4):1–36, oct 2009.

[53] Robert W. Shirey. Internet Security Glossary, Version 2. RFC 4949, August 2007.

[54] Giampaolo Bella and Lawrence C. Paulson. Accountability protocols: Formalized and verified. *ACM Trans. Inf. Syst. Secur.*, 9(2):138–161, May 2006.

[55] P. Malone and B. Jennings. Distributed accountability model for digital ecosystems. In *2008 2nd IEEE International Conference on Digital Ecosystems and Technologies*, pages 452–460, Feb 2008.

[56] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: Definition and relationship to verifiability. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, page 526–535, New York, NY, USA, 2010. Association for Computing Machinery.

[57] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and cryptocurrency technologies: A comprehensive introduction*. Princeton University Press, 2016.

[58] Organizationy AIOTI. The Alliance for the Internet of Things Innovation. Accessed on December 2018, 2018.

[59] K Pohl. *Requirements engineering: An overview*. RWTH, Fachgruppe Informatik, 1996.

[60] Dieter Gollmann. *Computer Security*. Wiley, 2011.

[61] Richard Kuhn, Vincent Hu, W Polk, and Shu-jen Chang. Introduction to public key technology and the federal pki infrastructure. *NIST special publication*, 800(32), 2001.

[62] Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to attribute based access control (abac) definition and considerations. *NIST special publication*, 800(162), 2014.

[63] Donald C Latham. Department of trusted computer system evaluation criteria. *Department of Defense*, 1986.

[64] D. F. C. Brewer and Michael J. Nash. The chinese wall security policy. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 1-3, 1989*, pages 206–214, 1989.

[65] NIST Joint Task Force Transformation Initiative. Security and privacy controls for federal information systems and organizations. *NIST special publication*, 800(53), 2013.

[66] Peter J. Denning. Third generation computer systems. *ACM Comput. Surv.*, 3(4):175–216, 1971.

[67] G. Scott Graham and Peter J. Denning. Protection: principles and practice. In *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1972 Spring Joint Computer Conference, Atlantic City, NJ, USA, May 16-18, 1972*, pages 417–429, 1972.

[68] Butler W. Lampson. Protection. *Operating Systems Review*, 8(1):18–24, 1974.

[69] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, Feb 1996.

[70] R. K. Thomas and R. S. Sandhu. *Task-based authorization controls (TBAC): a family of models for active and enterprise-oriented authorization management*, pages 166–181. Springer US, Boston, MA, 1998.

[71] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas. Attribute-based access control. *Computer*, 48(2):85–88, Feb 2015.

[72] OASIS. eXtensible Access Control Markup Language (XACML),Version 3.0. Accessed on Augst 2019, 2013.

[73] Dominick Baier, Vittorio Bertocci, Keith Brown, Matias Woloski, and Eugenio Pace. *A Guide to Claims-Based Identity and Access Control: Patterns & Practices*. Microsoft Press, Redmond, WA, USA, 1st edition, 2010.

[74] Javier Lopez and Juan E. Rubio. Access control for cyber-physical systems interconnected to the cloud. *Computer Networks*, 134:46 – 54, 2018.

[75] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3(2):125–143, 1977.

[76] Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: enforcement mechanisms for run-time security policies. *Int. J. Inf. Sec.*, 4(1-2):2–16, 2005.

[77] David Spence, George Gross, Cees de Laat, Stephen Farrell, Leon HM Gommans, Pat R. Calhoun, Matt Holdrege, Betty W. de Bruijn, and John Vollbrecht. AAA Authorization Framework. RFC 2904, August 2000.

[78] Carl Adam Petri. Communication with automata, 1966.

[79] Tadao Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, apr 1989.

[80] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.

[81] Javier Esparza. Decidability and complexity of Petri net problems—an introduction. In *Lectures on Petri Nets I: Basic Models*, page 55. Springer, Berlin, Heidelberg, 1998.

[82] Philip M. Merlin and David J. Farber. Recoverability of Communication Protocols—Implications of a Theoretical Study. *IEEE Transactions on Communications*, 1976.

[83] Kurt Jensen. Coloured Petri nets. In *Petri Nets: Central Models and Their Properties*, pages 248–299. Springer Berlin Heidelberg, Berlin, Heidelberg, 1987.

[84] Kurt Jensen. Coloured petri nets: A high level language for system design and analysis. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1990*, pages 342–416, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[85] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured petri nets and CPN tools for modelling and validation of concurrent systems. *STTT*, 9(3-4):213–254, 2007.

[86] W. M. P. van der Aalst. Putting high-level Petri nets to work in industry. *Computers in Industry*, 25(1):45–54, 1994.

[87] Kurt Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use - Volume 1, Second Edition*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 1996.

[88] Reiko Heckel. Open Petri Nets as Semantic Model for Workflow Integration. In *Petri Net Technology for Communication-Based Systems*, pages 281–294. Springer, Berlin, Heidelberg, 2003.

[89] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*, pages 332–344. IEEE Computer Society, 1986.

[90] Edmund M. Clarke. The birth of model checking. In Orna Grumberg and Helmut Veith, editors, *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2008.

[91] Rob J. van Glabbeek. Notes on the methodology of CCS and CSP. *Theor. Comput. Sci.*, 177(2):329–349, 1997.

[92] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[93] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, Berlin, Heidelberg, 1982.

[94] T. Basten. *In terms of nets : system design with Petri nets and process algebra*. PhD thesis, Department of Mathematics and Computer Science, 1998.

[95] Object Management Group. Business process model and notation version 2.0. Accessed on July 2019, December 2011.

[96] Dominik Bork, Dimitris Karagiannis, and Benedikt Pittl. A survey of modeling language specification techniques. *Information Systems*, 87:101425, 2020.

[97] Aguilar-Savén Ruth Sara. Business process modelling: Review and framework. *International Journal of Production Economics*, 90(2):129 – 149, 2004. Production Planning and Control.

[98] Sonja Meyer, Andreas Ruppen, and Carsten Magerkurth. Internet of things-aware process modeling: Integrating iot devices as business process resources. In Camille Salinesi, Moira C. Norrie, and Óscar Pastor, editors, *Advanced Information Systems Engineering*, pages 84–98, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[99] Harald Sundmaeker, Patrick Guillemin, Peter Friess, and Sylvie Woelfflé, editors. *Vision and Challenges for Realising the Internet of Things*. Publications Office of the European Union, Luxembourg, 2010.

[100] Claude Castelluccia, Aurélien Francillon, Daniele Perito, and Claudio Soriente. On the difficulty of software-based attestation of embedded devices. In *Proceedings of the 16th ACM conference on Computer and communications security - CCS '09*, page 400, New York, New York, USA, 2009. ACM Press.

[101] Working Group TCG. TCG Guidance for Securing Resource-Constrained Devices. Technical report, Trusted Computing Group (TCG), 2017.

[102] European Union (EU). EU GDPR Information Portal. Accessed on July 2018, 2018.

[103] Swarup Mohalik and R. Ramanujam. Assumption-commitment in automata. In S. Ramesh and G. Sivakumar, editors, *Foundations of Software Technology and Theoretical Computer Science*, pages 153–168, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[104] Manfred Broy. A functional rephrasing of the assumption/commitment specification style. *Formal Methods in System Design*, 13(1):87–119, May 1998.

[105] S Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Accessed on October 2018, 2008.

[106] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, Jan 1991.

[107] Melanie Swan. *Blockchain: Blueprint for a new economy.* " O'Reilly Media, Inc.", 2015.

[108] Nick Szabo. Smart contracts: building blocks for digital markets, 1996. *EXTROPY: The Journal of Transhumanist Thought*, 2001.

[109] Ethereum. What Are Smart Contracts - EthereumWiki. Accessed on March 2018, 2018.

[110] Vikram Dhillon, David Metcalf, and Max Hooper. *The DAO Hacked*, pages 67–78. Apress, Berkeley, CA, 2017.

[111] Ethereum. Solidity — Solidity. Accessed on August 2018, 2018.

[112] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making Smart Contracts Smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, pages 254–269, New York, New York, USA, 2016. ACM Press.

[113] Kevin Delmolino, Mitchell Arnett, Ahmed E Kosba, Andrew Miller, and Elaine Shi. Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab. *IACR Cryptology ePrint Archive*, page 460, 2015.

[114] Anastasia Mavridou and Aron Laszka. Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach, 2017.

[115] Maarten Decat, Bert Lagaisse, and Wouter Joosen. Scalable and secure concurrent evaluation of history-based access control policies. In *Proceedings of the 31st Annual Computer Security Applications Conference*, ACSAC 2015, page 281–290, New York, NY, USA, 2015. Association for Computing Machinery.

[116] Thang Bui, Scott D. Stoller, and Shikhar Sharma. Fast distributed evaluation of stateful attribute-based access control policies. In Giovanni Livraga and Sencun Zhu, editors, *Data and Applications Security and Privacy XXXI*, pages 101–119, Cham, 2017. Springer International Publishing.

[117] Daniel Servos and Sylvia L. Osborn. Current research and open problems in attribute-based access control. *ACM Comput. Surv.*, 49(4), January 2017.

[118] Konstantin Knorr. Dynamic access control through petri net workflows. In *16th Annual Computer Security Applications Conference (ACSAC 2000), 11-15 December 2000, New Orleans, Louisiana, USA*, pages 159–167, 2000.

[119] David Basin, Samuel J. Burri, and Günter Karjoth. Optimal workflow-aware authorizations. *ACM Symposium on Access Control Models and Technologies (SACMAT )*, pages 93–102, 2012.

[120] Vijayalakshmi Atluri and Wei-kuang Huang. An authorization model for workflows. In *Computer Security - ESORICS 96, 4th European Symposium on Research in Computer Security, Rome, Italy, September 25-27, 1996, Proceedings*, pages 44–64, 1996.

[121] Wei-Kuang Huang and Vijayalakshmi Atluri. SecureFlow: A Secure Web-enabled Workflow Management System. *Proceedings of the fourth ACM workshop on Role-based access control - RBAC '99*, pages 83–94, 1999.

[122] Luca Compagna, Daniel Ricardo dos Santos, Serena Elisa Ponta, and Silvio Ranise. Aegis: Automatic Enforcement of Security Policies in Workflow-drivenWeb Applications. *Proceedings of ACM on Conference on Data and Application Security and Privacy - CODASPY '17*, pages 321–328, 2017.

[123] Kjeld H. Mortensen. Automatic code generation method based on coloured petri net models applied on an access control system. In Mogens Nielsen and Dan Simpson, editors, *Application and Theory of Petri Nets 2000*, pages 367–386, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[124] Torsten Lodderstedt, David Basin, and Jürgen Doser. SecureUML: A UML-based modeling language for model-driven security. In *Proceedings of UML 2002*, pages 426–441. Springer, Berlin, Heidelberg, 2002.

[125] Jan Jürjens. UMLsec: Extending UML for Secure Systems Development. In *International Conference on the Unified Modeling Language*, pages 412–425. Springer, Berlin, Heidelberg, 2002.

[126] Marcos Vinicius Linhares, Alexandre Jose da Silva, and Romulo Silva de Oliveira. Empirical Evaluation of SysML through the Modeling of an Industrial Automation Unit. In *2006 IEEE Conference on Emerging Technologies and Factory Automation*, pages 145–152. IEEE, sep 2006.

[127] Modelio - Open Source Tool. Modelio - the open source modeling tool. Accessed on August 2018, 2018.

[128] Christian Wolter, Michael Menzel, Andreas Schaad, Philip Miseldine, and Christoph Meinel. Model-driven business process security requirement specification. *Journal of Systems Architecture*, 55(4):211–223, apr 2009.

[129] Sergio Gusmeroli, Salvatore Piccione, and Domenico Rotondi. Iot access control issues: a capability based approach. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 787–792. IEEE, 2012.

[130] Bayu Anggorojati, Parikshit Narendra Mahalle, Neeli Rashmi Prasad, and Ramjee Prasad. Capability-based access control delegation model on the federated iot network. In *The 15th International Symposium on Wireless Personal Multimedia Communications*, pages 604–608. IEEE, 2012.

[131] Sara Rouhani and Ralph Deters. Blockchain based access control systems: State of the art and challenges. In *IEEE/WIC/ACM International Conference on Web Intelligence*, WI '19, page 423–428, New York, NY, USA, 2019. Association for Computing Machinery.

[132] S. Ding, J. Cao, C. Li, K. Fan, and H. Li. A novel attribute-based access control scheme using blockchain for iot. *IEEE Access*, 7:38431–38441, 2019.

[133] Otto Julio Ahlert Pinno, Andre Ricardo Abed Gregio, and Luis CE De Bona. Controlchain: Blockchain as a central enabler for access control authorizations in the iot. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–6. IEEE, 2017.

[134] David Basin, Ernst-Ruediger Olderog, and Paul E. Sevinc. Specifying and analyzing security automata using csp-oz. In *Proceedings of the 2Nd ACM Symposium on Information, Computer and Communications Security*, ASIACCS '07, pages 70–81, New York, NY, USA, 2007. ACM.

[135] Luciano García-Bañuelos, Alexander Ponomarev, Marlon Dumas, and Ingo Weber. Optimized execution of business processes on blockchain. In *Business Process Management - 15th International Conference, BPM 2017, Barcelona, Spain, September 10-15, 2017, Proceedings*, pages 130–146, 2017.

[136] B. Kiepuszewski, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Fundamentals of control flow in workflows. *Acta Informatica*, 39(3):143–209, Mar 2003.

[137] Hiroaki Nakamura, Kohtaroh Miyamoto, and Michiharu Kudo. Inter-organizational business processes managed by blockchain. In Hakim Hacid, Wojciech Cellary, Hua Wang, Hye-Young Paik, and Rui Zhou, editors, *Web Information Systems Engineering – WISE 2018*, pages 3–17, Cham, 2018. Springer International Publishing.

[138] Anastasia Mavridou and Aron Laszka. Tool demonstration: Fsolidm for designing secure ethereum smart contracts. In *Principles of Security and Trust - 7th International Conference, POST 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, pages 270–277, 2018.

[139] Anastasia Mavridou, Aron Laszka, Emmanouela Stachtiari, and Abhishek Dubey. Verisolid: Correct-by-design smart contracts for ethereum. *CoRR*, abs/1901.01292, 2019.

[140] Stephan Philippi. Automatic code generation from high-level petri-nets for model driven systems engineering. *Journal of Systems and Software*, 79(10):1444 – 1455, 2006. Architecting Dependable Systems.

[141] Automatic code generation from high-level Petri-Netsfor model driven systems engineering. A Petri Nets Model for Blockchain Analysis. *The Computer Journal*, 61(9):1374–1388, 01 2018.

[142] Olivia Choudhury, Nolan Rudolph, Issa Sylla, Noor Fairoza, and Amar Das. Auto-generation of smart contracts from domain-specific ontologies and semantic rules. In *IEEE Blockchain Conference*, volume 2018, 2018.

[143] T. Tateishi, S. Yoshihama, N. Sato, and S. Saito. Automatic smart contract generation using controlled natural language and template. *IBM Journal of Research and Development*, pages 1–1, 2019.

[144] Elaine Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, and Richard Davis. Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography. *NIST special publication*, 800(56A-Rev3), 2016.

[145] Morrie Gasser, Andy Goldstein, Charlie Kaufman, and Butler Lampson. The digital distributed system security architecture. In *Proceedings of the 12th National Computer Security Conference*, pages 305–319, 1989.

[146] Ron Ross, Michael Mcevilley, and Janet Carrier Oren. Systems security engineering. *NIST special publication*, 800(160), 2016.

[147] W. M. P. van der Aalst, K. M. van Hee, A. H. M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, 23(3):333–363, May 2011.

[148] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town Crier: An Authenticated Data Feed for Smart Contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 270–282, New York, NY, USA, 2016. ACM.

[149] Massimo Bartoletti and Livio Pompianu. An Empirical Analysis of Smart Contracts: Platforms, Applications, and Design Patterns. In *International Conference on Financial Cryptography and Data Security*, pages 494–509. Springer, Cham, apr 2017.

[150] Edsger W. Dijkstra and Edsger W. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, aug 1975.

[151] Jim Blythe, Vijay Kothari, Sean Smith, and Ross Koppel. Work in progress: Usable security vs. workflow realities. *Workshop on Usable Security (USEC 2018)*, 01 2018.

[152] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML*. Morgan Kaufmann, 3 edition, 2008.

[153] The Official OMG SysML site. What Is OMG SysML? Accessed on April 2018, 2012.

[154] Wolfgang Reisig. *A primer in Petri net design*. Springer Compass International. Springer, 1992.

[155] Maryam Jamal and Nazir Ahmad Zafar. Transformation of Activity Diagram into Coloured Petri Nets Using Weighted Directed Graph. In *2016 International Conference on Frontiers of Information Technology (FIT)*, pages 181–186. IEEE, dec 2016.

[156] Messaoud Rahim, Malika Boukala-Ioualalen, and Ahmed Hammad. 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, June 23-24, 2014.*, pages 233–248, 2014.

[157] E. Andrade, P. Maciel, G. Callou, and B. Nogueira. A methodology for mapping sysml activity diagram to time petri net for requirement validation of embedded real-time systems with energy constraints. In *2009 Third International Conference on Digital Society*, pages 266–271, Feb 2009.

[158] H. Wang, D. Zhong, T. Zhao, and F. Ren. Integrating model checking with sysml in complex system safety analysis. *IEEE Access*, 7:16561–16571, 2019.

[159] RERUM. Rerum: Reliable, resilient and secure iot for smart city applications. Technical report, European Union's Seventh Programme for research, technological development and demonstration under grant agreement no609094., June 2016.

[160] IBM. Energy-Blockchain Labs and IBM Create Carbon Credit Management Platform Using Hyperledger Fabric on the IBM Cloud. *IBM Press Release*, pages 2–3, 2017.

[161] Thomas Freytag and Martin Sänger. Woped-an educational tool for workflow nets. In *BPM (Demos)*, page 31, 2014.

[162] W. M. P. van der Aalst and Arthur HM Ter Hofstede. Yawl: yet another workflow language. *Information systems*, 30(4):245–275, 2005.

[163] Christian Wolter and Andreas Schaad. Modeling of task-based authorization constraints in bpmn. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management*, pages 64–79, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[164] Christian Wolter, Andreas Schaad, and Christoph Meinel. Task-based entailment constraints for basic workflow patterns. In *Proceedings of the 13th ACM symposium on Access control models and technologies - SACMAT '08*, page 51, New York, New York, USA, 2008. ACM Press.

[165] Ludwig Seitz, Stefanie Gerdes, Göran Selander, Mehdi Mani, and Sandeep Kumar. Use Cases for Authentication and Authorization in Constrained Environments. RFC 7744, January 2016.

[166] Franck Pommereau. SNAKES: A flexible high-level petri nets library (tool paper). In *Application and Theory of Petri Nets and Concurrency - 36th International Conference, PETRI NETS 2015, Brussels, Belgium, June 21-26, 2015, Proceedings*, pages 254–265, 2015.

[167] Michael Weber and Ekkart Kindler. The Petri Net Markup Language. In *Petri Net Technology for Communication-Based Systems*, pages 124–144. Springer, Berlin, Heidelberg, 2003.

[168] Michael Jones, John Bradley, and Hannes Tschofenig. Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs). RFC 7800, April 2016.

[169] Microsoft Corporation. Microsoft threat modeling tool. https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool, 2019. Accessed: 2019-10-11.

[170] Microsoft Corporation. The security development lifecycle (sdl). https://www.microsoft.com/en-us/securityengineering/sdl, 2019. Accessed: 2019-10-11.

[171] Microsoft Corporation. Microsoft threat modeling. https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling, 2019. Accessed: 2019-10-11.

[172] Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam Shostack. Uncover security design flaws using the stride approach. *MSDN Magazine-Louisville*, pages 68–75, 2006.

[173] Microsoft Corporation. Stride chart. https://www.microsoft.com/security/blog/2007/09/11/str chart/, 2007. Accessed: 2019-10-11.

[174] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Trans. Information Theory*, 29(2):198–207, 1983.

[175] Benjamin Krebs and BMW. Connected Mobility Lab - center digitization.bayern. Accessed on October 2018, 2017.

[176] Renzo Navas, Göran Selander, and Ludwig Seitz. Lightweight Authenticated Time (LATe) Synchronization Protocol. Internet-Draft draft-navas-ace-secure-time-synchronization-00, Internet Engineering Task Force, October 2016. Work in Progress.

[177] Zach Shelby. Constrained RESTful Environments (CoRE) Link Format. RFC 6690, August 2012.

[178] Carsten Bormann and Paul E. Hoffman. Concise Binary Object Representation (CBOR). RFC 7049, October 2013.

[179] Yoav Nir and Adam Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 7539, May 2015.

[180] Zach Shelby, Klaus Hartke, and Carsten Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014.

[181] Paul A. Grassi, Michael E. Garcia, and James L. Fenton. Digital identity guidelines. *NIST special publication*, 800(63-3), 2017.

[182] Elaine Barker, Miles Smid, and Dennis Branstad. Telecommunications security guidelines for telecommunications management network. *NIST Special Publication*, 800(13), 1995.

[183] Elaine Barker. Recommendation for key management. *NIST special publication*, 800(57), 2016.

# Part VI

# Supplemental Information

# Appendix A

# Petri Nets Workflows

## A.1 Petri Net Markup Language (PNML) example

The exported PNML code of the supply chain use case presented in section 15.4 is shown in listing A.1. Some parts of the XML code are omitted for readability (brevity) reasons.

## A.2 Smart Contract Generation Solidity Code

The generated Solidity smart contract of the supply chain use case presented in section 15.4 is shown in listing A.2.

```
<?xml version='1.0' encoding='UTF-8'?>
<pnml>
  <net id='scd'>
    <place id='p9' isGlobalVar='true'>
      <initialMarking>
        <multiset>
          <item>
            <value>
              <object type='int'>19
              </object>
            </value>
            <multiplicity>1</multiplicity>
          </item>
        </multiset>
      </initialMarking>
    </place>
    <place id='p5' isGlobalVar='false'>
      <initialMarking>
        <multiset>
          <item>
            <value>
              <object type='bool'>true
              </object>
            </value>
            <multiplicity>1</multiplicity>
          </item>
        </multiset>
      </initialMarking>
    </place>

    ...<omitted for brevity>...

    <transition id='t12'>
      <guard id='1'>
        <condition>scanned_id == asset_id
        </condition>
        <symbol>correct_asset</symbol>
        <expression>true</expression>
      </guard>
      <guard id='2'>
        <condition>scanned_id != asset_id
        </condition>
        <symbol>stop</symbol>
        <expression>true</expression>
      </guard>
    </transition>

    <transition id='t2'>
      <guard id='1'>
        <condition>
          _humidity>50 || _temp>25
        </condition>
        <symbol>threshold_reached</symbol>
        <expression>true</expression>
      </guard>
      <guard id='2'>
        <condition>_humidity>50 || _temp>25
        </condition>
        <symbol>humidity</symbol>
        <expression>humidity</expression>
      </guard>
      <guard id='3'>
        <condition>_humidity>50 || _temp>25
        </condition>
        <symbol>temperature</symbol>
        <expression>_temp</expression>
      </guard>
      <guard id='4'>
        <condition>_temp <=25 && _humidity <=50
        </condition>
        <symbol>threshold_reached</symbol>
        <expression>false</expression>
      </guard>
    </transition>

    ...<omitted for brevity>...

    <arc id='ge6' source='t2' target='p5'>
      <inscription>
        <expression>threshold_reached
        </expression>
      </inscription>
    </arc>
    <arc id='ge10' source='p9' target='t2'>
      <inscription>
        <variable>temperature</variable>
      </inscription>
    </arc>
    <arc id='ge8' source='p7' target='t2'>
      <inscription>
        <variable>humidity</variable>
      </inscription>
    </arc>
    <arc id='ge13' source='p11' target='t12'>
      <inscription>
        <variable>scanned_id</variable>
      </inscription>
    </arc>
    ...<omitted for brevity>...

  </net>
</pnml>
```

Listing A.1: The snippet of the PNML of the Petri Net model shown earlier.

```
pragma solidity ^0.4.24;

contract Scd {

    // Template Variables
    address owner;

    // Global variables
    bool threshold_reached;
    bool threshold_reached_enabled;
    uint asset_id = 8462674;
    bool asset_id_enabled;
    uint temperature;
    bool temperature_enabled;
    uint humidity;
    bool humidity_enabled;
    bool asset_exists;
    bool asset_exists_enabled;

    constructor()
    public {
        owner = msg.sender;
    }

    // State Machine
    function execute(
        uint _scanned_id,
        uint _temp,
        uint _humidity
    )
    public
    returns(string _functionName) {
        asset_id_enabled = true;

        if (asset_id_enabled)
            fire_t12(_scanned_id);

        if (asset_exists_enabled)
            fire_t2(_temp,_humidity);

        return "execute";
    }

    // Transitions
    function fire_t2(
        uint _temp,
        uint _humidity
    )
    internal
    returns(string _functionName) {

        // LOCAL FUNCTION VARS
        bool asset_exists_local = asset_exists;
        uint humidity_local;
        uint temperature_local;
        bool threshold_reached_local;

        if (_humidity > 50 || _temp > 25) {
            // Consume
            humidity_enabled = false;
            temperature_enabled = false;
            threshold_reached_enabled = false;

            // Assign
            threshold_reached_local = true;
            humidity_local = _humidity;
            temperature_local = _temp;

            // Produce
            threshold_reached = threshold_reached_local;
            threshold_reached_enabled = true;
            humidity = humidity_local;
            humidity_enabled = true;
            temperature = temperature_local;
            temperature_enabled = true;
        }

        if (_temp <= 25 && _humidity <= 50) {
            // Consume
            threshold_reached_enabled = false;

            // Assign
            threshold_reached_local = false;

            // Produce
            threshold_reached = threshold_reached_local;
            threshold_reached_enabled = true;
        }

        return "t2";
    }

    function fire_t12(uint _scanned_id)
    internal
    returns(string _functionName) {

        // LOCAL FUNCTION VARS
        uint asset_id_local = asset_id;
        bool asset_exists_local;

        if (_scanned_id == asset_id_local) {
            // Consume
            asset_id_enabled = false;

            // Assign
            asset_exists_local = true;

            // Produce
            asset_exists = asset_exists_local;
            asset_exists_enabled = true;
        }

        return "t12";
    }
}
```

Listing A.2: The generated Solidity smart contract for the Petri Net model shown earlier.

# Appendix B

# Privacy Enhanced Tokens (PAT) Profile

The PAT protocol includes three actors: Resource Server (RS), Client (CL), and Authorization Server (AS). PAT message flow is shown in Fig. B.1 and in the following, they are described in detail. Note: the messages in [square brackets] mean they are optional.

A PAT message sent from actor A to actor B is represented using the following notation: "A -> B : Message Name". The arrow between entities depict the message flow direction.

## B.1 RS <-> AS: Security Association Setup

The RS and its AS share a long term shared secret (K) when device commissioning but how it is done is out of scope. The long term secret is considered a symmetric secret in the context of PAT profile but ACE-OAuth allows also asymmetric secret. During the commissioning phase, the internal clock of RS is synchronized with the AS and the cryptographic algorithms, parameters and profiles supported by the RS are registered with AS. For time authenticated time synchronization, the Lightweight Authenticated Time (LATe) Synchronization Protocol [176] could be used together with PAT profile.

## B.2 [CL -> RS : Resource-Request]

Initially, CL may not have a valid Access-Token (AT) to access a protected R hosted in RS. If the CL does not have the corresponding AS-Info to request AT from AS. To receive the appropriate AS-Info, CL may send a Resource-Request message to RS without a valid AT.

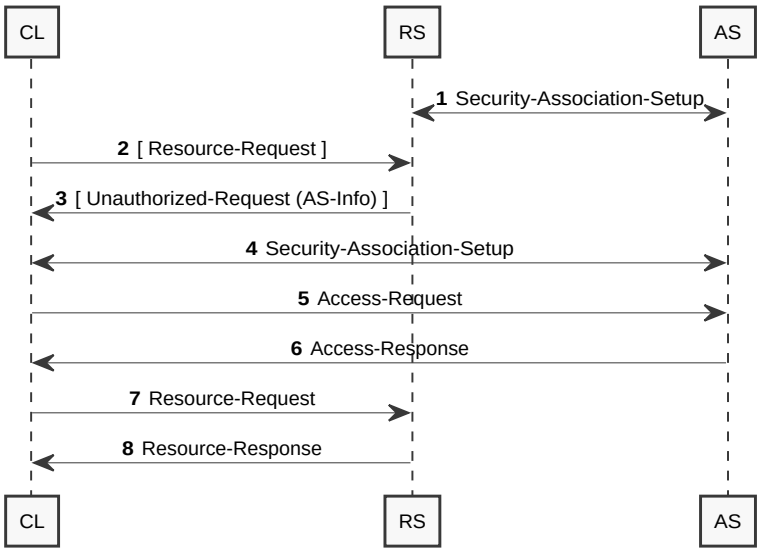



Figure B.1: PAT protocol message flow

To enable resource discovery, RS may expose the URI `/.well-known/core` as described in RFC 6690 [177], but this resource itself may be protected. Thus, CL can optionally make a CoAP GET request to the URI `/.well-known/core`.

## B.3 [RS -> CL : Un-Authorized-Request(AS-Info)]

Once RS receives a resource request from a CL, it checks:

- If CL has attached a valid Access-Token (AT) with the request or not. If not, then RS must respond to CL with code 4.01 unauthorized request. Optionally, RS may include information to reach the /token endpoint exposed by the AS (AS-info).
- If CL has attached the valid Access-Token (AT), but not for the requested resource then RS must respond with 4.03 (Forbidden)
- If CL has a valid access token, but not for the method requested then RS must respond with 4.05 (Method Not Allowed)
- If CL has a valid access token, then RS must follow the procedure described in Sec. B.8 to create a valid response to CL.

Figure B.2 shows the sequence of messages with CoAP types between CL and RS for the above Un-Authorized-Request.

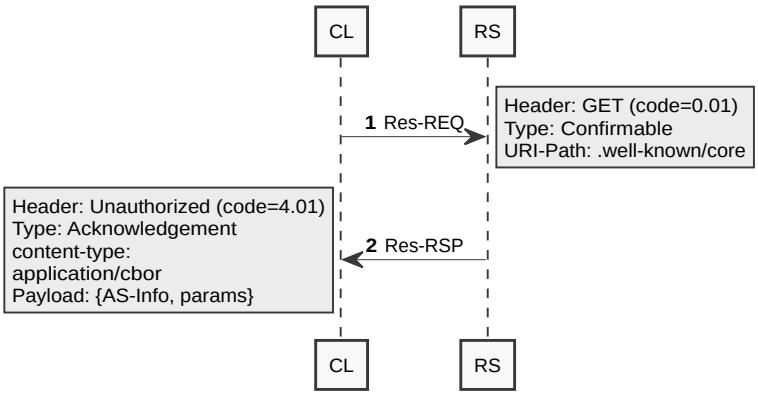



Figure B.2: C<->RS Resource-Request and Unauthorized as response

```
Header: 4.01 (Unauthorized)
Content-Type: application/cbor+pat;
              pat-type="UnAuthReq"
Payload:
  {
   AS-Info: "coaps://as.example.com/token",
   #protected
   TIC params:
     {
      nonce: 'rs-nonce..',
      kid: '..',
      [alg]: '..',
      TAG: '..'
     }
  }
```

Listing B.1: AS information + LATe time synchronization payload

The RS may send an unauthorized response with additional information such as AS-Info and parameters (params) to mitigate attacks based on time synchronization. The Lightweight Authenticated Time (LATe) synchronization protocol's description of scenario in section 6.2 suits PAT protocol and can be used for time synchronization. LATe protocol is published as an IETF draft (see [176]).

The response payload from AS may include AS information (AS-info) and parameters (params) that include LATe time synchronization's information object such as key-reference ID (kid) shared secret between RS and AS, a nonce to prevent replay attacks and optionally, the Message Authentication Codes (MsgAuthCode) algorithm used for producing the MsgAuthCode. RS is recommended to create a MsgAuthCode tag for LATe parameters and objects. Listing B.1 shows RS example response message to CL encoded using CBOR [178] with pat-type="UnAuthReq".

## B.4 CL <-> AS : Security Association Setup

After receiving AS-Info from RS, CL must establish a secure channel with the AS for making further requests. The AS establishes a confidential channel with CL. How this secure connection (e.g., a DTLS channel) should be established is out of the scope.

Notice that, it is important to ensure that the connection between AS and CL must be reliable and secure since the PAT protocol relies on the fact that the messages exchanged between CL and AS are protected and confidential. If the Client is also a constrained device, then CL may use DTLS-profile as described in [49] to create a secure channel with the AS.

The RO registers information about clients in AS and therefore, AS may have an access-control list for allowing authorized clients to access a particular list of resources. How this access control list or similar mechanisms are implemented is out of scope. With this access-control list, AS can validate two things: (a) whether the client is allowed to establish a secure connection or not; (b) whether the client has privileges to access the requested R in RS or not. In the next steps, these details are discussed.

## B.5 CL -> AS : Access-Request

Once CL establishes a secure communication channel with AS, CL sends an access-request to AS at endpoint /token requesting an access token RS as described in [48].

Optionally, CL includes the details about the resources R in the request. If optional info is not available, then AS should prepare an access token with default permissions. Fine-grained access to resources (R) of RS depends on the infrastructure or services the RS offers. Listing B.2 shows the detailed access request message from CL to AS.

## B.6 CL <- AS : Access-Response

After AS receiving an access-request message from a CL and successful validation AS performs the following actions:

- If the access request from CL is valid (i.e., operations are covered by the privileges defined by the RO), then AS prepares the AT and sends it with COAP response code 2.01 (Created).

```
Header: POST (Code=0.02)
Uri-Host: "coaps://as.example.com"
Uri-Path: "token"
Content-Type: "application/cbor+cwt+late ;
late-type=tic"
Payload:
   {
     "grant_type" : "client_credentials",
     "client_id": "...",
     "client_secret": "...",
     "aud" : "tempSensor",
     "scope": "GET|POST",
     ... omitted for brevity ...
     LATe params:
       { [if exist]
          nonce:'rs-nonce..', # same rs-nonce sent by RS
          kid: '..'
       }
     TAG: .. # LATe MsgAuthCode tag produced by RS
               using the shared key k with AS.
   }
```

Listing B.2: Example Client Access-Request message to AS

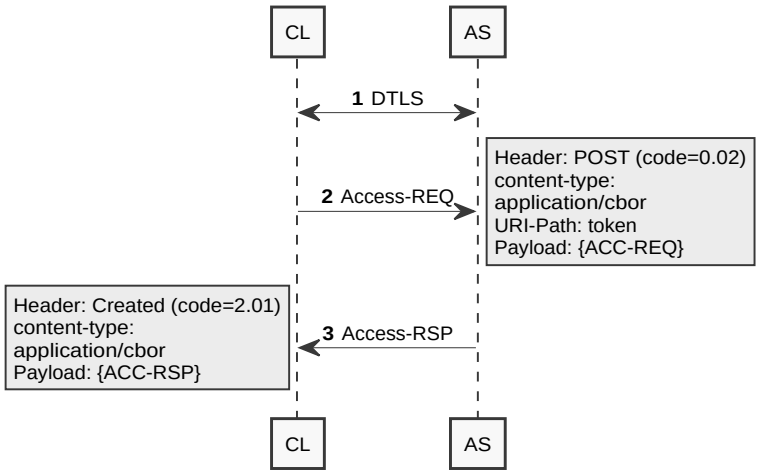



Figure B.3: Example CL Access-Request message to AS and its response from AS

- If the Access-Request from CL contains Lightweight Authenticated Time (LATe) time synchronization objects then appropriate parameters are included in the response.
- If the client request is invalid then AS must send appropriate COAP error response code as specified in [48].

Figure B.3 shows an access-request message sent from CL to AS to get an access token (AT).

The AS constructs the AT and the `verifier` (the symmetric PoP key) as the answer for a valid access request from CL. The contents of the access-response (ACC-RSP) payload are logically split into two parts: the Access-Token (AT) and the `Verifier` (which is the Symmetric PoP key), they are explained in detail below.

### B.6.1 Access-Token construction:

The Access-Token (AT) is constructed by AS using the CBOR Web Token (CWT) claim parameters as described below:

- `iss` (issuer): AS identity
- `aud` (audience): resource server URI or specific resource URI for a fine-grained procedure.
- `exp` (Expiration Time): token expiration time
- `iat` (Issued At): token issued at time by AS
- `cti` CWT ID should be unique for every AT.

- `scp` (Scope): Note that scp is not a CWT claim. It can specify allowed methods such as GET, POST, PUT or DELETE.

Other CWT claims are optional. It is recommended to avoid the CWT claim `sub` (subject) as it exposes client identity.

### B.6.2 Verifier or PoP key construction:

Verifier (the Symmetric PoP key), is constructed as the result of the following operation: `G (K, AT)` where,

- `G`: the MAC algorithm which is used to create the `verifier`, we propose Poly1305 RFC 7539 [179]. Notice that `G` is a function which takes two parameters (key, data) as input and it produces a keyed digest as the output
- `K`: the shared key between AS and RS
- AT: constructed using CWT claims as explained before

The Client will use the `verifier` as the key material to communicate with the RS, i.e., if CL wants to encrypt its payload, it uses the `verifier` as the key. Note: the `verifier` is never sent in cleartext.

Therefore, the access-response message with the access token and the `verifier` must be sent to CL through a secure channel. The shown example considered a DTLS channel between CL and AS.

The time-synchronization between AS and RS may be implemented based on the application requirements using LATe time synchronization protocol [176]. The AS should specify required parameters as described in [48] such as the type of token, etc. Also, if the Access-Request from CL does not include any profile, AS must signal CL to use an appropriate or default profile.

Listing B.3 shows the example of an Access-Response sent from AS to CL after successful validation of C's credentials which were presented using an Access-Request message.

## B.7 CL -> RS : Resource-Request

Once CL receives the access-response from AS, CL can construct a Token (Tk) which will demonstrate that CL has got the sufficient authorization to access resources (R) in RS. Note: the difference between Access-Token (AT) and Token (Tk) is that the former is constructed by AS and the later is constructed by CL using the AT from AS.

```
Header: 2.01 (Created)
Content-Type: application/cbor+cwt+pat; pat-type="tk"
Location-Path: token/...
Payload:
{
   "access token": b64'SlAV32hkKG ...
   {
    "iss": https://as.example.com
    "aud": "tempSensor",
    "scp": "read",
    "iat": 1...,
    "cti": "..", # Unique can be a Sequence Number
    "exp": 5...,
    "alg": "chacha20/poly1305",
    "profile": "ace_pat"
   }
  "cnf":
  {
   COSE_Key: {
     "kty": "symmetric",
     "kid": h'...
     "k": b64'jb3yjn...  #[verifier]
   }
  }
 LATe Param:{
   as_time: '..',
   nonce: 'rs-nonce..',
   }
 tag: '..' #LATe tag
}
```

Listing B.3: Example Access-Response message sent from AS to CL with detailed CWT params and payload info

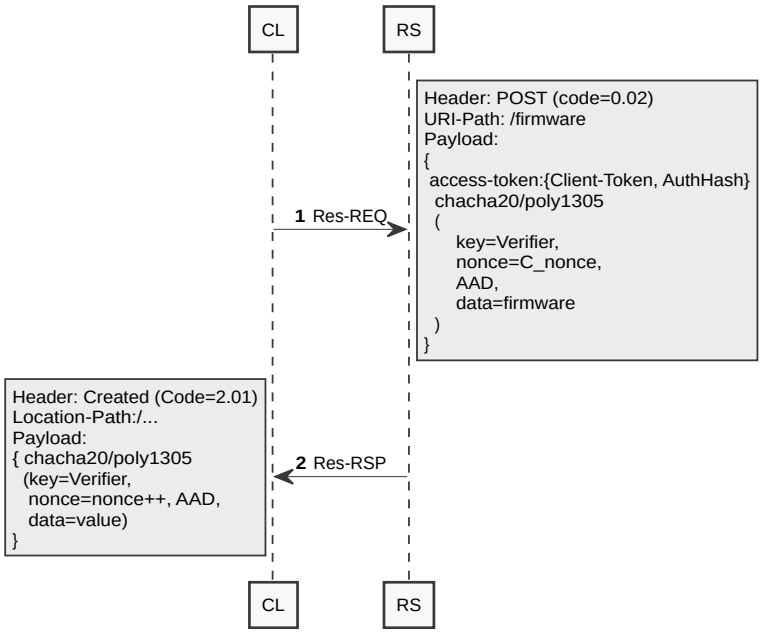



Figure B.4: RES-REQ from CL using /authz-info implemented at RS

A Tk must be attached to each RES-REQ sent to RS by CL. If payload data are included, then CL should encrypt the payload using the `verifier` as key and optionally it can include an Authentication Hash (`AuthHash`). The `AuthHash` is constructed by concatenating both `verifier` and a nonce (`C_nonce`) created by the client i.g., `AuthHash = hash(verifier + nonce)`. PAT profile provides necessary recommendations i.e., using Authenticated Encryption with Associated Data (AEAD) of chach20/poly1305.

As an example if CL performs:

- A CoAP GET request without payload. In this case, the request from CL may be sent un-encrypted since it does not include confidential data, but the response from RS with payload must be always encrypted and only the valid CL must be able to decrypt it.
- A CoAP POST, PUT, or DELETE request with payload must be protected and encrypted by using AEAD. PAT profile proposes to use ChaCha20-Poly1305-AEAD authenticated encryption mechanism, while using the `verifier` (PoP key) as the key and a client nonce (`C_nonce`). The `AuthHash` may be protected by using it as Additional Authentication Data (AAD) in the AEAD algorithm.

The RS must implement /authz-info endpoint to allow any Client to transfer the Tk as described in [48] and listing B.4 shows the GET request from CL to RS described in [48], with `pat-type ="AuthReq"`.

```
Header: GET
Content-Type: application/cose+cbor+pat;
                  pat-type="AuthReq";
Uri-Host: "coap://rs.example.com"
Uri-Path: /authz-info
Payload:
{ token: {
       "access token": .. {
         "aud": "tempSensor"
         "scp": "read"
         ... #CWT omitted for brevity.
       }
       "nonce": ..
       "AuthHash": .. #[AuthHash=hash(verifier+nonce)]
  }
  LATe:{
  time:'as-time',
  nonce:'rs-nonce',# rs-nonce from RS LATe object
  } tag: '..' #LATe tag
}
```

Listing B.4: Example of valid GET RES-REQ from CL to RS including time-sync using endpoint /authz-info.

Table B.1: RS Internal state table of ATs

| Verifier | cti_x-1 | exp | scp | next cti (cti_x) |
|---|---|---|---|---|
| G(k,AT) | cti_x0 = cti of AT | of AT | of AT | cti_x1 = hash(cti_x0, verifier) |

The CL performs a GET request to "tempSensor" using CWT claim `aud`, and together CL also transfers Tk to the RS. PAT allows performing both RES-REQ and transferring authorization information in RES-REQ. The next example shows how to perform a resource request if CL performs a POST request with encrypted payload information.

Listing B.5 shows an example of POST Resource-Request from CL to RS described in [48], with pat-type="AuthReq". Listing B.5 shows the POST Resource-Request where the Uri-Path "/authz-info" allows the authorized client to perform firmware upgrade on the RS using the CWT claim "aud:firmwareUpd". PAT recommends protecting sensitive information such as the payload using AEAD algorithm (chacha20/poly1305). The client should use the `verifier` or PoP key as the key, a nonce, and `AuthHash` as AAD.

## B.8 RS -> CL : Resource-Response

After receiving the request, RS verifies the Tk. RS can construct its own version of the `verifier` or PoP key by performing `G (K, AT)` from the AT. RS checks whether `AuthHash = Hash (verifier + nonce)` is valid or not. If Tk and AuthHash are valid, then RS sends an encrypted response using the `verifier` (PoP key).

When the RES-REQ with a Tk arrives from CL to RS, RS must evaluate the resource request and the Tk in the following order:

- Step 0: Check whether the contents of Tk are derived from an AT or not.
- Step1: If Tk contains the AT from AS, extract AT. Extract nonce and Authentication Hash (AuthHash) from the request message.
  - Step1.1: (If available) Verify the freshness of the sequence number (cti) in the access token presented by AS.
  - Step1.2: Generate the `verifier` by computing `G(k, AT)` where `K` is the shared key between AS and RS.
  - Step1.3: Compute a verification hash as `hash(verifier+nonce)` and compare the result with AuthHash for correctness.

```
Header: POST (Code=0.02)
Content-Type: application/cose+cbor+pat;
              cose-type="encrypt0";
              "pat-type="AuthReq";
Uri-Host: "coap://rs.example"
Uri-Path: /authz-info
Payload:
{# COSE
        token: {
        "access token": .. {
                "aud": "firmwareUpd"
                "scp": "write"
                ... CWT omitted for brevity,
                }
        "nonce": ..  # nonce
        "AuthHash": .. # [AuthHash=hash(verifier+nonce)]
         LATe params:{
             time:'as-time',
                 nonce:'rs-nonce', # rs-nonce from \gls{RS} LATe objects
             } tag: '..' #LATe tag
        }
        # COSE_Encrypt0 + COSE_MAC0 Protected
        ciphertext:{
        #Chacha20/Poly1305 AEAD payload using
                # key=verifier,
                # nonce=..,
                # AAD=AuthHash
                },
        tag: ..
}
```

Listing B.5: Example of valid POST request from CL to RS

  - Step1.4: Check if the access token has valid CWT parameters such as `aud, scp, exp, nbf`, etc for the requested resource or action to be performed.
  - Step1.5: If LATe params info available, synchronize RS internal clock using LATe object as described in [176].

- Step2: If the token is valid, RS should create a temporary internal state as shown in Tab. B.1 below with details of CWT claims `cti, exp, scp`, and the `verifier` (PoP key).
- Step 3: If the token is valid, then RS decrypts the payload from the client (if exist) `verifier` (PoP key).
- Step 4: After that, RS prepares the response and encrypts the payload with a fresh nonce (`RS_nonce`), PoP key. Only the CL with a valid key (the `verifier`) can decrypt the payload.

The RS internal state table which is shown in Tab. B.1 also includes "next cti". The next `cti (cti x)` value is computed as the Hash of previous `cti (cti x-1)` and the `verifier`. The purpose of this is explained in the Sec. B.9.

### B.8.1 RS Response-codes to CL

- If the Tk is valid – as discussed above –, then RS must respond with payload-data as described above with the appropriate response code as described in RFC 7252 [180]. For example, a POST request with 2.01 (created) or 2.04 (changed).
- If the Tk is invalid, then RS must respond with code 4.01 (Unauthorized)
- If the Tk is valid but does not match the `aud` or resource CL is requesting for then RS must respond with code 4.03 (Forbidden)

## B.9 Construction of Derived-Tokens (DT)

The objectives to create Derived-Token (DT) are the following:

- To produce Unlinkable Token (Tk). It is not efficient for the client to request a new AT from AS everytime. Also, if CL uses the same AT from AS, info such that the same client is accessing the resource several times can leak because of the CWT claim `cti` (which is a unique identifier).
- To reduce Tk size (efficiency in transport) that the client must send to RS /authz-info in every resource request.

- To create Tk that may have limited access to protected-resources – fine-grained resource access tokens – from the original AT that could grant more privileges to protected-resources on RS. For example, an AT could provide permissions to access all protected-resources on RS via CWT claims audience `aud` and scope `scp`. The client could derive a Tk providing access to a reduced set of protected-resources available on RS from the initial AT.

### CL -> RS : Resource-Request via Derived-Tokens (DT)

The CL receives an encrypted response from RS after its first RES-REQ with the AT from AS. The CL creates a new Derived-Token (DT) using CWT claims as described below. In order to minimize the data size, we use only the claims which are required. Listing B.6 shows the detailed resource request via derived tokens.

- Client may prepare a DT with a subset of scope `scp` operations that the client received from the initial AT. It creates the first derived `cti_x1` by `hash(cti_x0 + verifier)` from the CWT claim `cti` of the original AT. The subsequent derivation of `cti_x` can be performed by a generic function `cti_x = hash(cti_x-1 + verifier)`. Note that DT must include all the necessary CWT claims such as `cti_x, aud, exp, scp`. All other CWT claims are optional.
- Client creates the `AuthHash=(verifer+nonce)`.
- Client prepares encrypted content using `verifier` as the key – if there is any payload –.
- Note: in AAD, CL includes AuthHash and Derived-Token (DT), so that the payload cannot be misused/exchanged with another RES-REQ or nonce.

### RS -> CL : Resource-Response to Derived-Token (DT)

After receiving the Tk which encapsulates the Derived-Token (DT) from CL, RS performs the following Steps. If any of them fails, then RS must send an UnAuthorized response to CL, and CL must use the first AT, which was received from the AS, or request a new AT based on RO. The Tab. B.2 shows the RS internal state table with an example.

The RS performs the following actions before creating a response:

- RS extracts CWT claim `cti (cti_x)` from the Derived-Token (DT) and checks if it exists in its internal state table. If RS finds the `cti_x`, then RS uses the corresponding `verifier`, `cti_x-1, exp, scp` to perform the validation of next steps.

```
Header: POST (Code=0.02)
Content-Type: application/cbor+cwt+cose++pat;
              cose-type="encrypt0";
              "pat-type="AuthReq";
Uri-Host: "coap://rs.example"
Uri-Path: /firmware
Payload:
{
        # COSE
        token:
        {      derived-token(DT):
                "aud": "firmwareUpd",
                "exp": ..
                "scp": "write",
                "cti": hash(cti_x+verifier)
                # cti_x=hash(cti_x-1+verifier).
        }
        "nonce": ..  # new nonce
        "AuthHash": h'bfa03.. #[Hash=(verifier+nonce)]
        # COSE_Encrypt0 + COSE_MAC0 Protected
        ciphertext:
        {
                # Chacha20/Poly1305 AEAD payload using
                # key=verifier,
                # nonce=..,
                # AAD=AuthHash,DT
                h'....omitted for brevity
        },
        tag: h'... omitted for brevity
}
```

Listing B.6: Example of valid Resource-Request from CL to RS using a derived-token(DT)

Table B.2: RS updating its internal table with cti values both old and current

| msg# | Verifier (V) | cti_x-1 | exp | scp | cti_x= hash(cti_x-1+V) |
|---|---|---|---|---|---|
| 0 | G(k,AT) | 0x00 | of AT | of AT | 0xAB hash(0x00+V) |
| 1 (upd) | G(k,AT) | 0xAB | of AT | of AT | 0xFF hash(0xAB+V) |

- RS checks that `cti_x = hash(cti_x-1 + verifier)`
- RS checks that `AuthHash = hash(verifier + nonce)`
- RS checks that the permissions are valid using `scp` and expiration time `exp`
- RS updates the new `cti_x-1, cti_x` in its internal state table
- RS creates an encrypted response to be sent to CL with a payload including payload-data.

# Glossary

**automata theory** "Automata theory is the study of abstract computing devices, or machines´´ [2]. 35

**bearer assertion** The token (or an assertion) a party presents as proof of identity, where possession of the assertion itself is sufficient proof of identity for the assertion bearer (see [181]).. 50

**data confidentiality** Data Confidentiality deals with protecting against the disclosure of information by ensuring that the data is limited to those authorized or by representing the data in such a way that its semantics remain accessible only to those who possess some critical information (e.g., a key for decrypting the enciphered data) (see [182]). 25

**data integrity** A property whereby data has not been altered in an unauthorized manner since it was created, transmitted or stored [183].. 25

**Internet-of-Things** IoT is defined as "[..] a dynamic global network infrastructure with self-configuring capabilities-based on standard and interoperable communication protocols where physical and virtual things have identities, physical attributes, virtual personalities, use intelligent interfaces, and are seamlessly integrated into the information network"(see[99]). vii, 3, 47, 210

**OAuth access token** OAuth access tokens are credentials used to access protected resources. An access token is a string representing an authorization issued to the client (see [48]).. 37

**policy language** " A language that is used to write or represent a security policy is called a *Policy Language*" [11].. 34

**principle of least privilege** "The principle of least privilege states that a subject should be given only those privileges that it needs in order to complete its task" (see [11]).. 30

**security mechanism** "A Security Mechanism is a method, tool, or procedure for enforcing a security policy" [11].. 25, 26

**security policy** "A Security Policy is a statement of what is, and what is not, allowed" [11]. 26, 27

**trust** "Trust is characteristic of an entity that indicates its ability to perform certain functions or services correctly, fairly and impartially, along with assurance that the entity and its identifier are genuine" [10].. 4

# Acronyms

**AAD** Additional Authentication Data. 199, 204

**ABAC** Attribute-Based Access Control. 33, 34, 37, 57, 58, 62

**AC** Access Control. 16, 18

**ACE** Authentication and Authorization for Constrained Environments. 5, 10

**ACL** Access Control List. 31, 34, 62

**AEAD** Authenticated Encryption with Associated Data. 199

**AI** Artificial Intelligence. 47

**AIOTI** Alliance for the Internet of Things Innovation. 21

**API** Application Programming Interface. 108

**AS** Authorization Server. xvii, xxiii, 7, 50, 61, 70, 88–91, 113, 114, 117–119, 122, 162, 191–194, 196, 197, 201, 203, 204

**AT** Access-Token. xix, 50, 88–91, 191, 192, 194, 196, 197, 201, 203, 204

**BPMN** Business Process Model and Notation. 43, 86

**CapBAC** Capability-Based Access Control. 60, 62

**CAS** Client Authorization Server. 89

**CBOR** Concise Binary Object Representation. 110, 193

**CCAAC** Capability-Based Context-Aware Access Control. 60

**CCS** Calculus of Communicating System. 42

**CL** Client. xvii, xxi, xxiii, 50, 70, 88–92, 121, 191–194, 196, 197, 199–206

**CoAP** Constrained Application Protocol. 16, 47, 49, 90, 91, 192, 199

**COI** Conflict of Interest. 27, 34

**CPN** Colored Petri Nets. 41, 59, 64, 78

**CRM** Customer Relationship Management. 57

**CSP** Communicating Sequential Processes. 42

**CWT** CBOR Web Token. 92, 110, 196, 201, 203, 204

**DAC** Discretionary Access Control. 28, 29

**DoS** Denial-of-Service. 129

**DT** Derived-Token. 89, 90, 203, 204

**DTLS** Datagram Transport Layer Security. 16, 87, 118, 162

**ENISA** European Union Agency for Network and Information Security. 48

**ERP** Enterprise Resource Planning. 57

**GDPR** General Data Protection Regulation. 13, 16, 48, 165

**HCAP** History-based Capability systems for IoT. 8, 60–62

**HR** Human Resource. 57

**HTTP** Hypertext Transfer Protocol. 47

**HTTPS** Hypertext Transfer Protocol Secure. 3

**IETF** Internet Engineering Task Force. 10, 88

**IIoT** Industrial Internet of Things. 15

**IoT** Internet-of-Things. vii, viii, 3–5, 7–13, 15–19, 21, 22, 34, 43, 47–49, 55–62, 69, 74, 75, 78, 79, 81, 83, 87, 90, 95, 96, 109, 119, 132, 134, 139, 155–157, 161, 162, 210, *Glossary:* Internet-of-Things

**IPSec** Internet Protocol Security. 3

**JWT** Java Web Token. 110, 121, 122

**KB** Kilo Bytes. 47

**MAC** Mandatory Access Control. 28, 29

**MBSE** Model-Based Systems Engineering. 85

**MsgAuthCode** Message Authentication Codes. 193

**NIST** National Institute of Standards and Technology. 27, 28, 69

**OASIS** Organization for the Advancement of Structured Information Standards. 37, 43

**OAuth** Web Authorization. 5, 10, 32, 37, 69

**OMG** Object Management Group. 43, 86

**Open-PN** Open Petri Nets. 79, 84

**OWASP** Open Web Application Security Project. 15, 131

**PAP** Policy Administration Point. 37

**PAT** Privacy Enhanced Tokens for ACE OAuth. 88, 92, 118, 135, 155, 156, 162

**PDP** Policy Decision Point. 37, 57, 58

**PEP** Policy Enforcement Point. 37, 57

**PIN** Personal Identification Number. 18

**PIP** Policy Information Point. 37, 57, 58

**PKI** Public Key Infrastructure. 129

**PN** Petri Nets. 77, 78, 95, 98, 101, 103–105, 111, 119

**PNML** Petri Net Markup Language. 97–99, 101, 103, 111, 120, 121

**PoP** Proof-of-Possession. 50, 89, 90, 92, 196, 197, 199, 201, 203

**PRP** Policy Retrieval Point. 37

**R** Resource. 88, 89, 110, 122, 191, 194

**RAM** Random Access Memory. 47

**RBAC** Role-Based Access Control. 18, 32–34, 37, 57, 62

**RERUM** REliable, Resilient and secUre IoT for sMart city applications. 88, 90

**REST** Representational State Transfer. 120

**RFC** Request for Comments. 37

**RO** Resource Owner. xxiii, 50, 70, 88–92, 110–112, 115, 117, 121, 194, 204

**RS** Resource Server. xvii, xix, xxi, xxiii, 7, 50, 61, 70, 87–92, 110, 117, 118, 120, 122, 162, 191–194, 197, 199–204, 206

**SC** Smart Contract. 96–98, 104, 105

**SoS** System of Systems. 85, 86

**SQL** Structured Query Language. 128

**SysML** Systems Modeling Language. 13, 22, 85, 86, 97, 98, 107, 108, 115, 119, 139, 162

**TBAC** Task-Based Access Control. 33, 34

**TCSEC** Trusted Computer System Evaluation Criteria. 27

**TEE** Trusted Execution Environment. 121, 135, 137, 156

**Tk** Token. 89, 90, 197, 199, 201, 203, 204

**TLS** Transport Layer Security. 3, 16, 47, 118

**TPM** Trusted Platform Module. 135, 137

**TPN** Timed Petri Nets. 41

**UML** Unified Modeling Language. 22, 85, 86, 119

**WF** Workflow. 110, 113, 117, 121, 122, 146–148, 152

**WF-App** Workflow Application. 110, 111, 113, 114, 117–123, 132, 162

**WF-Store** Workflow store. 110, 112, 118–120

**WFAC** Workflow-Aware (or Workflow-Driven) Access Control. viii, xvi, 8–13, 17, 37, 56, 62, 69, 70, 74, 75, 81, 85, 92, 95, 107–110, 115, 118–120, 122, 127–132, 134–136, 139, 143, 155, 156, 161–163, 165

**WP** Workflow Participant. 87, 88, 110, 113, 117, 118, 122, 123

**WS-BPEL** Web Services Business Process Execution Language. 43

**XACML** Extensible Access Control Markup Language. 33, 34, 37

**XML** Extensible Markup Language. 43

# Index