Architectural Uncertainty Analysis for Access Control Scenarios in Industry 4.0

Master's Thesis of

Nicolas Boltz

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

Reviewer:	Prof. Dr. Ralf H. Reussner
Second reviewer:	Prof. Dr.-Ing. Anne Koziolek
Advisor:	M.Sc. Maximilian Walter
Second advisor:	M.Sc. Sebastian Hahner

11. Jan 2021 – 12. Jul 2021

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

PLACE, DATE

. . . . . .

(Nicolas Boltz)

Abstract

Industry 4.0 systems are characterized by their high complexity, connectivity and data exchange. Due to these characteristics, it is critical to ensure confidentiality of data. An often-used mechanism to ensure confidentiality is the use of access control. Based on software architecture modeling, access control can conceptually be applied during design time of the system, which enables early identification of potential confidentiality issues and the ability to analyze the impact of what-if scenarios on confidentiality before deploying changes. However, uncertainties of the systems environment, which result from the abstract view of the software architecture model, or ambiguity in the early stages of development can have an direct effect on existing access control policies, which might result in reduced confidentiality. To mitigate their effect on access control, it is important to identify and handle these uncertainties.

In this thesis, we present our approach to handle uncertainty in access control during design time. We define a characterization of uncertainties in access control on the architectural level to provide a better understanding and overview of the kinds of uncertainty that are present. We describe a concept of trust in the validity of access control properties, as a way to handle uncertainty, that has been described in publications defining or extending access control models. The concept of trust is a composition of environmental factors that impact the validity of and consequently trust in access control properties. We propose the use of fuzzy inference systems as a way of defining how environmental factors are combined to calculate trust values for each individual access control property. These trust values are than used by an analysis process to identify issues which can result from a lack of trust.

We extend an existing data flow diagram approach to design time information flow and access control analysis with our concept of trust. Our approach of adding knowledge to a software architecture model and providing a way to analyze model instances for access control violations shall enable software architects to increase the quality of models and further verify access control requirements under uncertainty, in early stages of the software development.

We evaluate the applicability based on the availability of necessary data during different phases of development and the potential value that can be added to existing systems. We also measure the accuracy of the analysis in identifying issues and the scalability regarding the execution time, when individually scaling the model aspects we add as part of our approach.

Zusammenfassung

Industrie 4.0-Systeme zeichnen sich durch ihre hohe Komplexität, Konnektivität und ihren hohen Datenaustausch aus. Aufgrund dieser Eigenschaften ist es entscheidend, eine Vertraulichkeit der Daten sicher zu stellen. Ein häufig verwendetes Verfahren zum Sicherstellen von Vertraulichkeit ist das Verwenden von Zugriffskontrolle. Basierend auf modellierter Softwarearchitektur, kann eine Zugriffskontrolle bereits während der Entwurfszeit konzeptionell auf das System angewendet werden. Dies ermöglicht es, potentielle Vertraulichkeitsprobleme bereits früh zu identifizieren und bietet die Möglichkeit, die Auswirkungen von Was-wäre-wenn-Szenarien auf die Vertraulichkeit zu analysieren, bevor entsprechende Änderungen umgesetzt werden. Ungewissheiten der Systemumgebung, die sich aus Unklarheiten in den frühen Phasen der Entwicklung oder der abstrakten Sicht des Softwarearchitekturmodells ergeben, können sich jedoch direkt auf bestehende Zugriffskontrollrichtlinien auswirken und zu einer reduzierten Vertraulichkeit führen. Um dies abzuschwächen, ist es wichtig, Ungewissheiten zu identifizieren und zu behandeln.
In dieser Arbeit stellen wir unseren Ansatz zum Umgang mit Ungewissheiten der Zugriffskontrolle während der Entwurfszeit vor. Wir erstellen eine Charakterisierung von Ungewissheiten in der Zugriffskontrolle auf der Architekturebene, um ein besseres Verständnis über die existierenden Arten von Ungewissheiten zu erhalten. Darauf basierend definieren wir ein Konzept des Vertrauens in die Gültigkeit von Eigenschaften der Zugriffskontrolle. Dieses Konzept bietet die Möglichkeit mit Ungewissheiten umzugehen, die bereits in Publikationen zu Zugriffskontrollmodellen beschrieben wurden. Das Konzept des Vertrauens ist eine Zusammensetzung von Umgebungsfaktoren, die die Gültigkeit von und folglich das Vertrauen in Zugriffskontrolleigenschaften beeinflussen. Um Umgebungsfaktoren zu kombinieren und so Vertrauenswerte von Zugriffskontrolleigenschaften zu erhalten, nutzen wir Fuzzy-Inferenzsysteme. Diese erhaltenen Vertrauenswerte werden von einem Analyseprozess mit in Betracht gezogen, um Probleme zu identifizieren, die aus einem Mangel an Vertrauen entstehen.
Wir erweitern einen bestehenden Ansatz zur Analyse von Informationsfluss und Zugriffskontrolle zur Entwurfszeit, basierend auf Datenflussdiagrammen. Das Wissen, welches wir mit unserem Konzept des Vertrauens hinzufügen, soll Softwarearchitekten die Möglichkeit geben, die Qualität ihrer Modelle zu erhöhen und Anforderungen an die Zugriffskontrolle ihrer Systeme bereits in frühen Phasen der Softwareentwicklung, unter Berücksichtigung von Ungewissheiten zu verifizieren.
Die Anwendbarkeit unseres Ansatzes evaluieren wir anhand der Verfügbarkeit der notwendigen Daten in verschiedenen Phasen der Softwareentwicklung, sowie des potenziellen Mehrwerts für bestehende Systeme. Wir messen die Genauigkeit der Analyse beim Identifizieren von Problemen und die Skalierbarkeit hinsichtlich der Ausführungszeit, wenn verschiedene Modellaspekte individuell vergrößert werden.

Contents

List of Figures

List of Tables

1 Introduction

Organizations always try to optimize and streamline production lines and supply chains to increase productivity and save costs. To achieve these goals, the use of the concepts and technologies associated with cyber-physical systems (CPS), the Internet of Things (IoT), and the Internet of Services (IoS) have steadily increased. This lead to the concept of Industry 4.0 (I4.0) [60] or Industrial Internet of Things (IIoT) [11]. Boyes et al. [11] describe the IIoT as a system that is comprised of multiple subsystems, including networked smart objects, cyber-physical assents, and generic information technologies, as well as optional cloud or edge computing platforms. The interconnected system enables real-time, intelligent, and autonomous access, collection, analysis, and exchange of process, product, and/or service information. This way, the overall production value can be optimized. The production process can be streamlined by implementing self-organizing product lines and supply chains, with ad-hoc cooperation between the involved machines, humans, and organizations. For organizations, this may improve product or service delivery, boost overall productivity, reduce labor cost and energy consumption, as well as reducing the build-to-order cycle [11].

I4.0 and IIoT systems use and process data of the involved entities as decentralized resources that communicate with each other. As much communication happens with outside organizations like, for example, suppliers or producers, there is much concern about unauthorized access to each other's data. The high flexibility and complexity and the amount of data exchange of such systems makes it more critical than ever to be able to ensure the confidentiality of data [60].

An often-used measure to ensure this kind of confidentiality is using access control mechanisms to authorize access or processing of data. Depending on the underlying access control model, varying information is used to determine whether access should be granted or denied in a specific scenario. The information is processed by various services, which provide this information to the access control system. The services that provide this information to the system introduce an element of uncertainty to the access control that might implicate reduced validity or confidentiality. For example, there are multiple ways to obtain an employee's location, each with differing uncertainties: Physical access control systems can verify the position within a building as soon as the employee enters a room using his magnet card on an RFID reader. However, an RFID reader cannot know whether multiple people entered the room at the same time or whether the room has already been left. A GPS sensor might experience inaccuracies and varying processing delays depending on satellite reception or signal dampening [33]. Depending on the environment, these factors can result in a deviation of the position supplied by the GPS sensor of multiple meters. This deviation causes, e.g., the assignment of a person to a specific room to be error-prone and reduces the validity of this information. Delays and the general age of the supplied information might also decrease its validity and increase uncertainty. The

flexible and distributed property of I4.0 and IIoT systems also introduces uncertainty about the current system structure, system behavior, and system environment [13]. This is why uncertainty in access control needs to be considered, especially in the context of I4.0 or IIoT.
In the following, we present the outline of our contribution and give an overview of the thesis' structure.

1.1 Contribution

We propose an approach of narrowing down and handling uncertainty in access control within the context of design time software architecture modeling of Industry 4.0 systems. We aim to define a characterization of uncertainties in access control on the architectural level, to be able to better understand the kinds of uncertainty that can be present. To do so, we first discuss how characteristics of existing uncertainty taxonomies [51], [75] can be applied to design time software architecture. We further classify properties used in access control and specific uncertainties that can be associated with these properties.
With this characterization, we aim to identify scenarios in research regarding access control in which uncertainty is described. We use these scenarios to further narrow down the characterization of uncertainty and define a way to handle the influence of uncertainty on access control. Using our approach shall enable software architects of I4.0 systems to not only analyze the validity of data-flow criteria based on fixed access control information but also to factor in uncertainties of the system and its environment.
As part of this thesis, we aim to answer the following research questions:

RQ1 Which kinds of uncertainty exist in access control on the architectural level, and how do these uncertainties manifest themselves?

RQ2 How can the uncertainty be handled in access control analyses on the architectural level?

We answer **RQ1** by creating the characterization of uncertainties in access control on the architectural level, we have described above.
To answer **RQ2**, we make model uncertainty explicit by defining the concept *trust* as a composition of environmental factors of the system that influence the correctness or validity of access control properties. We extend an existing data flow based software architecture analysis approach with this concept [66].

1.2 Outline

The remainder of this thesis is structured as follows: In Chapter 2, we explain our foundations, like uncertainty classification, data flow modeling, and fuzzy inference systems. In Chapter 3, we present related work regarding uncertainty in software and access control, as well as analysis of architectural models. We elaborate our characterization of uncertainty in the I4.0 environment, in Chapter 4, narrowing down and combining several views

on uncertainty in software architecture and access control. In Chapter 5 we present a running example that we use throughout the remainder of the thesis. Chapter 6 gives a detailed overview of the extension to a data flow model to include the concept of trust. We explain how we utilize the extended data flow model by extending an existing access control analysis approach in Chapter 7. In Chapter 8 we evaluate our approach. Chapter 9 concludes the thesis and provides an outlook on the future.

2 Foundations

This chapter gives an overview of the foundations the contributions of this thesis are based on. Section 2.1 covers the uncertainty characterization that we discuss in this thesis, which strongly bases on the taxonomy of uncertainty for software systems [51] and the general definition of uncertainty [75], that the taxonomy bases on. The implementation of our concept relies on two main foundations. Section 2.2 covers the data flow diagram approach [66], which builds our foundation for a data flow definition and analysis. In Section 2.3 we explain the fundamentals of fuzzy logic and fuzzy inference systems (FIS), which we utilize as part of our contribution.

2.1 Uncertainty Classification

One of the most prominent publications regarding uncertainty is the paper "Defining Uncertainty" from Walker et al. [75]. They describe a terminology and typology and a three-dimensional uncertainty matrix for identifying and characterizing uncertainty. The three dimensions of uncertainty are the location, level, and nature. The location indicates where the uncertainty manifests itself within the observed system. They define some generic locations that apply to most models. These locations are Context uncertainty, model structure uncertainty, model technical uncertainty, parameter uncertainty, and model outcome uncertainty. The level of uncertainty is dependent on the knowledge about uncertainty. It raises with an increase of uncertainty and a decrease of knowledge about the uncertainty. The level terminology is structured as follows: Determinism, statistical uncertainty, scenario uncertainty, recognized ignorance, and total ignorance. Nature is divided into uncertainty, resulting from the imperfection of knowledge (epistemic) or inherent variability present in a system, e.g., through human behavior or random events in nature (aleatory). The taxonomy of Perez-Palacin and Mirandola [51] is heavily based on

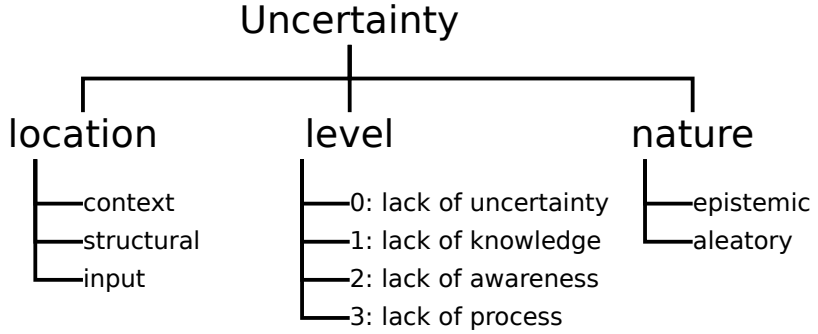


Figure 2.1: Uncertainty taxonomy of Perez-Palacin et al [51].

the three dimensions of the uncertainty matrix of [75], but coin them to be more focused on software engineering. This three-dimension classification is shown in Figure 2.1. They narrow down the originally very open definition of the locations to be specific to software systems. Instead of the progressive transition of levels proposed by [75], they propose a scale of five clearly defined levels of uncertainty, based on the five orders of ignorance [3]. Nature is defined slightly differently, but the overall proposition remains unchanged.

2.2 Data Flow Diagrams

The data flow diagram (DFD) approach of Seifermann et al. [66] is based on the DFD notion of DeMarco [18]. They propose an extension of the DFD syntax to overcome several limitations regarding the usability for confidentiality analyses.
Instances of the extended DFD are analyzed using a logic program given in Prolog [12]. The DFD instance is first transformed to Prolog facts and rules, which define the structure of the DFD and relationships between structural entities. These facts and rules are later used to define queries used to identify possibly illegal data flows.

2.2.1 Extended Data Flow Diagram

Data flow diagrams are unidirectional graphs representing the viewpoint on systems based on data processing. The extended DFD metamodel is shown in Figure 2.2 and is made up of several elements:
Graph *Nodes* are either *Actors*, *Processes*, *Stores* or *ActorProcesses*. Edges that connect *Nodes* are called *Data flows* and describe a data transmission between the connected nodes.
Actors are *source* and *sink* nodes, which start or terminate a sequence of data flows. In *Process* nodes, incoming data is transformed and passed on as outgoing data. The *ActorProcess* nodes describe complex data processing on behalf of an actor. *Store* nodes hold or emit data.
Characteristics represent properties of nodes. Each *Characteristic* is an instance of a *CharacteristicType*. This strong typing of *Characteristic* makes it possible to identify and match properties of the same type. The value of a property can be one out of a set of discrete values. These sets are represented by an *Enumeration* and the discrete values are represented by *Label*. Each *CharacteristicType* defines a *Enumeration* as the range of possible values for properties of the corresponding type. A *Characteristic* selects a subset of available labels from its type. Each node can hold multiple *Characteristics*. This means, that a *Node* has the properties that correspond to the labels of its held *Characteristics*.
A *Pin* describes required input data or output data of a node. The entirety of *Pins* of a *Node* represents the interface of the *Node*. Instead of connecting *Nodes*, the *DataFlow* edges connect output pins to input pins. Using *Pins* lowers the modeling effort by enabling the reuse of nodes. An additional edge from an input pin represents an alternative data flow. An additional edge to an output pin represents another forked data flow.
The way *Nodes* manipulate and transform data is represented by *BehaviorDefinitions*. A *BehaviorDefinition* consists of input and output pins and *Assignments*. The *Terms* of an

Assignment describe how labels are assigned to output pins. This assignment can refer to labels of input pins or constants. [66]

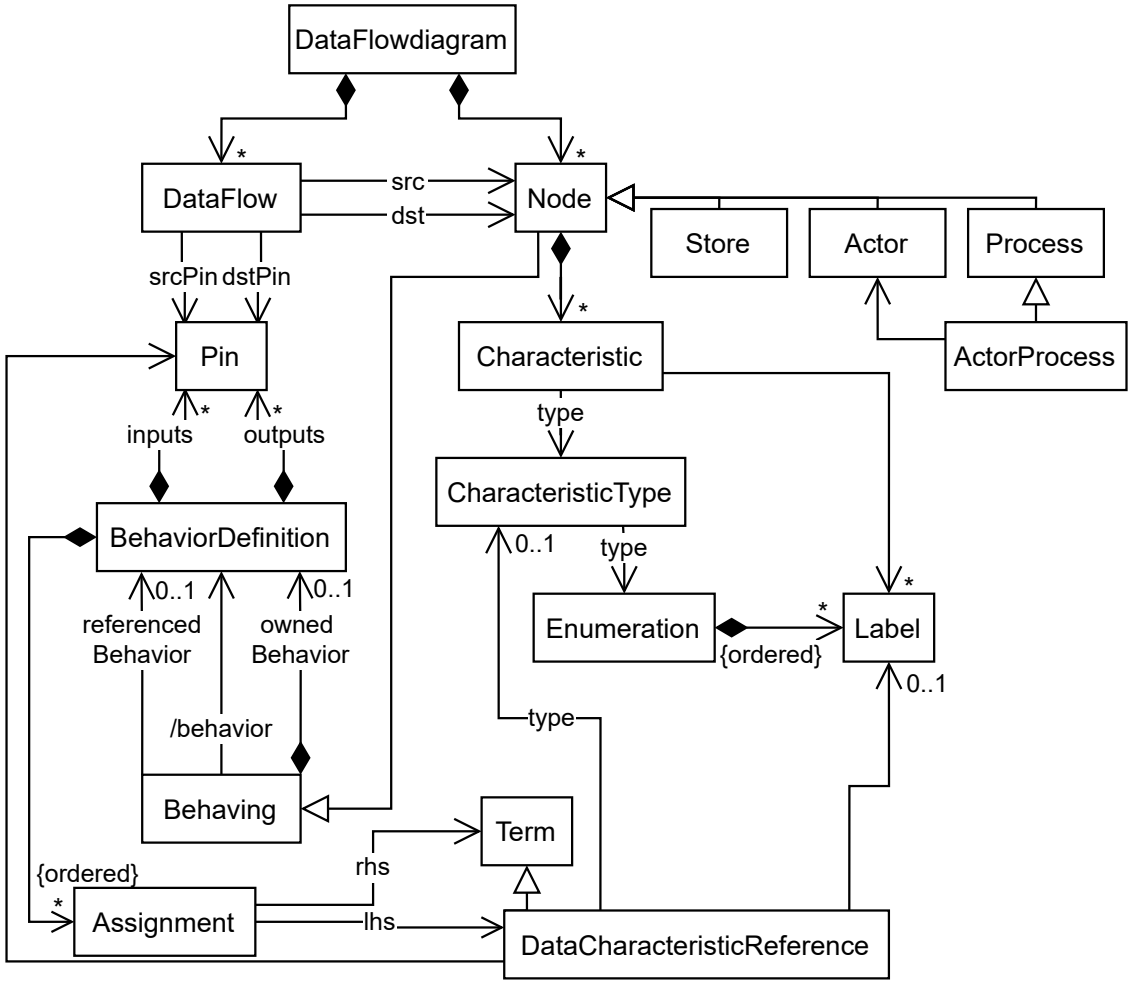


Figure 2.2: Metamodel of DFD with confidentiality extensions [66].

2.2.2 Transformation to Logic Program

As shown in Figure 2.2, actors, stores, processes, and actor processes are subclasses of node. Each node is transformed to one Prolog fact, which corresponds to its actual type, replacing the placeholder variable `N` in Line 1 of Listing 2.1 with the identifier of the node. For actor processes, the placeholder variable `A` is replaced by the identifier of the corresponding actor. Characteristic types are transformed by creating a fact as shown in line 2 of Listing 2.1, replacing `CT` with the identifier of the characteristic type. For each label of the enumeration, which the characteristic type refers to, a fact as shown in line 3 of Listing 2.1 is created. `CT` is replaced by the identifier of the characteristic type, `V` with the identifier of the actual label, and `I` with the position of the label in the enumeration. Each label that is assigned to a node becomes one fact by replacing the `N` in line 4 of Listing

2.1 with the identifier of the node, CT with the identifier of the characteristic type and V with the identifier of the assigned label.

Listing 2.1: Prolog facts representing the DFD structure [66].

```
1 actor(N), store(N), process(N), actorProcess(N, A),
2 characteristicType(CT),
3 characteristicTypeValue(CT, V, I),
4 nodeCharacteristic(N, CT, V).
```

Facts are created for every input and output pin of each node, by replacing N in line 1 of Listing 2.2 with the identifier of the corresponding node and PIN with the identifier of the pin. Data flows are transformed to facts, shown line 2 of Listing 2.2. F is replaced by the identifier of the data flow, N_SRC and PIN_SRC are replaced by the identifiers of the source node and pin, N_DST and PIN_DST are replaced by the identifiers of the destination node and pin. As behavior is specified for a pin of a node, we replace the variables N and PIN, of the rule shown in line 3 of Listing 2.2, with the identifiers of the pin and corresponding node. The rule body is created in a way, that it evaluates to true, if the *value* label V of characteristic type CT is available on the pin PIN of node N. S represents a flow tree, that will be traversed, VF represents a list of all flows that have already been traversed. [66]

Listing 2.2: Prolog facts and rule representing node behaviors [66].

```
1 inputPin(N, PIN), outputPin(N, PIN),
2 dataflow(F, N_SRC, PIN_SRC, N_DST, PIN_DST),
3 characteristic(N, PIN, CT, V, S, VF) :- ...
```

2.2.3 Analysis

The analysis is defined with a query to the Prolog program. Prolog automatically tries to solve the query and by doing so considers all data paths via backtracking. For access control, the queries compare labels of the nodes receiving data with labels of the received data, or labels of data to each other.
The Prolog clauses that can be used to define analysis queries are the union of the Prolog facts and rules described in the previous section and additional helper clauses. Line 1 of Listing 2.1 shows the clauses to find an identifier N representing one of the node types described in Section 2.2.1. Lines 2 and 3 of show clauses to find the identifier CT of a characteristic type and the label identifier V of characteristic type CT. Line 4 of shows the clause to find the label identifier V of characteristic type CT that is active on the node with identifier N.
Line 1 in Listing 2.2 shows clauses to find an identifier PIN for either input or output pins of a node with identifier N. Line 2 shows the clause to find an identifier F of a data flow, that connects the pin with identifier PIN_SRC of the node with identifier N_SRC to the pin with identifier PIN_DST of the node with identifier N_DST. Line 3 Listing 2.2 shows the clause to find the label identifier V of characteristic type CT that is available on the pin with identifier PIN of the node with identifier N.

Line 1 in Listing 2.3 shows the clause to find all input pins `PINS` for a given node with identifier `N`. Line 2 shows the clause to build a valid flow tree `S` for the pin with identifier `PIN` of the node with identifier `N`. Line 3 shows the clause to find out if the node with identifier `N` has been traversed by the flow tree `S`.
Using these clauses different queries can be tailored to fit different policy types.

Listing 2.3: Helper clauses for analysis queries [66].

```
1 findAllInputPins(N, PINS),
2 flowTree(N, PIN, S),
3 traversedNode(S, N).
```

2.3 Fuzzy Inference Systems

Fuzzy Inference Systems (FIS) use fuzzy set theory to map system inputs to outputs. The Mamdani fuzzy inference system [43] that we use in this thesis was initially developed to control the steam engine and boiler combination. FISs are used for several use cases, including handwriting recognition, prediction systems for early recognition of earthquakes [5] and as adaptive-network-based fuzzy inference systems (ANFIS) [35] in the field of artificial neural networks and machine learning [68], [74].
A FIS is made up of four main components [14], [39], [50], shown in Figure 2.3:

- A fuzzifier, which translates the crisp inputs to the system into fuzzy values, by applying the real-valued crisp input values to a set of membership functions.
- A fuzzy inference engine, which uses the fuzzy input from the fuzzifier and available fuzzy rules to infer a fuzzy output.
- A defuzzifier, which translates the fuzzy output of the inference engine back to a crisp value, by aggregating the fuzzy outputs using a defuzzification method.
- A knowledge base, which comprises a collection of fuzzy rules, the rule base, and a collection of membership functions, the database.

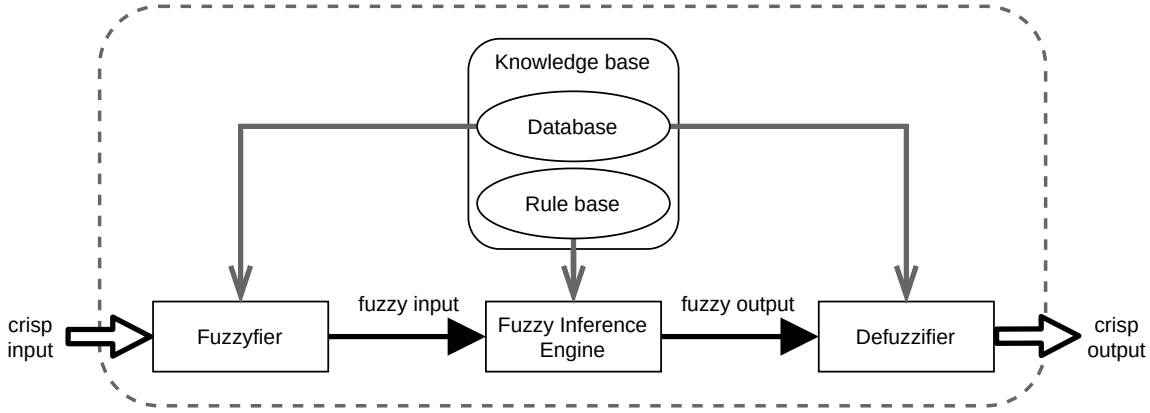


Figure 2.3: Basic structure of a FIS [50].

In the fuzzifier, the crisp numerical inputs to the system are assigned to fuzzy sets. This assignment is represented by a degree of membership within an interval of [0,1]. The degree of membership value represents the degree of truth that the input value belongs to the fuzzy set. These fuzzy sets are defined by their respective membership function, which calculates the degree of membership for a given value.
Referencing natural language, each input represents a linguistic concept used to abstract from the actual crisp inputs. The membership functions represent the linguistic values of the corresponding concept. For example, rainfall might be measured by the number of liters per square meter. This measurement is the crisp input. A linguistic concept might be the *Heaviness of Rain* [46]. Figure 2.4 shows the exemplary fuzzy input for the heaviness of rain, depending on the amount of rainfall in liters per square meter. The linguistic values might be *lite*, *medium*, or *heavy* rain. Three corresponding fuzzy sets are defined by the membership functions `lite`, `medium` and `heavy`. A crisp value of 7.5 liters of rain, results in degrees of truth of 0.2 for *lite*, 0.55 for *medium* and 0.0 for *heavy*.

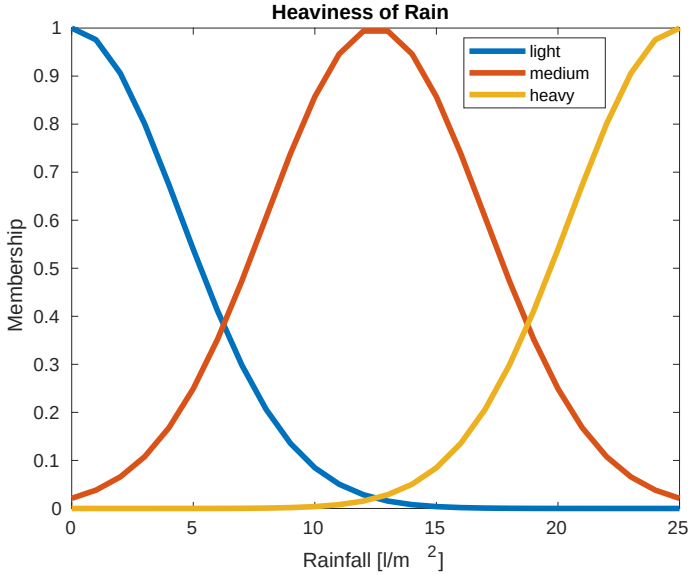


Figure 2.4: Linguistic concept 'heaviness of rain', with exemplary linguistic values.

While the fuzzy sets in Figure 2.4 are defined by gaussian functions, in general any fuzzy set A can be defined by a function μ_A over a discrete universe X with the mapping rule $\mu_A : X \rightarrow [0, 1]$ [76].
Fuzzy rules which characterize the behavior of the fuzzy inference engine, are a set of linguistic description rules. The rules are structured in the form **IF** (a set of conditions are satisfied) **THEN** (a set of consequences can be inferred). For a FIS with two inputs and one output a rule is structured like this:

$$\textbf{IF } a \textit{ is } A_i \textit{ and } b \textit{ is } B_i \textbf{ THEN } z \textit{ is } Z_i.$$

Here a and b are inputs structured similar to the example input in Figure 2.4, A_i, B_i and Z_i are fuzzy sets or membership functions, from each respective input/output. This rule is

implemented by a fuzzy implication R_i, which is defined as

$$R_i(u, v, w) = [A_i(u) \; and \; B_i(v)] \rightarrow Z_i(w).$$

The logical connection *and*, and corresponding *or*, can be implemented by a variety of functions. The Fuzzy Logic Toolbox in MATLAB [44] and the ISO IEC 61131 standard regarding fuzzy control programming [34] define the min operator of Mamdani (MIN), the product (PROD), and the bounded difference (BDIF) for the logical *and*. For the logical *or*, the max operator of Mamdani (MAX), the algebraic sum (ASUM), and the bounded sum (BSUM) are defined. The corresponding formulas are presented in Table 2.1. For Mamdani FISs, the default implementation is the MIN operator $\wedge$. This results in

$$[A_i(u) \wedge B_i(v)] \rightarrow Z_i(w) = min\{A_i(u), B_i(v)\} \rightarrow Z_i(w).$$

And Operator		Or Operator	
MIN	$min(n, m)$	MAX	$max(n, m)$
PROD	$n \cdot m$	ASUM	$n + m - n \cdot m$
BDIF	$max(0, n + m - 1)$	BSUM	$min(1, n + m)$

Table 2.1: Operators that can be used for the logical connection in rules. [34]

Multiple rules of this kind might be defined. To obtain the fuzzy output, the fuzzy inference engine utilizes the given fuzzy inputs a_0 and b_0 to calculate the consequent fuzzy output Z. $Z(w)$ is calculated by accumulating the result of the fuzzy implications of each rule:

$$Z(w) = Accu(A_1(a_0) \wedge B_1(b_0) \rightarrow Z_1(w), ..., A_n(a_0) \wedge B_n(b_0) \rightarrow Z_n(w)) \quad \forall w \in W$$

Similar to logical connections *and* and *or*, MATLAB [44] and the ISO IEC 61131 standard [34] define the max operator of Mamdani (MAX), the bounded sum (BSUM) and the normalized sum (NSUM) as operators for *Accu*. The corresponding formulas are presented in Table 2.2. For Mamdani FIS *Accu* is defined as the MAX operator $\vee$.

$$Z(w) = Z_1(w) \vee ... \vee Z_n(w) = max\{Z_1(w), ..., Z_n(w)\}$$

Accu Operator		
Mamdani max	MAX	$max(n, m)$
bounded sum	BSUM	$min(1, n + m)$
normalized sum	NSUM	$\frac{n+m}{max(1, MAX(n'+m'))}$

Table 2.2: Accumulation methods. [34]

In the defuzzifier, the calculated fuzzy output Z is translated back to a crisp value z_0. This is done by utilizing a defuzzification operator: $z_0 = defuzzifier(Z)$. Fuller et al. [24]

define the center of gravity (COG), center of area (COA), left most max (LM) and right most max (RM) as defuzzification operators. The corresponding formulas are presented in Table 2.3.

Defuzzifiaction Method		
center of gravity	COG	$\frac{\int_W xZ(x)\,dx}{\int_W Z(x)\,dx}$
center of area	COA	$\frac{\sum_j x_j Z(x_j)}{\sum_j Z(x_j)}$
left most max	LM	$min\{x \mid Z(x) = max_w(Z(w))\}$
right most max	RM	$max\{x \mid Z(x) = max_w(Z(w))\}$

Table 2.3: Calculation rules for the defuzzification process. [24]

Figure 2.5 shows the process of making inferences with a multiple-input-single-output (MISO) FIS, in this case two-input-single-output. The shown example uses the Mamdani min and max operators for the logical *and* connection and aggregation. For defuzzification, the described COG operator is used. x_0, y_0 represent crisp input values, z_0 represents the crisp output value.

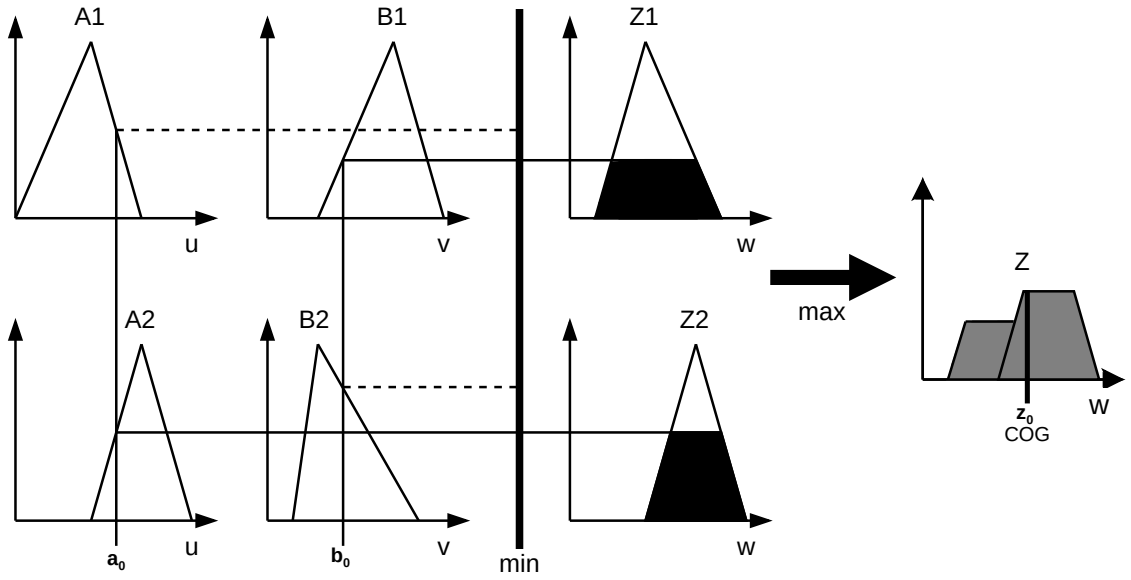


Figure 2.5: Inferences with Mamdani's min, max operators and GOG defuzzification [24].

3 Related Work

In this chapter, we give an overview of the related work of this thesis. We split the related work into two areas of research. In Section 3.1, we discuss different areas where uncertainty has been researched and distinguish our contribution from approaches in these areas. We also discuss related work regarding the design-time analysis of software architectures, describe different existing approaches, and differentiate our contribution of this thesis from these approaches in Section 3.2.

3.1 Uncertainty

The related work presented in this section focuses on uncertainty in various research areas that are partly related to the contribution of this thesis. In Section 3.1.1 we present related work in the area of design-time uncertainty. Section 3.1.2 gives an overview of multiple approaches that address uncertainty in access control. In Section 3.1.3 we present three related publications in the topic of uncertainty and cyber-physical systems. Finally, in Section 3.1.4 we present related work that uses fuzzy systems to handle or represent uncertainty in the context of access control.

3.1.1 Uncertainty During Design-Time

Lytra and Zdun [41] propose a fuzzy logic-based approach for architectural design decision making to provide automated guidance for recurring architectural design decisions under uncertainty. They create a DSL for specifying decision models with uncertainty and implement a fuzzy inference system for deriving best-fitting design solutions.
Famelis and Chechik [21] propose a methodological approach for managing uncertainty using partial models. They work out stages in the lifecycle of uncertainty-related design decisions and encapsulate abstracted information of the lifecycle of design-time uncertainty in software artifacts in the so-called Design-Time Uncertainty Management (DeTUM) model.
Both approaches provide a good base understanding of uncertainty during design-time but primarily focus on uncertainty in the structure and input of the software system. The characteristic of uncertainty in the system context, as described in Section 2.1, is not regarded.

3.1.2 Uncertainty in Access Control

Hengartner and Ge [29] present an access control model that addresses challenges that arise when deploying uncertainty-aware access control. They define these challenges as

identifying and authenticating people and their intended actions, associating uncertainty with time, and handling similar/redundant uncertain statements.
Bures et al. [13] present access-control related situational patterns for tackling uncertainty in a system. They provide a classification and representative examples of uncertainty in access control in Industry 4.0 systems. the situational patterns are based on the classification and are aimed to serve as meta-adaptation strategies in unanticipated situations.
While we use many aspects of these approaches for our proposed concept, they only focus on representing uncertainty present at runtime.

3.1.3 Uncertainty Associated with Cyber-Physical Systems

There are multiple publications of Zhang et al. that revolve around the topic of uncertainty and cyber-physical systems (CPS). The first publication [79] defines the uncertainty which is inherent in situations where CPS might interact with environmental factors like humans. Other publications focus on testing and test generation [77][78], which leverage the uncertainty of its operating environment to ensure its reliable operation.
While CPS generally fits in the theme of I4.0, the uncertainty involving the environment and operation of CPS shows no direct relation to the theme of uncertainty in access control of I4.0 systems.

3.1.4 Using Fuzzy Logic

Several fuzzy logic approaches describe how to handle uncertainty. In the realm of access control, we found an approach of Hosmer [30] that uses fuzzy logic to represent security patterns. Cheng et al. [15] use the inherent uncertainty and risk in access control decisions to create a fuzzy logic risk-adaptive access control model. They illustrate their approach by showing how it could be used to implement a fuzzy multi-level-security model.
The work of Platenius et al. [52], [53] uses fuzzy logic to match service requests to potentially incomplete service specifications using signature or privacy policy matching.
Similar to the approach of Lytra and Zdun [41] the fuzzy logic approaches focus on making decisions by factoring in uncertainty. This thesis, however, mainly focuses on representing uncertainty in the software architecture and providing a process to analyze the effect of uncertainty on access control during design time.

3.2 Analyses of Software Architecture and Security

The related work presented in this section focuses on approaches of analyzing software architecture or security analysis which are partly related to our contribution. We first describe the Palladio approach of software architecture modeling and simulation in Section 3.2.1. In Section 3.2.2 we present an approach of model driven security analysis. Section 3.2.3 gives an overview of several data flow modeling and analysis approaches. In Section 3.2.4 we present two analysis approaches which base on source code. We conclude this section in Section 3.2.4 with a comparison of all the presented approaches and compare them to our contribution.

3.2.1 Palladio

Palladio is a tool-supported software architecture simulation approach used to predict an architecture's Quality of Software properties, like performance or reliability.
The Palladio Component Model (PCM) is a metamodel of component-based software architectures [59]. The basic PCM consists of a repository, system, resource environment, allocation, and usage metamodel, each representing a different architectural view on a system.
The repository model describes components and their interfaces, with the inner behavior of components being described by service effect specifications (SEFF). The system model combines components that are described in the repository model to specify the software architecture. The resource environment model describes the specifications of available processing resources. The allocation model describes which components are deployed on which resource. The usage model describes the behavior of users interacting with the system [59].
There are multiple approaches to analyzing PCM model instances. The most prominent is the method of simulation for performance predictions during the design process [7]. Another analysis method has been presented by Koziolek et al. [40], transforming the PCM to layered queueing networks (LQN) and using an LQN solver. This approach gives a performance advantage over the previously described discrete-time simulations, but results need to be interpreted manually. A different simulation approach has been proposed by Meier et al. [47]. The approach provides an automated model-to-model transformation that transforms PCM model instances to queuing petri nets (QPN). The modeled system is then analyzed by simulating the resulting QPN.
All of the before mentioned approaches based on the Palladio Component Model (PCM) focus on performance prediction during design time and not access control.
In previous work, Seifermann et al. propose a software architecture description and analysis process, called Data-Driven Software Architecture (DDSA) [65]. The approach extends the PCM and utilizes the control flow derived from a PCM instance for access control analysis. They introduce the concept of data and data processing operators as first-class entities to the PCM. Data can have sets of characteristics, which represent abstract meta-data of the affiliated data [64]. The analysis is realized as queries to a Prolog program that a transformation chain derives from the extended PCM instances. While the DDSA approach can analyze a software architecture model instance for access control violations, the focus of the access control analysis is on the control flow and not the more fine granular flow of data.

3.2.2 Model-Driven Security Modeling

Model-driven security approaches try to solve security questions during design time. Similar to the Palladio approach described in Section 3.2.1, modeling and analysis of critical security aspects in the early stages of the system development process can prevent more cost-intensive changes in subsequent stages.
Jürjens et al. describe an approach which is based on Unified Modeling Language (UML), called UMLSec [36]. With the use of stereotypes and constraints, standard UML diagrams

are annotated with security attributes. Model instances with those security attributes are transformed into abstract state machines and analyzed with a model checking approach. The approach supports analyses of information flow and access control in the form of RBAC policies. While the proposed information flow analysis considers data, the RBAC access control analysis only considers actions.

3.2.3 Data Flow Modeling

Additional to the data flow diagram approach of Seiferman et al. [66], which we have described in Section 2.2, there are several other data flow oriented security analysis approaches.
Katkalov et al. propose an environment for modeling flow-sensitive applications, called IFlow [38]. Similar to the UMLSec approach, an annotated UML diagram is transformed into an abstract state machine. The resulting machine is then used as an input for theorem proving. The goals are automatically generated based on annotations by the user. To realize an information flow analysis, IFlow requires behavior descriptions, which are very detailed and are challenging to properly create during design time [66], [73].
Peldszus et al. present an approach using an annotated architectural level data flow diagram model, called SecDFD [49]. They propose a mapping between DFD and a codebase, which includes the matching of structures and signatures. The mapping can be used to apply security metrics as a form of compliance check.
The FlowUML approach of Alghathbar et al. [1] is a "logic-based system to validate information flow policies at the requirements specification phase of UML based designs [1]." The approach derives DFDs from UML sequence diagrams and maps them in a logic program. They describe how to create policies that can detect violations in information flow and access control. However, the access control policies are limited to the Discretionary Access Control (DAC) and Mandatory Access Control (MAC) models. Also, at the time of writing this thesis, there exist no publications that report on an evaluation of FlowUML.

3.2.4 Source Code Confidentiality Analysis

When focusing on security analysis, there also are multiple approaches revolving around the topic of source code confidentiality analysis. As the field is very big, we only present two different approaches as our related work.
The generic information flow analysis JOANA of Snelting et al. [72] supports flagging of data and the use of multiple security levels.
More specified approaches, like the FlowDroid approach of Arzt, Steven, et al. present a static source code analysis for Android applications [4]. They utilize the technique of 'tainting' data to track sensitive information through an application.
While both approaches can detect violations in information flows, they can only be used in a later development phase when source code is already available. At this time in the development process, the approaches can detect the effect of a design issue, but do not reveal the design issue itself [66].

3.2.5 Comparison

We add Table 3.1 to better summarize the comparison and describe our reasoning in using the DFD approach of Seifermann et al. (see Section 2.2) as the base for our contribution. The Palladio and DDSA approaches only focus on the control flow. Even though DDSA allows for access control, it is not defined on the data level. The IFlow and SecDFD approaches focus on information flow but do not provide the possibility of defining access control policies. While the UMLsec and FlowUML approaches generally allow for the definition of access control policies, they come with many restrictions and are not universally applicable. The approaches regarding source code analysis fall out of the scope for our contribution, as we aim to focus on the design time software architecture.
The DFD approach of Seifermann et al. (see Section 2.2), offers access control analysis on the data level, during design time. The definition of access control policies is also not restricted, like the UMLsec and FlowUML approaches, and is able to define access control policies for all commonly known access control models.

Approach	Development phase	Scope	Analysis method
Palladio	design time	control flow	analytical solver & simulation
DDSA	design time	control flow & access control	logic program
UMLsec	design time	information flow (& access control)	model checking
IFlow	design time	information flow	theorem proving
SecDFD	design time	information flow	theorem proving
FlowUML	design time	information flow (& access control)	logic program
FlowDroid	implementation	information flow	static taint analysis
JOANA	implementation	information flow	static code analysis
Contribution (based on 2.2)	design time	information flow & access control	logic program

Table 3.1: Comparison of related work regarding design time analysis of software architecture.

4 Characterizing Uncertainty

In this chapter, we discuss how uncertainty can be characterized. In Section 4.1 we narrow down an existing characterization of uncertainty to fit the area of software architecture. We further narrow down our characterization with regards to access control in Section 4.2 by conducting a preliminary literature research regarding access control properties and discussing how some publications already consider uncertainty in different ways. Based on our research, we define our concept of trust in the validity of an access control property to handle the observed uncertainty in Section 4.3.

4.1 Uncertainty in Software Architectures

To characterize and discuss the uncertainty in access control on the architectural layer, we first need to establish a general characterization of uncertainty in software architecture modeling and software architecture analysis. We base our characterization on the classification of Perez-Palacin, and Mirandola [51], described in Section 2.1.
To narrow down this classification and characterize uncertainty in software architecture, we first analyze if the characteristics are applicable in the context of software architecture:

Level: *Level 0 uncertainty*, or *lack of uncertainty* establishes a theoretical base for the level characteristic of uncertainties and exists in every system. A system that only has level 0 uncertainties is equivalent to a deterministic system without uncertainties. Any software architecture analysis approach can already cover this kind of uncertainty if no uncertainty exists in the system.
Level 1 uncertainty, or *lack of knowledge* is uncertainty that one is aware of but is not resolvable to level 0 due to missing knowledge. Considering software architecture, if there is awareness about uncertainty, it might be possible to gather the information related to the uncertainty and add it to the software architecture. A software architecture analysis process can use this added information to gain knowledge and mitigate the uncertainty. However, during design time, the software architecture might still be made up of assumptions, approximations, or averages which are uncertainties in themselves. Additionally, uncertainties that one is not aware of might still be present, so fully reducing the uncertainty to level 0 is not universally possible.
To consider and consequently handle uncertainty in an analysis on the architectural level, it needs to be treated as a first-class entity and explicitly represented. To have a representation of something in a model, one needs to be aware of its existence. Uncertainties on levels 2, *lack of awareness*, and 3, *lack of process*, exist in a system because there is no awareness about the uncertainty or no known process to gain awareness of that uncertainty. This means that uncertainties of levels 2 and 3 can theoretically always be present in software architecture.

Location: The taxonomy of Perez-Palacin and Mirandola [51] is already focused on the modeling of software systems, so the locations of the taxonomy can directly be applied to software architecture. *Context uncertainty* concerns the completeness of the metamodel that is used to create a software architecture with respect to the real world. *Model structural uncertainty* concerns how accurately the structure of the software architecture represents the modeled subset of the real world. *Input parameter uncertainty* is associated with the uncertainty about the actual value of variables given as input to the software architecture.

Nature: Both natures, epistemic and aleatory, can be present in software architecture. However, *aleatory uncertainty*, due to the inherent variability of parts of the system or random events, can not be reduced with an analysis approach, as this kind of randomness can always occur. *Epistemic uncertainty* can be reduced by collecting more precise data about the system and analyzing the improved software architecture.

4.2 Uncertainty in Access Control

To further narrow down the kinds of uncertainty relevant to an architectural access control analysis for Industry 4.0, we aggregate relevant access control properties from the literature. We utilize these properties to further narrow down the uncertainty characteristics that we focus on in this thesis.

4.2.1 Access Control Properties for Industry 4.0

When observing an I4.0 software system and its environment, many properties could be considered for access control. To get an overview about which access control properties might be used and in how far they can experience uncertainty, we conduct preliminary literature research to find access control properties that are well documented or researched. Our research suggested that the subject area of access control is extensive while at the same time yielding only minimal information regarding actual access control properties, especially with a focus on industrial applications like I4.0 or IIoT systems. This is why we decided to widen our view and focus on well-documented access control properties in general.
From preliminary work [10] we already know the concepts of role-based access control (RBAC) and Organization Based Access Control (OrBAC). We also know that location information can be used to express access control rules or requirements. Using this knowledge as our base, we explored research regarding these three main types of properties. The characteristic of a property can differ within the same property type. For example, the location can specify the position within the perimeter of a production plant or the country/city geolocation. The location information can originate from various sources, like GPS-Sensors, but can also be derived from IP-address resolution. Within a chosen access control model, similar properties can also be represented in entirely different ways. We chose the presented publications because they directly focus on the access control property and not only the access control model. Many of the presented publications go into detail on how a property is obtained and what characteristics make up the property. Some publications go as far and to describe a kind of uncertainty that can coincide with

the characteristics of the presented access control property. We explicitly do not list the property of current time and attempted action, as we did not find sufficient publications specifically addressing these properties or the uncertainty that might come with them.
We categorize the identified access control properties by the information they are based on. Properties that we can identify are:

4.2.1.1 (Geo-)Location

Ardagna et al. [2] define an approach to Location-Based Access Control (LBAC). This access control model only uses location-based conditions and predicates to define access control policies. The model considers the limitations of technology by including confidence and timeout values of the location providing services.
Ray et al. [58] extend the Role-Based Access Control (RBAC) model to incorporate locations and show how location information can be used to determine whether a subject has access to a given object. They formalize their model using the Z specification language [71]. Damiani et al., which proposes an extension of the RBAC model, has suggested a similar approach, enhanced with spatial-and location-based information.
Skandhakumar et al. [70] propose a graph theoretic representation of building information models (BIM) that can be used to improve access control administration.
We additionally identified some more general approaches to access control that use the location as a minor matter, or in examples [16], [29], [57].

4.2.1.2 Roles

Role-based Access Control (RBAC) has been covered by multiple publications [22], [23], [63]. Permissions are associated with roles, and roles are assigned to users, thereby acquiring the roles' permissions. This reduces the complexity and cost of setting up large-scale authorization management.
As RBAC and roles are a well known concept in the field of access control, roles are often used in examples of related publications [17], [29], [32], [57], [58].

4.2.1.3 Organizational Affiliation

The Organization-based Access Control (OrBAC) model, suggested by Kalam et al. [37], extends the RBAC model by introducing the concept of organizations and contexts. Organizations are a set of subjects, e.g., users or other organizations. Contexts are used to specify the concrete circumstances or conditions in which specific permissions are granted.
The organizational affiliation is also considered in the definition of ABAC [32] and access control models for cloud computing systems [42], [62].

4.2.2 Trust in Access Control Properties

In the publication "Guide to Attribute Based Access Control (ABAC) Definition and Considerations [32]" of the National Institute of Standards and Technology (NIST), Hu et al. describe the trust chain of ABAC, regarding the attributes used to make access control decisions. Trust chains help determine the ownership of information and services and

requirements for technical solutions to validate and enforce trust relationships. The predicate of a trust relationship revolves around the idea that the access control system can trust the validity or correctness of the information, e.g., attributes supplied by the owner, e.g., an authorization service. Depending on the access control model, many trust relationships are required to achieve a properly working access control system. In a subsequent publication regarding ABAC, Hu et al. define levels of attribute assurance (LOAA) [31]. An LOAA combines the *Accuracy*, *Integrity* and *Availability* properties of attributes which are supplied via the described trust chains to give an approximation about the assurance that the access control system has a valid attribute at its disposal.
Considering RBAC, there might be different services that verify a particular role of a user, see Section 4.2.1.2. Depending on the kind of service, the trust in a supplied role might be higher, e.g., using face recognition, or lower, e.g., using magnet cards. This is similar to the confidence value of different location services described in [2]. A confidence value that a location service can guarantee is composed by combining accuracy, environmental and weather conditions, requested location, and measurement technique.
This trust in attributes is crucial for Industry 4.0 software systems, as these systems are often comprised of many subsystems and might even span multiple organizations, as we described in the introduction 1. If, for example, a technician of another organization is called in, the trust in the role of technician needs to be deliberated, as the owner of that information is the other organization.

4.2.3 Environmental Factors

Interpreting the research regarding access control properties we have already done in Section 4.2.1, we assume that uncertainty of the environment of the system has a significant impact on access control. For example, the confidence value defined by Ardagna et al. [2] is dependent on the service and environmental conditions. The confidence can be considered as the certainty in the location's validity, which depends on environmental factors. The spatial context definition of Cuppens et al. [16] precisely combines the hardware and software architecture and the environment as an additional condition for access control. Based on the work of Hengartner and Ge [29], and Bures et al. [13] we can identify environmental factors that influence the uncertainty in access control. These factors are generally applicable to all access control properties we have identified in Section 4.2.1. They also align well with the idea of trust relationships we discuss in Section 4.2.2 and can be represented in software architecture:

- Source of the information
- Age of the information
- Amount of redundant information

In big and distributed systems, such as I4.0 systems, multiple services might exist and use different sources to deliver similar information that the access control properties are relying on. Location information, for example, might be derived from a physical access control mechanism or GPS data. Each of these sources has a different margin of error

or accuracy. The physical access control mechanism might verify that a person is in a particular room, even though he only opened the door and never entered. The GPS, on the other hand, has varying accuracy that might be dependent on environmental influences, as we discussed in the introduction.
The age of the information is a combination of the time it took for a service to gather the information and for that information to be processed to an access control property and the overall time that has passed since this access control property has been created. With an increase in age, the uncertainty rises. For example, the processing time of a GPS sensor might increase if satellite reception is not optimal. The location of a moving person might have changed significantly at the time the processing has finished.
The amount of redundant information could either increase or decrease uncertainty. Location information from a physical access control system using magnet cards and GPS location information can either decrease the uncertainty if they complement and verify each other or increase uncertainty if the information is conflicting. Using redundant information to make access control decisions violates monotonicity [8] or opens up the possibility for Sybil attacks [19]. Monotonicity in access control ensures that, given a set of access control properties grants access, a superset of this set will not deny access [8]. This means that uncertainty can only be decreased and not increased when combining redundant information. A Sybil attack exploits monotonicity by creating a lot of similar information that complements each other. Even if each information has high uncertainty, by verifying each other, the combined uncertainty could be decreased to the point of being sufficiently low for somebody to be granted access [29].
To further narrow down our view on uncertainty and how it can impact architectural access control analysis, we need to discuss how to classify these environmental factors regarding the uncertainty taxonomy of Perez-Palacin, and Mirandola [51]. We have already narrowed down the taxonomy to be more compliant with software architecture modeling and software architecture analysis in Section 4.1. Bures et al. [13] suggest a variation on the uncertainty taxonomy of Perez-Palacin and Mirandola [51], to classify uncertainty in access control, mainly changing the definition of the location property. They define the location of *system environment* as a combination of the context in which the system is executed and the input data for the system, which fits the identified environmental factors. This definition aligns with the *context* and *input parameter* location defined by Perez-Palacin and Mirandola.
Combining the discussion about uncertainty in software architecture, in Section 4.1, with the uncertainty regarding access control, we narrow down the kind of uncertainty we aim to mitigate as a contribution of this thesis. Figure 4.1 shows a version of the original taxonomy, with our narrowed-down view of uncertainty highlighted.

4.3 Uncertainty of Trust

By researching uncertainty in software architecture and access control in this chapter, we became aware that uncertainty in the validity of access control properties exists. By gaining this awareness, we reduce the *level 2 uncertainty* related to the described validity of access control properties to an *uncertainty of level 1*. We can now address and try to

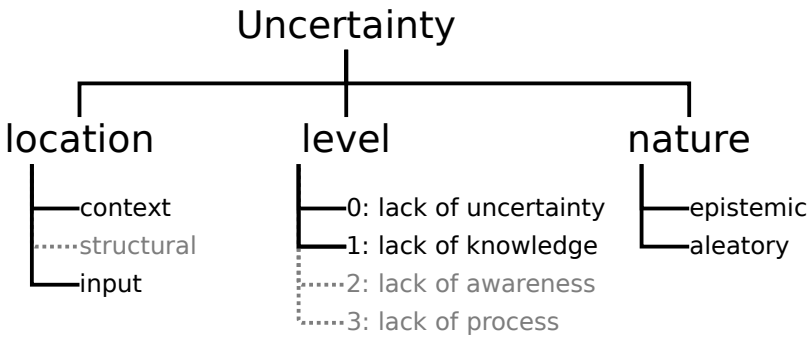


Figure 4.1: Highlighted uncertainty taxonomy.

handle this *level 1 uncertainty*, by using software architecture modeling and analysis, as described in Section 4.1.
To handle the uncertainty, we add the concept of *trust* in the validity of access control properties to an already existing software architecture model and access control analysis on the architectural level. We define *trust* as a composition of the environmental factors described in Section 4.2.3. The trust in an access control property is generally similar to the confidence level of location services described by Ardagna et al. [2]. The confidence level depends on the extent to which given environmental factors affect the validity of the individual location services. However, our concept of *trust* will be applied to all services that supply information used for access control while additionally taking the actual source and age of the information into account (see Section 4.2.3).
We extend the data flow approach described in Section 2.2, as it already offers a data flow model to represent software architecture and an automated analysis that can be used to detect access violations. By extending the DFD model syntax with the concept of *trust*, we include *context uncertainty* into the software architecture and enable a closer representation of the real world. As each of the environmental factors that influence the trust in a service needs to be represented, the number of input parameters of the model increases, consequently increasing the *input parameter uncertainty*.
We reckon that an analysis approach can more easily address *input parameter uncertainty* by executing multiple analysis runs with varying input parameters. This procedure simultaneously enables the handling of *epistemic uncertainty*, as each analysis run results in data that can be used to get better knowledge about the software architecture, as well as handling the described *level 1 uncertainty*.

5 Running Example

This chapter introduces an example scenario, which we use to demonstrate our contribution regarding an extended software architecture model and mapping to a logic program.
The example is based on the inaccuracy of using GPS for location detection, depending on various environmental factors. For this example, we additionally focus on the reduced signal quality and increased acquisition time of indoor operation [20].
The scenario is comprised of three actors:

- Visitor, an actor that does not belong to the company but has a meeting in the meeting room.
- Worker, an actor working for the company, currently in the meeting room waiting for the Visitor.
- Scientist, an actor that works in the laboratory and has exclusive access to the room.

The location of each actor is monitored using a GPS location service. The location services of the Worker and Scientist use a high-sensitivity GPS sensor, while the service of the Visitor uses a standard GPS sensor. The blue circles, shown in Figure 5.1, represent the position accuracy of these sensors.

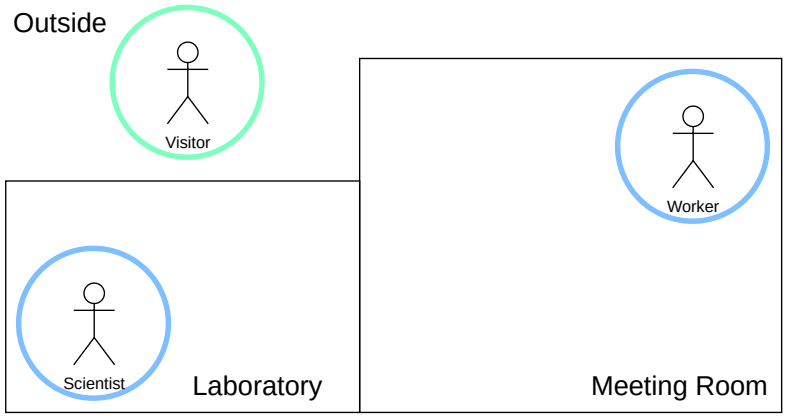


Figure 5.1: Initial setup of the running example.

In the scenario, a database exists, which is comprised of sensitive research data from the Scientist. Read and write access to the database is only permitted if the accessing user is located inside the laboratory. We define a data flow diagram of the scenario, shown in Figure 5.2. The data flow from the Scientist to the *Write DB Process* and from the *Write DB Process* to the *Laboratory DB* shows the sensitive research data, which is stored in the

database. Regularly, only data flows are allowed to the Scientist, as he has exclusive access to the laboratory. Both data flows from the *Read DB Process* to the Worker and Visitor, which are normally prohibited, are highlighted in red. These prohibited data flows could be found with an access control analysis of this data flow diagram.

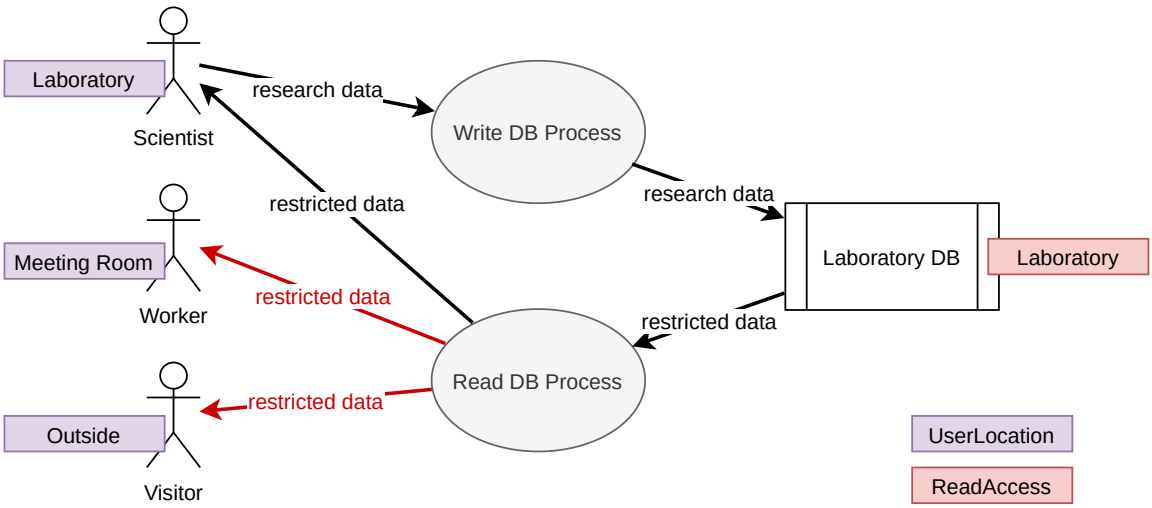


Figure 5.2: Data flow diagram of the initial setup of the running example.

The Visitor now enters the meeting room to meet with the Worker. This reduces the position accuracy and signal quality and increases the acquisition time of the standard GPS sensor used by the Visitor's location service. As shown in Figure 5.3, this increases the overall location deviation and introduces uncertainty about the actual location of the Visitor.

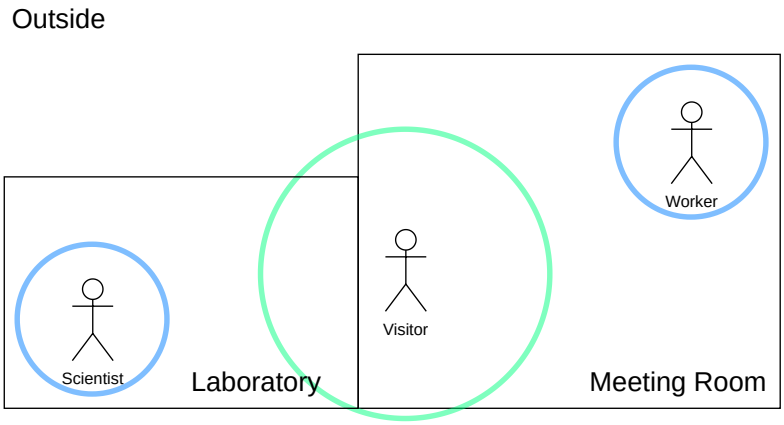


Figure 5.3: Progressed setup, with uncertainty about Visitor location.

As the exact location is uncertain, the location service of the Visitor might report that the Visitor is located in the laboratory. This results in the data flow diagram shown in Figure 5.4, where a normally prohibited data flow can not be identified as such by an access control analysis.

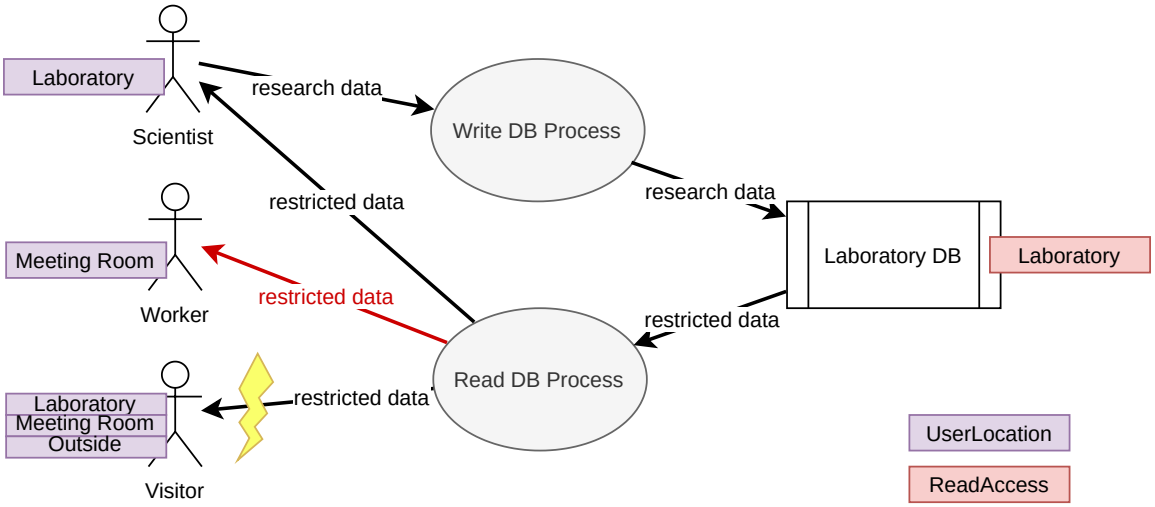


Figure 5.4: Data flow diagram, with unidentified illegal data flow to Visitor.

By adding our concept of trust, we combine the environmental factors of the location service. Due to the reduced position accuracy and signal quality of the location service, we can calculate that the trust in the correctness of the location information of the Visitor is *low*. We also add a trust requirement to the read access location, which is required to access the laboratory database. This results in the data flow diagram shown in Figure 5.5, where the illegal data flow to the Visitor can be correctly identified again.

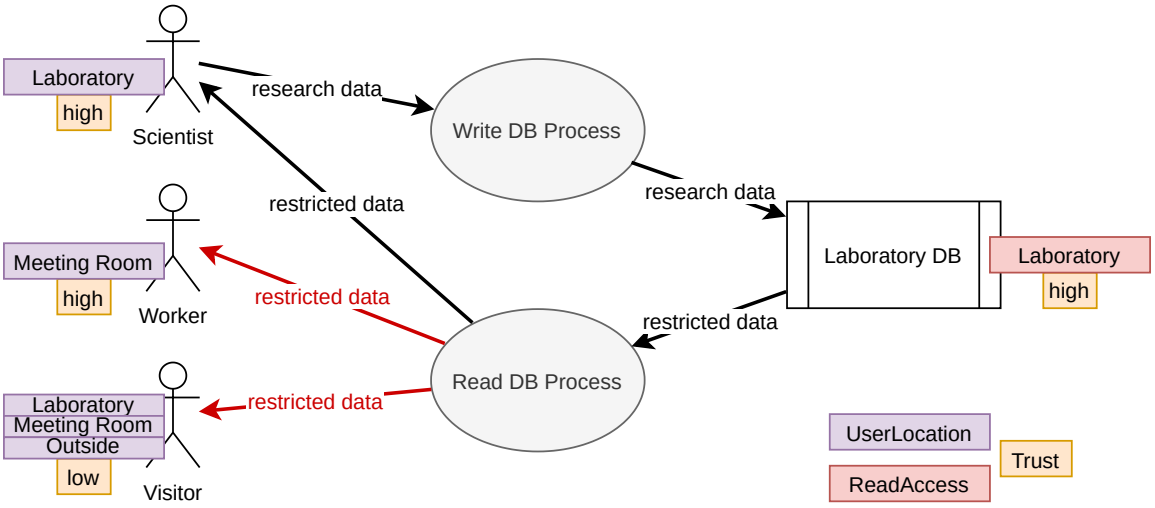


Figure 5.5: Data flow diagram, with added trust.

6 Metamodel Extension

In this chapter, we present the metamodel representation of the concept of trust we have defined in Chapter 4. We first give a very detailed description of our model representation of fuzzy inference systems (FIS) as means to combine environmental factors of the access control properties into a trust value in Section 6.1. In Section 6.2 we discuss the possibilities of how to represent trust in a model and explain our reasoning behind the representation we chose as part of our contribution. Finally, Section 6.3 describes how we extend the DFD metamodel to cover everything described in the previous sections. We go into detail on how trust is calculated, how we extend characteristics, and how existing reusable behavior definitions are modified to consider our concept of trust.

6.1 Fuzzy Inference System Modeling

We define our concept of *trust* as a composition of environmental factors. Consequently, we need to define a way to combine values of environmental factors to form a single value that represents trust.
We propose the use of FISs to provide a way of combining the environmental factors. Our proposed FIS first fuzzifies the crisp input values of the environmental factors by mapping values to existing linguistic values. An example is the way we mapped the crisp value of 7.5 liters of rain per *m* to the values of the linguistic concept of 'heaviness of rain' in Section 2.3, using the membership functions shown in Figure 2.4. Rules utilize these linguistic values to combine inputs and form a *trust* output distribution. The resulting distribution is then defuzzified, using one of a set of predefined defuzzification functions, to create a crisp *trust* output value. This trust value represents the trust in the correctness of the access control properties supplied from the service. We see multiple benefits from using FIS:
Linguistic concepts are used to abstract from the actual mathematical input and to enable the implementation of rules in a natural language way. This increases the understandability of defined inputs, rules, and trust output, making it easier to create new and interpret existing FISs calculation rules. When compared to a calculation rule in the form of a formula, the FIS's increased interpretability allows the calculation rule to be extended and modified more easily.
Another benefit of using linguistic concepts as input is that a software architect, creating the calculation rules, does not necessarily have to deal with the actual values of the environmental factors. The knowledge of which specific value may be 'high' or 'low', for example, can be taken over by someone else involved in the development process. The software architect only has to work with linguistic concepts he might already be familiar with. The concrete input membership functions of the linguistic values might be defined

by another role (security expert, system administrator), which is familiar with, e.g., the technical specifications of the used sensors or how physical conditions influence them. When using our proposed FIS, we can ensure that the same kinds of properties, e.g., location information from different services, work with the same defuzzifier and set of *trust* membership functions. This makes the resulting trust values of different information services comparable to each other while having different inputs and rules.
We create a metamodel representation of our proposed FIS. Creating this metamodel representation enables a software architect or security expert to define a FIS without the need to utilize tools they might not already be familiar with, like the Fuzzy Logic Toolbox of MATLAB [44].

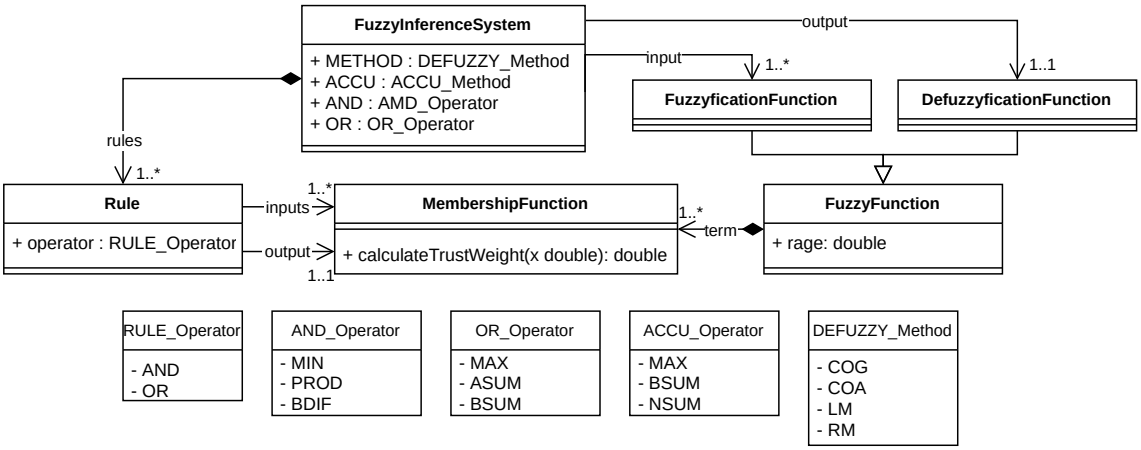


Figure 6.1: Class diagram excerpt of the FIS metamodel representation.

Figure 6.1 shows a class diagram representation of our metamodel representation. A *FuzzyInferenceSystem* class contains the fuzzifier, defuzzifier, and rules and also represents the knowledge base of the FIS, see 2.3.
The database of the FIS is split into *FuzzificationFunction*, utilized by the fuzzifier, and *DefuzzificationFunction*, utilized by the defuzzifier. The set of instances of *Fuzzification-Function* represent the inputs in the FIS, *DefuzzificationFunction* represents the outputs. As we only aim to calculate a single trust output, the amount of *DefuzzificationFunction* of each *FuzzyInferenceSystem* is limited to one. Both *FuzzificationFunction* and *Defuzzifica-tionFunction* are derived from *FuzzyFunction*. *FuzzyFunction* is an abstract representation of a linguistic concept used by the FIS as input or output. The *FuzzyFunction* defines a range and contains a set of *MembershipFunction*. As the naming suggests, *MembershipFunction* represent fuzzy sets of the linguistic values.

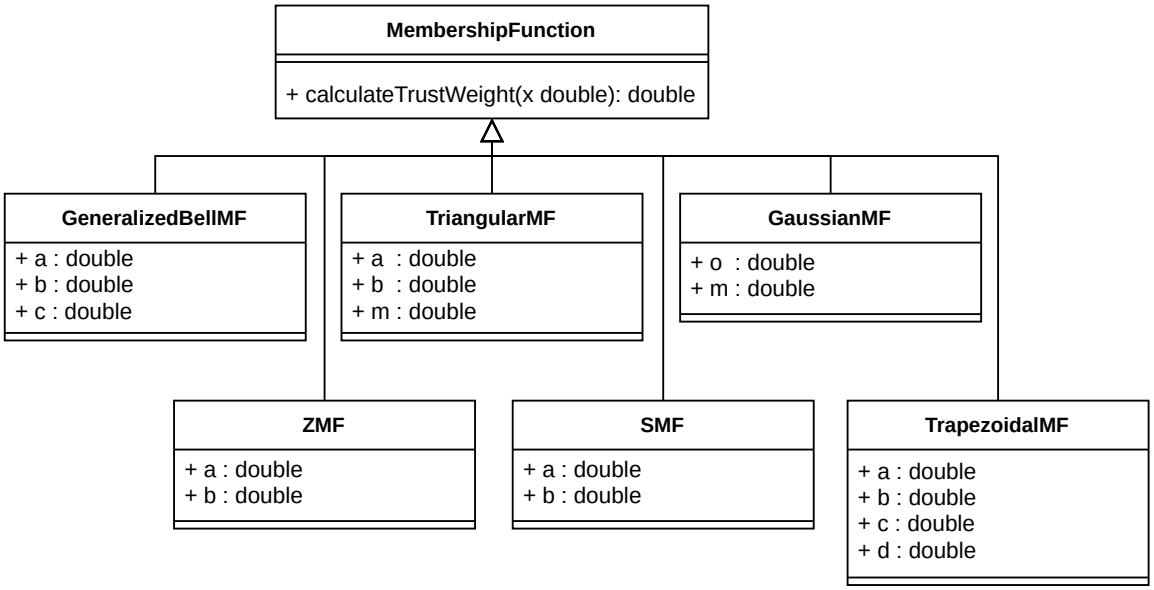
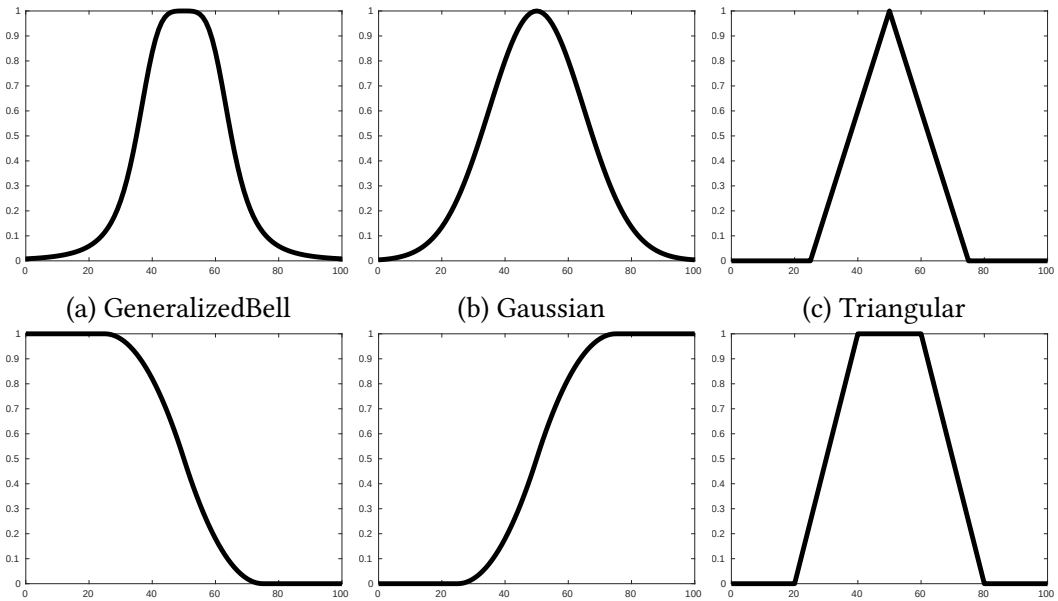


Figure 6.2: Class diagram excerpt of the membership functions of the FIS metamodel.

Following the Fuzzy Logic Toolbox [44], we implement six predefined kinds of membership functions, shown in Figure 6.3. Each membership function offers a different set of parameters that influence the general shape of the function. Exemplary instances to showcase the overall shape of the membership functions are shown in Figure 6.3. As shown in Figure 6.3, each *MembershipFunction* implements *calculateTrustWeight(x)*, which calculates the trust weight of x, with regards to the instance and parameters of the membership function.

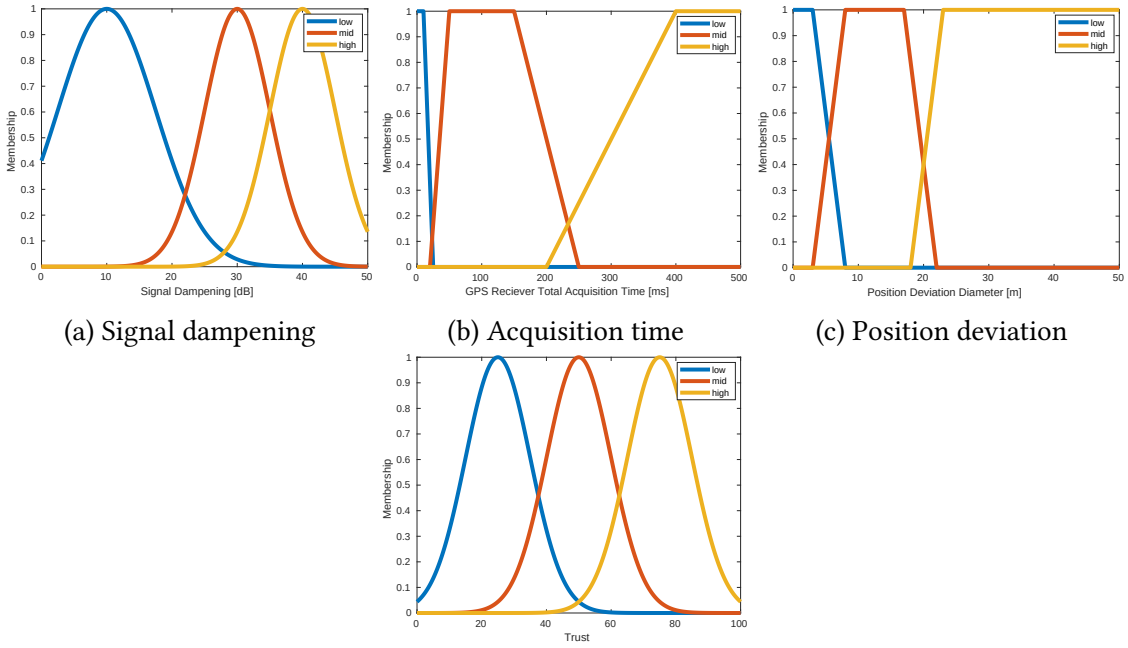


(a) GeneralizedBell (b) Gaussian (c) Triangular

(d) ZMF (e) SMF (f) Trapezoidal

Figure 6.3: Examples of membership functions.

Figure 6.4, shows the environmental factor inputs and trust output we have defined for our running example. Figures 6.4a, 6.4b, 6.4c show the inputs for the three environmental factors of a GPS location service. As described in Chapter 5, we derive the shape and values of the membership functions and corresponding fuzzy sets from the sample data we could extract from existing literature [20]. For the trust output shown in Figure 6.4d, we simply defined three equal gaussian membership functions which are an equal distance apart.



(a) Signal dampening (b) Acquisition time (c) Position deviation

(d) Trust

Figure 6.4: Membership functions of inputs and output of the running example.

The rule base of the FIS is made up of a set of *Rule*. Each *Rule* references a set if *MembershipFunction* as inputs which are connected logically, with a *RULE_Operator*. The *RULE_Operators* are either the logical *AND* or *OR* operator.
The *FuzzyInferenceSystem* contains multiple enumeration values to instantiate the actual operators used for the mapping of the input to output values. All of the possible operators and calculation rules we represent in our model are based on the settings which are possible in Fuzzy Logic Toolbox in MATLAB [44] and the ISO IEC 61131 standard regarding fuzzy control programming [34].
The *DEFUZZY_Method* specifies the calculation rule used in the defuzzification process. We support the center of gravity (COG), center of area (COA), left most max (LM), and right most max (RM). The corresponding formulas are presented in Table 2.3.
The *ACCU_Operator* defines the methods used for accumulating the results of each rule to the fuzzy output. We support the max operator of Mamdani (MAX), the bounded sum (BSUM), and the normalized sum (NSUM). The corresponding formulas are presented in Table 2.2.
The *AND_Operator* and *OR_Operator* specify the operators that can be used as the im-

plementation of the logical connection in *Rule*. For the logical *AND*, we support the min operator of Mamdani (MIN), the product (PROD), and the bounded difference (BDIF). For the logical *OR*, we support the max operator of Mamdani (MAX), the algebraic sum (ASUM), and the bounded sum (BSUM). The corresponding formulas are presented in Table 2.1.

6.2 Representation of Trust

Using the FISs described in Section 6.1 to calculate the trust still leaves the question on how to represent trust in a metamodel and analysis explicitly. As we focus on Industry 4.0 software systems, the applicability to Industry 4.0 specific scenarios must be considered when choosing a representation. For an analysis approach, trust in a property of, e.g., an actor, needs to be compared to the trust in properties that make up data classifications, so the comparability of the chosen representation is crucial.
We identify three possible solutions for trust to be represented:

Crisp output value The crisp output value of the FIS can be represented using a double value. This makes it possible to compare the value to possible access control requirements very accurately. During an analysis run, e.g., using data flows, see Section 2.2, changes to the trust of a property can be made. A valid use case might be data flowing through a filter, which reduces sensor noise by a given factor, thus increasing the trust value of specific properties. These changes can be made very finely granular, depending on the context. Especially in edge cases, these fine granular changes could result in better analysis results or reveal unexpected outcomes. At the time of writing this thesis, however, we struggle to identify use cases, either from real-world industry applications or scientific research, which would benefit from these kinds of fine granular changes. Additionally, it is tough to reason why a certain part of the system might increase or decrease trust by a fixed factor. In the sensor noise use case, trust values might not just be dependent on sensor noise but rather be a combination of varying environmental factors. As there might be different services supplying similar properties, which experience different environmental factors, see Section 4.2.3, reducing sensor noise might not increase the trust value equally for each service.
For trust values to be comparable, they need to be based on the same mathematical interval or range of values. As described before, different services might supply similar properties, but the calculation of their trust value is done using different FIS. To make the resulting values comparable, each trust value must be calculated with the same possible range. When considering the crisp trust value as a percent value, this interval would be $[0, 100]$. Additionally, it would need to be assured that the output of each FIS can reach the outer limits of the interval. This is not always possible depending on the fuzzy inputs, rules, and output of a FIS. A solution would be to normalize each trust value to be in the same interval. To do so, the theoretical minimum and maximum crisp output value of each FIS would need to be known. Calculating the minimum and maximum can only be done by calculating the crisp output of the FIS for each possible combination of input values. The amount of combinations scales exponentially with the number of inputs in the FIS, making this solution very computationally intensive.

Fuzzy output function Using the fuzzy output function, as shown in Figure 2.5, to represent the trust has similar advantages to using the crisp output value. It enables even more fine granular changes to the trust, as much information is still present compared to the defuzzified crisp value. However, one of the most significant downsides is that these potentially complex functions are hard to represent in a model. Additionally, like when using the crisp output value, it is hard to find valid reasoning for doing any of the fine granular changes, as we do not have information about any use cases or scenarios from the real world. The problem regarding comparability is emphasized due to the variability in comparing functions.

Labels Using labels to represent trust is inspired by the way properties are represented in the data flow analysis approach described in Section 2.2. Labels can be represented as literals of an enumeration. We propose a mapping of the crisp output value to one of a fixed set of labels. Each label represents one fuzzy set of the fuzzy output of a FIS. A trust output of a FIS, which is made up of three fuzzy sets, 'low', 'mid', and 'high', results in three possible trust values that can be utilized for access control analysis. Simply defining the possible trust labels beforehand and forcing each output of a FIS to be made up of fuzzy sets according to these labels makes the resulting mapped trust labels comparable. Depending on the situation or system setup, a change in trust is achieved by choosing a trust label that is semantically higher or lower than the current label. This makes changes in the trust values less abstract and easier to understand than the two previous solutions. More fine granular changes to the trust are not possible. By increasing the number of trust labels and consequently the number of fuzzy sets of the fuzzy output of a FIS, we can generally increase the granularity, with the downside of simultaneously increasing the complexity and modeling effort of the FIS. However, this solution still grants the software architect the ability to analyze the system and change environment parameters of services iteratively, and monitor if these changes in the system's deployment can result in potential violations against access control rules. Additionally, as discussed in the two previous solutions, we currently do not have any use cases that would benefit from fine granular changes in trust or where these changes could not be represented equally using labels.
As we have discussed, we currently do not see any real-world benefit of representing trust as the fuzzy or crisp outputs. Since it integrates well with the existing DFD approach, we described in Section 2.2, and as we currently see no benefit in using the other representations, we choose to represent trust as labels.

6.3 Extended DFD Metamodel

As we have discussed in Section 6.2, we choose to use labels to represent trust in access control properties. Using labels to represent the trust enables us to reuse concepts and model elements that already exist in the DFD approach we described in Section 2.2. The concept of labels is used in the DFD model and analysis to represent the clearance of nodes and the classification data. We add additional labels representing trust that can be calculated using a FIS and explicitly assigned to property labels. An excerpt of the class diagram representation of the extended metamodel is shown in Figure 6.5. We highlight the model elements added by our approach by using bold lines.

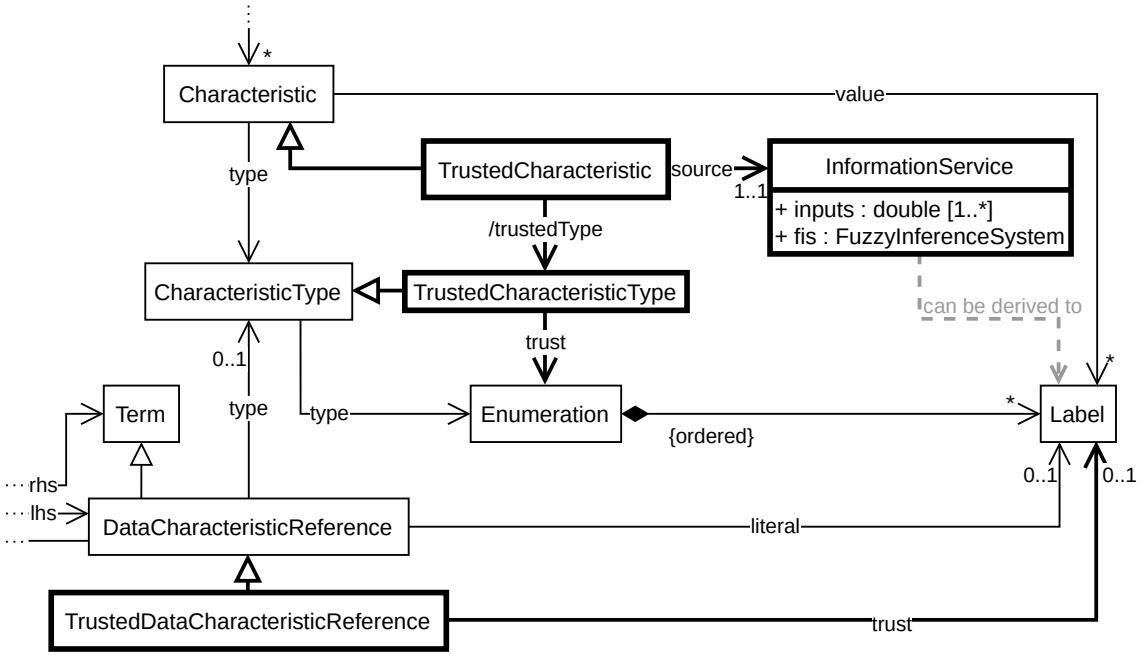


Figure 6.5: Class diagram showing the extended metamodel representation of the DFD.

6.3.1 Information Services

InformationService is an explicit representation of a service that supplies information used for access control, as we have described in Section 4.3. An information service contains a set of inputs and a FIS.
The set of inputs represents the values of environmental factors that influence the trust in the validity or correctness of the information. Environmental factors might be the accuracy of the source that is used to measure the information, e.g., a sensor, the average age of the supplied information, and various actual environment parameters that influence the trust.
The FIS specifies how the trust is calculated, that is associated with the properties supplied by the *InformationService*.

6.3.2 Extending Characteristics

To add the trust labels to the model, we add the *TrustedCharacteristicType*, which derives from *CharacteristicType*. In addition to *CharacteristicType*, the *TrustedCharacteristicType* references an additional *Enumeration*, which is made up of the trust labels.
To explicitly correlate a *InformationService* with a property, we add *TrustedCharacteristic*, which derives from *Characteristic*. *TrustedCharacteristic* references a *InformationService* and a *TrustedCharacteristicType*, which is derived from the *type* reference of *Characteristic*. The *value* reference, which represents an access control property is trusted with the trust label that can be calculated from the service referenced with *source*.
Based on the way we added *TrustedCharacteristic*, we also add *TrustedDataCharacteristicReference*, which derives from *DataCharacteristicReference*, which refers to an additional

Label representing the *trust* in the label referenced as *literal*. The *TrustedDataCharacteristicReference* enables the use of trust labels in assignments of behavior definitions.

Using our extended data flow diagram metamodel, we have implemented the running example that we describe in Chapter 5. We present an object diagram representation of an excerpt of this metamodel instance in Figure 6.9. The Scientist, Worker, and Visitor are represented by instances of *Actor*. Each of the actors has an owned behavior and a *TrustedCharacteristic*. The behavior is not specially shown in the model. The characteristics are of the type 'Location', which represents the location of the actor. In the running example, the location information of the scientist and worker is supplied by a service, which uses a high sensitivity GPS sensor. In contrast, the visitor's location is supplied by a service that uses a standard GPS sensor. These services are represented by the instances of *InformationService* 'HighSensitivityGPS' and 'StandardSensitivityGPS'. We have already described the FIS instance of the information services in Section 6.1, and show a representation of the membership functions in Figure 6.4. The characteristic of the scientist has the value 'Laboratory', and the characteristic of the worker has the value 'Meeting_Room' because the high sensitivity service supplies the precise location. Due to the poor quality of the standard sensitivity service when used indoors, the characteristic of the visitor has the values, 'Outside', 'Meeting_Room' and 'Laboratory'.
The 'read_DB' instance of *Process* connects two instances of *DataFlow*. One data flow from the laboratory database to the process and a data flow from the process to the actors. The laboratory database is represented by an instance of *Store*. Similar to the other actors, the laboratory database has an instance of *TrustedCharacteristic*. In the running example, this characteristic defines the access rights for read access. The database should only be read by actors located in the laboratory, so the characteristic's value is 'Laboratory'. The trust of the characteristic is fixed to 'trust_high', as the access rights should not depend on environmental factors of the system.

6.3.3 Behavior Definition

The data flow diagram approach we described in Section 2.2 represents behavior definitions as a combination of input and output pins, as well as *Assignments*. These assignments are made up of *Terms* which define how properties are assigned from the input pin to the output pin.
The original publication of the data flow diagram approach describes behavior templates that define reoccurring behavior of nodes in use-cases and case studies:

- Forward
- Sync
- Declassify
- Characteristic Changer
- Join

Figures 6.6, 6.7 and 6.8 give a visual representation of the behavior templates. Each pair of purple and orange rectangles represents a characteristic, which either is the input or the output of the behavior. The purple rectangle represents the access control property label, which the *TrustedCharacteristic* references with *value*. The orange rectangle represents the trust label of the corresponding access control property, which the *TrustedCharacteristic* references with *trust*. These characteristics define the properties required to access the data that flows to or from the node for access control.
We split these five behavior templates in two categories, templates where the behavior does not modify the input characteristics before passing it on as their output and templates where the behavior modifies the input in some way.
Figure 6.6 shows the two behavior templates, that do not modify the input. The *Forwarder* directly copies the characteristic from input to output, without changing it. A *Syncer* acts like the Forwarder behavior template, but waits for additional input, without considering the characteristics of the additional input.

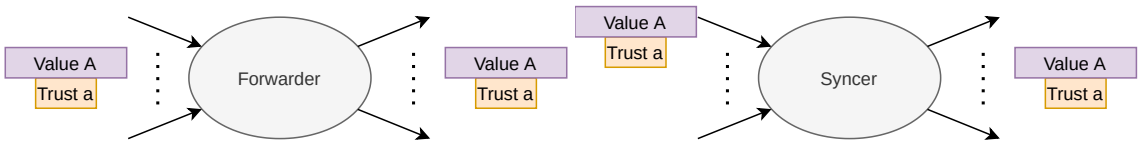


Figure 6.6: Extended behavior templates [66].

The behavior templates that modify the input characteristics are shown in Figure 6.7. The *Declassifier* copies the characteristics from the input and adds a predefined characteristic to the output. For access control, the characteristics represent access rights. By including an additional access right, the *Declassifier* loosens the overall constraints of the output.
The *Characteristic Changer* acts like the Forwarder but changes the value of the input characteristic to a defined value. For example, for a node that moves data from a database located in the USA to a database located in Asia, the *Characteristic Changer* template might be used to change the location value of the input characteristic from 'USA' to 'Asia'.
The *Joiner* determines the output characteristics by merging the received value labels of each input characteristic. The merging can be specified depending on the use case. For example, the *Joiner* might propagate the highest classifying label, create a union of all incoming data characteristics, or apply an intersection. As can be seen in Figures 6.6 and 6.7, the described behavior templates do not address the added trust directly and only forward the trust labels. We propose two additional behavior templates, shown in Figure 6.8, which are explicitly centered around changes in trust.
The first proposed behavior template is the *Trust Decreaser*. Similar to the *Declassify* behavior shown in Figure 6.7, the *Trust Decreaser* copies the input value and trust combination and adds an additional value trust combination to the output. The added value trust combination has the same value as the input but sets the trust to a defined label.
The second proposed behavior template is the *Trust Changer*. The *Trust Changer* behavior template is based on the *Characteristic Changer* behavior. Instead of explicitly changing the value of a characteristic, the *Trust Changer* changes the trust label to an explicitly set

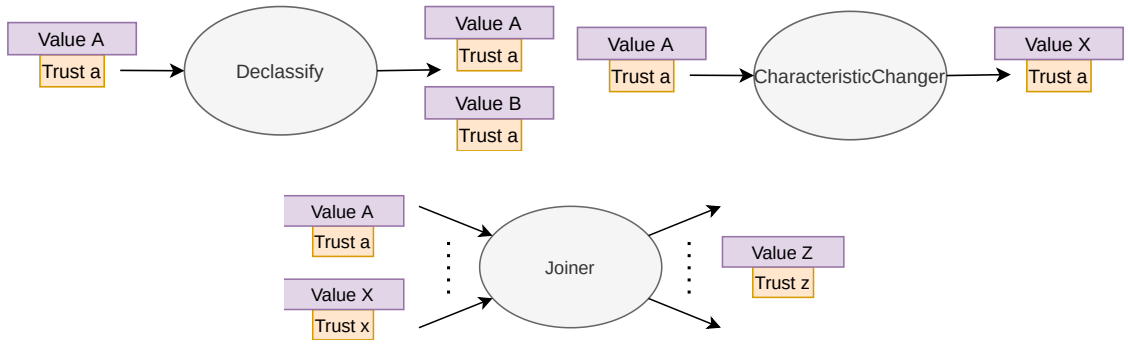


Figure 6.7: Behavior templates [66] that change or add labels.

trust label.
Trust labels can be viewed as hierarchical, where a higher label implies the existence of the lower labels. It is up to the query that is to be executed to define whether or not the hierarchies are considered. If the query takes the hierarchy of trust labels into account, using a *Trust Changer* to change the input trust label to a lower one, or a *Trust Decreaser* to add an additional characteristic with a lower trust label, have an equal effect, as the lower trust labels already implied the higher trust labels. However, if these hierarchies are not taken into account by the query, reducing the required trust label with the *Trust Decreaser* leaves access with a higher trust label still allowed, while changing the trust label with the *Trust Changer* excludes access with higher trust labels.

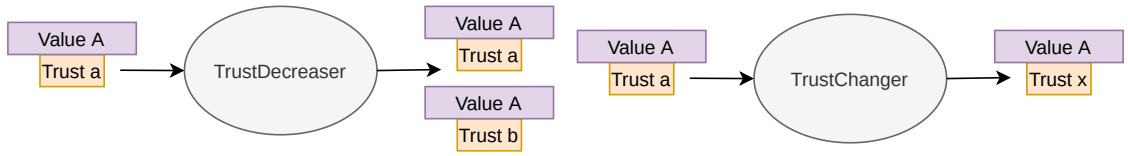


Figure 6.8: Proposed behavior templates that explicitly change trust of an label.

All the behavior templates we describe in this section can be combined to represent the behavior of a system with our extended data flow diagram metamodel. As described in the original publication [66], instances of nodes that have one of these templates as their behavior can be connected to represent system behavior. For example, a node might declassify the characteristic value, followed by another node which increases the trust. Changes in characteristic value and trust might occur simultaneously, and representing this behavior in two separate nodes might be semantically incorrect. To combat this situation, all behavior templates we describe in this section can be combined to form a single behavior. For example, the *Declassifyer* and *Trust Increaser* are combined into one behavior, which can be set as the behavior of a single node.
As shown in Figure 6.9, we use the *Forwarder* template as the behavior of the 'read_DB' instance of *Process* and the 'Laboratory_DB' instance of *Store*. Additionally, all behaviors add the value and trust label of the current node to the data.

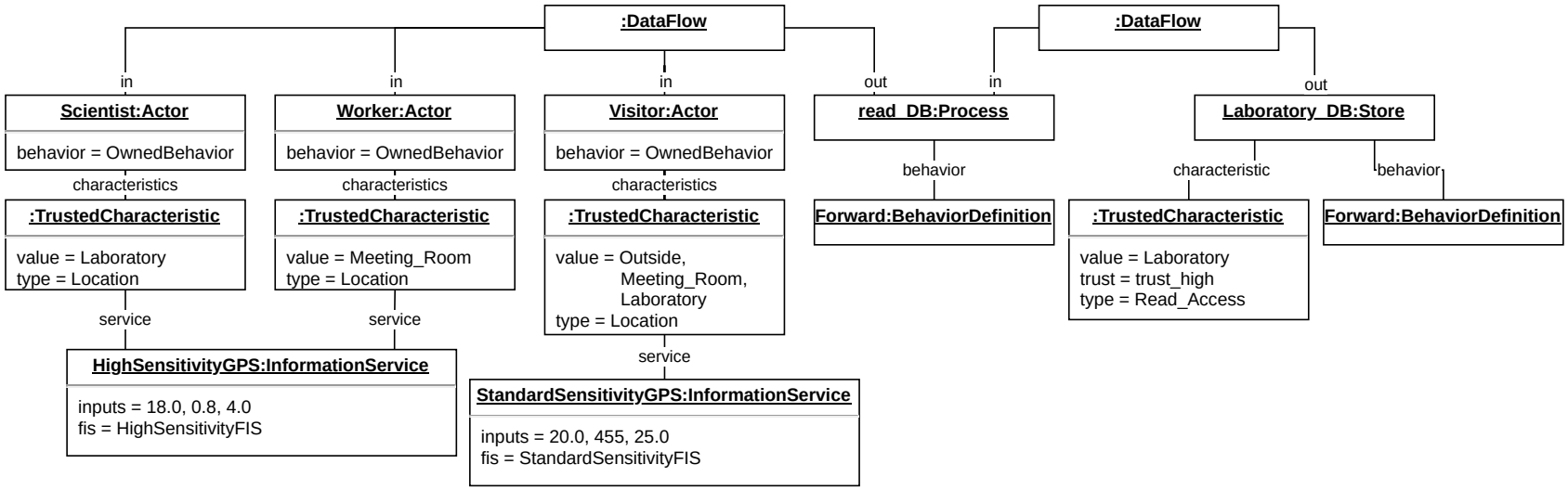


Figure 6.9: Object diagram excerpt of the running example DFD.

7 Analysis Process

This chapter focuses on our extended analysis process that is used with the data flow diagrams that we describe in Chapter 6. The analysis process consists of the transformation of the data flow diagram to a Prolog code base, which we describe in Section 7.2. This transformation includes the calculation of a trust value based on the added information services, which is described in Section 7.1. The resulting Prolog code base is then used to solve a Prolog query that describes illegal data flows. A query usually consists of a conjunction of clauses. We describe the query for our running example (see Chapter 5) in Section 7.3.

7.1 Calculation of Trust

The calculation of the actual trust values is done during the transformation of the data flow model to Prolog. We transform each *InformationService* to a single label. First, we transform the contained FIS to a fis file. This file format is defined by the Fuzzy Logic Toolbox of MATLAB [44] to have a textual representation of a FIS. Using fuzzy logic control libraries [56] with the created .fis file, we run the FIS, pass the input values of the *InformationService* as inputs and receive the defuzzified crisp output as a result.
To map the crisp output value to fixed labels, as described in Section 6.2, we first calculate the truth values for each membership function, of which the output of the transformed FIS is made up of. Due to the restriction that forces each FIS output to be made up of fuzzy sets according to the set of trust labels, we can map the resulting truth-values to each of the corresponding trust labels. As we want to transform each *InformationService* to a single trust label, we choose the label with the highest truth value. If two labels have an equal truth value, the one that represents the weaker linguistic value is chosen, i.e., if 'high' and 'mid' trust have the same truth value, 'mid' is chosen as label. Choosing the truest truth label seemed the most logical approach, especially considering that use cases, which could be used to deduce other more demanding selection processes, are currently not available. We have already described this problem in Section 6.2, where we decided to use labels because there are no use cases that would demand more fine granular representation.

7.2 Extended Prolog Mapping

To take advantage of the newly added trust value in the DFD analysis (see Section 2.2.3), we have to extend the transformation of the DFD model instance to Prolog. This transformation has already been described in Section 2.2.2, so we will only cover the parts of the transformation that we had to modify to add the trust value.

Listing 7.1: Prolog facts representing the DFD structure, extended with trust label.

```
1 characteristicType(CT),
2 characteristicTypeValue(CT, V, I),
3 characteristicTypeTrust(CT, T, I),
4 nodeCharacteristic(N, CT, V, T).
```

Similar to the model representation in Section 6.3, our changes to the transformation of DFD to Prolog are mainly focused on characteristics and characteristic types. We underline our newly added fact and changes to the existing facts, in Listing 7.1 and Listing 7.4. The transformation of characteristic types shown in line 1 and 2 of Listing 7.1 stays unchanged. For each label of the enumeration to which a trusted characteristic type refers to as *trust*, we add the creation of a fact, shown in line 3 of Listing 7.1. Similar to line 2 of Listing 7.1 we replace `CT` with the identifier of the trusted characteristic type and `T` with the identifier of the trust label. `I` is replaced with the position of the trust label in the enumeration. When transforming to the `nodeCharacteristic` fact in line 4 of Listing 7.1, we do not want to create a fact for each label assigned to a node, like it is described in Section 2.2, as this would mix *value* labels and *trust* labels. We rather view the *trust* labels as additional information to *value* labels, as a trust label does reflect trust in the correctness of a value (see Section 6.2). Consequently we create a fact as shown in line 4 of Listing 7.1 for each characteristic, that is assigned to a node. We first calculate the trust value of the referenced *InformationService* and map the value to a trust label, see Section 7.1. Similar to the original transformation (see Section 2.2.2), we replace `N` with the identifier of the corresponding node, `CT` with the identifier of the characteristic type and `V` with the identifier of the *value* label. We replace `T` with the identifier of the *trust* label, that has been calculated and corresponds to the *value* label, assigned to the node.

Listing 7.2: TrustedCharacteristicType Location of the running example transformed to Prolog.

```
1 characteristicType('Location').
2 characteristicTypeValue('Location', 'Outside', 0).
3 characteristicTypeValue('Location', 'MeetingRoom', 1).
4 characteristicTypeValue('Location', 'Laboratory', 2).
5 characteristicTypeTrust('Location', 'trust_low', 0).
6 characteristicTypeTrust('Location', 'trust_mid', 1).
7 characteristicTypeTrust('Location', 'trust_high', 2).
```

Listing 7.2 shows the transformation of the trusted characteristic type *Location* from our running example, see Chapter 5. The three *value* labels are each transformed to the Prolog facts in lines 2 to 4. The corresponding *trust* labels are transformed to the Prolog facts in lines 5 to 7. In their first argument, the facts representing the *value* labels and the facts representing the *trust* labels explicitly point to 'Location', which is defined as a characteristic type in line 1.
Listing 7.3 shows the transformation of the characteristics which are assigned to the *Visitor* of our running example once he enters the meeting room. Like described in Section 2.2.2, the *Visitor* node is defined as an actor in line 1. Lines 2 to 4 each describe a characteristic

of the *Visitor*. For each of these characteristics, the trust label is calculated, as described in Section 7.1, and set as the last argument of each of the Prolog facts.

Listing 7.3: Visitor node characteristic value labels and calculated trust label.

```
1 actor('Visitor').
2 nodeCharacteristic('Visitor', 'Location', 'Laboratory', '
    trust_low').
3 nodeCharacteristic('Visitor', 'Location', 'MeetingRoom', '
    trust_low').
4 nodeCharacteristic('Visitor', 'Location', 'Outside', 'trust_low
    ').
```

Listing 7.4: Prolog rule representing node behavior, extended with trust label.

```
1 characteristic(N, PIN, CT, V, T, S, VF) :- ...
```

The behavior specification rule shown in line 1 of Listing 7.4, also needs to consider trust. Consequently, with the added trust, the rule body is now created in a way, that it evaluates to true, if the *value* label `V` and *trust* label `T` of characteristic type `CT` is available on the pin `PIN` of node `N`.

Listing 7.5: read_DB behavior Prolog transformation excerpt of forward behavior.

```
1 process('read_DB').
2 inputPin('read_DB', 'in').
3 outputPin('read_DB', 'out').
4 characteristic('read_DB', 'out', 'Location', 'Laboratory',
    'trust_high', S, VISITED) :-
5   inputFlow('read_DB', 'in', _, F0, VISITED),
6   S0 = [F0 | _],
7   S = [S0],
8   characteristic('read_DB', 'in', 'Location', 'Laboratory',
      'trust_high', S0, VISITED).
```

For the *read_DB* process in our running example, Listing 7.5 shows an excerpt of the transformed behavior description. Line 1 defines that *read_DB* is a process node. Line 2 and 3 define an input and output pin, which are assigned to the *read_DB* process.
The `characteristic` rule in line 4 of Listing 7.5, is defined for the output pin from line 3. Lines 5 to 8 define the rule body. The 'read_DB' process has the forward behavior described in Section 6.3.3. The rule states that the value label 'Laboratory' with trust label 'trust_high' of type 'Location' is available at the output pin 'out' if there is an input flow to the process that has the same labels.

7.3 Prolog Query

As described in Section 2.2.3, the analysis is realized with queries to the generated Prolog program. The clauses that can be used to define the queries are based on the facts and rules that make up the Prolog program. To account for the trust labels we added, we add a

new clause and extend two clauses that were described in Section 2.2.3. We underline our newly added clause and changes to the existing clauses, shown in Listing 7.6.

Listing 7.6: Modified clauses of the Prolog API to specify confidentiality analyses.

```
1 characteristicTypeTrust(CT, T, I),
2 nodeCharacteristic(N, CT, V, T),
3 characteristic(N, PIN, CT, V, T, S).
```

The clause in line 1 of Listing 7.6 is similar to `characteristicTypeValue(CT, V, I)` in Listing 2.3. It can be used to find the trust label identifier `T` of characteristic type `CT`. Line 2 shows a clause to find a tuple of label identifier `V` and corresponding trust label identifier `T` of characteristic type `CT` that is active on the node with identifier `N`. Line 3 shows the clause to find a tuple of label identifier `V` and corresponding trust label identifier `T` of characteristic type `CT` that is available on the pin with identifier `PIN` of the node wit identifier `N`.

Listing 7.7: Query to find illegal data flows in our running example.

```
1 actor(A),
2 store(ST),
3 inputPin(A, PIN),
4 flowTree(A, PIN, S),
5 traversedNode(S, ST),
6 nodeCharacteristic(A, 'Location', LOC, TRUST),
7 \+ nodeCharacteristic(ST, 'Read_Access', LOC, TRUST).
```

For our running example, the clauses are combined to form the query shown in Listing 7.7. In our running example, each actor has one or multiple active 'Location' values and trust label pairs that indicate the actor's location. The *Laboratory DB* store has an active 'Read_Acess' value and trust label pair that defines the location that is required in order to access the data base. For the analysis, we want to match the 'Location' value and trust label pairs of the actors with the 'Read_Access' value and trust label pair of the store. If no label pair can be matched, the corresponding data flow is illegal.
In lines 1 and 2, we first define that the identifier `A` belongs to an actor and identifier `ST` belongs to a store. Line 3 defines that the pin with identifier `PIN` is an input pin of the actor with identifier `A`. As we only compare labels of nodes, we do not have a flow tree from a `characteristic` clause to verify that there even exists a data flow between actor and store. Line 4 contains the helper clause to build a flow tree `S` of data flows that arrive at the input pin with the identifier `PIN` of the actor with identifier `A`. Line 5 checks if the store with identifier `ST` is traversed in the flow tree `S`. Line 6 defines, `LOC` and `TRUST` as placeholders for all tuples of label and trust label identifiers of the 'Location' characteristic type that are active on actor `A`. Line 7 defines that the tuple of placeholders `LOC` and `TRUST` are also identifiers of labels of the 'Read_Access' characteristic type and active on store `ST`. However, the line starts with `\+` which is the 'not provable' operator of Prolog. This means that line 7 is only true if the labels with identifiers `LOC` and `Trust` are either not from characteristic type 'Read_Access' or are not active on store `ST`, effectively saying that the labels could not be matched.

Listing 7.8: .

```
1  solution 0:
2    A: Worker
3    ST: Laboratory DB
4    LOC: MeetingRoom
5    TRUST: trust_high
6    S: invalid read(dbEntry), LaboratoryDB->readDB(dbEntry),
       write(dbEntry)->LaboratoryDB, Scientist->write(dbEntry)

8  solution 1:
9    A: Visitor
10   ST: Laboratory DB
11   LOC: Laboratory
12   TRUST: trust_low
13   S: invalid read(dbEntry), LaboratoryDB->readDB(dbEntry),
       write(dbEntry)->LaboratoryDB, Scientist->write(dbEntry)

15 solution 2:
16   A: Visitor
17   ST: Laboratory DB
18   LOC: MeetingRoom
19   TRUST: trust_low
20   S: invalid read(dbEntry), LaboratoryDB->readDB(dbEntry),
       write(dbEntry)->LaboratoryDB, Scientist->write(dbEntry)

22 solution 3:
23   A: Visitor
24   ST: Laboratory DB
25   LOC: Outside
26   TRUST: trust_low
27   S: invalid read(dbEntry), LaboratoryDB->readDB(dbEntry),
       write(dbEntry)->LaboratoryDB, Scientist->write(dbEntry)
```

Executing the described query shown in Listing 7.7 on the situation of our running example shown in Figure 5.5 yields the solutions shown in Listing 7.8. Each solution states the actual values of attributes that Prolog was able to resolve. For the query in Listing 7.7 this is the actor `A` that reads data from the database but is not supposed to, the store `ST` which defines his read access rules and from which the data is read, the location `LOC` and trust `TRUST` of the actor, that do not match with the read access rules defined by the store. `S` shows the flow tree, which can be used to trace the data flow to its origin.
Solution 0 in lines 1 to 6 show the solution that correctly identifies the illegal data flow to the *Worker* as the location is mismatched. Solutions 1 to 3 correctly identify the illegal data flow to the *Visitor*. Lines 8 to 13 show one violation as the trust is low, even though the location theoretically matches. Lines 15 to 20 and 22 to 27 show other violations that occur as the other locations and trust of the *Visitor* do not match the read access rules of the store either.

8 Evaluation

In this chapter, we present the evaluation of our contribution. To gain a well-designed evaluation structure, we use the Goal Question Metric (GQM) approach of [6]. We further partition this chapter in the following sections:
Section 8.1 covers the evaluation design, including the GQM plan. In the evaluation design, we define our evaluation goals of Applicability, Accuracy, and Scalability. We further describe each goal and define the corresponding questions and metrics. In Section 8.2, we give a detailed description of the exact setups that we used to evaluate each question. In Section 8.3 we present and briefly discuss the findings of the evaluation of each question. To conclude this chapter, we discuss the threads of validity of our evaluation and the assumptions and limitations of our contribution in Section 8.4 and Section 8.5.

8.1 Evaluation Design

To gain a well-designed evaluation structure, we use the Goal Question Metric (GQM) approach of [6]. We define three evaluation goals:

- EG-1: Examine the applicability of the approach regarding the iterative creation process of software architecture.
- EG-2: Examine the accuracy of the approach, with defined classes of errors, by comparing the output of analysis runs to expected results.
- EG-3: Examine the scalability of the approach when analyzing large systems.

8.1.1 Evaluation Design for Applicability

To evaluate the applicability of the proposed concept of trust, we focus on the iterative creation and evolution process of software architecture. This raises questions regarding the applicability during design time and when the system is actively in use. Another part of the contribution is the calculation of trust by the means of a FIS. This raises a question about the applicability of FISs within the software development process. Another part of applicability is the ability of the extended data flow diagram to define confidentiality analyses, which raises a question about the expressive power of our approach.

We formulate the following questions:

- **Q1.1** Is the information to apply the proposed concept of trust to the described metamodel and analysis process available during design time?
- **Q1.2** Does adding the proposed concept of trust to the described metamodel and analysis process produce additional value if a system is already deployed?
- **Q1.3** Is the proposed FIS suitable to describe calculation rules for trust values?
- **Q1.4** Is the expressiveness of the original data flow diagram approach and analysis affected by the addition of the proposed concept of trust?

Since the questions described are not suited to be answered through metrics or measurements, we strive to answer the questions argumentatively. All questions will be discussed and, if appropriate, answered using examples.

8.1.2 Evaluation Design for Accuracy

To evaluate the accuracy, we base our evaluation design regarding evaluation goal **EG-2** on the way the data flow diagram approach that our contribution is based on (see Section 2.2) was evaluated. The left side of Figure 8.1 shows the original design. They define a pair of scenarios for a given data flow diagram instance. One scenario without an issue and an additional scenario where an issue has been introduced that leads to violations concerning the analysis. The issues are either derived from related work or by themselves.
By adding our concept of trust, we identify different kinds of issues concerning the analysis. An issue can be due to mismatched characteristic labels or be due to mismatched trust labels. Based on this distinction, we define four different scenarios, which need to be considered to evaluate the accuracy of our contribution. Figure 8.1 shows the pair of scenarios of the original approach, compared to the four scenarios we define for our contribution.
The four scenarios we define are:

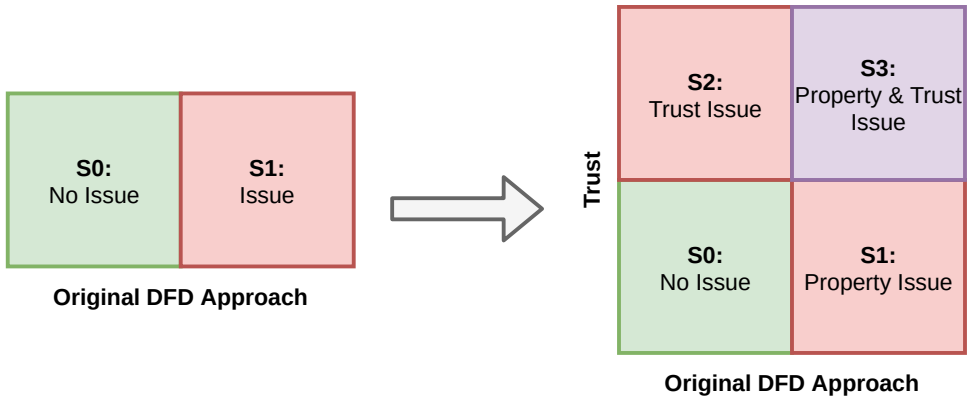


Figure 8.1: Increased number of scenarios that have to be considered for accuracy.

S0: No issue.

S1: An issue that is due to mismatched property labels is introduced by adding an illegal data flow to the data flow diagram.

S2: An issue that is due to mismatched trust labels is introduced by identifying a scenario based on the data flow diagram that reduces the trust label of a property.

S3: An issue is introduced, that is due to mismatched property and trust labels, by combining **S1** and **S2**.
From these scenarios, we derive the following evaluation questions to evaluate **EG-2**:

- **Q2.1** What is the accuracy of the extended analysis in identifying issues in data flows regarding issues that result from mismatched properties?
- **Q2.2** What is the accuracy of the extended analysis in identifying issues in data flows regarding issues that result from low trust?
- **Q2.3** What is the accuracy of the extended analysis in identifying issues in data flows regarding issues that result from a combination of mismatched properties and low trust?

To answer the questions we create pairs of scenarios. Each pair consists of a **S0** scenario without an issue and either a **S1**, **S2** or **S3** scenario, depending on the evaluation question. The metrics used to evaluate the questions concerned with goals **EG-2** are:

M-2.1 Precision (p) calculates the ratio of correctly identified issues t_p to the sum of $t_p + f_p$. f_p, or the false positive value, are the number of issues that where identified as such, but actually are not an issue that was introduced in the **S1**, **S2** or **S3** scenarios. $p = \frac{t_p}{t_p+f_p}$ [55].

M-2.2 Recall (r) calculates the ratio of correctly identified issues t_p to the overall positive value P. For our use, the overall positive value P is the number of actually introduced issues in the **S1**, **S2** or **S3** scenarios. P is the sum $t_p + f_n$. f_n, or the false negative value, are the number of issues that should have been identified, but where not. $r = \frac{t_p}{P} = \frac{t_p}{t_p+f_n}$ [55].

8.1.3 Evaluation Design for Scalability

Since our approach can be used to represent large systems, with many different *InformationServices*, *trust values* and general types of properties, our third evaluation goal (**EG-3**) is concerned with the scalability of our approach. The execution time of a query on the Prolog code mainly depends on the structure of the data flow diagram instance, the executed query, the used Prolog library, and optimization techniques, which is out of the scope of this thesis. For the scalability evaluation, we focus on the time needed to calculate trust and the time needed to map an extended data flow diagram to Prolog code, as our approach primarily influences the mapping. The execution time of the mapping mainly depends on the size of the resulting Prolog code. Some model aspects, like the number of characteristics and pins of each node or the amount of data flows in the system,

can have a strong impact on the size of the created Prolog code. Each assignment of the behavior of a node, for example, adds a `characteristic(N, PIN, CT, V, T, S, VF) :- ...` rule for each value of each characteristic type to the Prolog clauses of the corresponding node (see Section 7.2). However, we want to focus on the scalability of the model aspects we added or modified as part of our approach.
This raises the following questions:

- **Q3.1** How does the time required for calculating the trust label scale when increasing the number of environmental factors that impact trust?
- **Q3.2** How does the time to map the model to Prolog scale when increasing the number of information services and properties?
- **Q3.3** How does the time to map the model to Prolog scale when increasing the number of trust labels?
- **Q3.4** How does the time to map the model to Prolog scale when increasing the number of types of properties?

We use the time needed for each run of the FIS or mapping as metric *M-3* to evaluate the questions concerned with EG-3. The times needed to answer each question are measured, and corresponding aspects of the models are incremented individually. The times of each increment of runs are plotted to better view the general tendency of the time behavior.

8.2 Evaluation Setup

In this section, we define the setups that we use to answer the questions presented in the previous section. These setups describe the DFD instances and analysis queries used to evaluate the applicability, accuracy, and scalability of our approach.

8.2.1 Setups for Evaluating Applicability

The first two questions regarding applicability, **Q1.1** and **Q1.2**, both focus on the applicability of our approach in different phases of the software development process. We generally view the software development process to be of iterative nature. During design time, the software architect iteratively constructs and changes the software architecture. After each iterative change, the model can be validated against a set of requirements using an analysis process like the one we present in our contribution. This iterative process continues after the system is deployed. Detailed models that reflect the deployed system are used with analysis processes to investigate the impact of what-if scenarios before implementing them. This reduces the risk of introducing errors into the existing system, which is particularly important in the context of access control.

8.2.2 Setups for Evaluating Accuracy

Due to the lack of existing publications that focus on the trust in access control properties, see Section 6.2, we can not use any existing scenarios from other publications for the trust issue scenarios (S2) described in the evaluation design. For property issue scenarios (S1), we can reuse the scenarios that were used by the original data flow diagram approach [66], that our contribution is based on.
To still be able to answer **Q2.2** and **Q2.3**, we extend existing use-cases with real-world environmental factors that are comprehensible and consistent with the use case description. We extract exact values for these environmental factors that produce a potential issue from existing research. As the research regarding real-world environmental factors, without any background knowledge, was very time consuming and had to be done for each use case individually in order to be consistent with the use case description, we base the accuracy evaluation regarding questions **Q2.2** and **Q2.3** on only one use case. To answer questions **Q2.2** and **Q2.3**, we use the ABAC use case, which has also be used to evaluate the original data flow diagram approach [66]. We chose the ABAC use case, as the ABAC model has high expressive power. Additionally, the trust chains, we have already described in Section 4.2.2, are originally defined for the ABAC access control model and can be closely represented with the environmental factors of our concept of trust.
The existing ABAC use case describes a banking system, which is deployed in the USA and Asia. Actors in the system can be Clerks or Managers. Clerks can register customers, look them up and determine a credit line. Managers can additionally also register celebrities, or move customers between regions. A data flow diagram representation of the use case is shown in Figure 8.2.

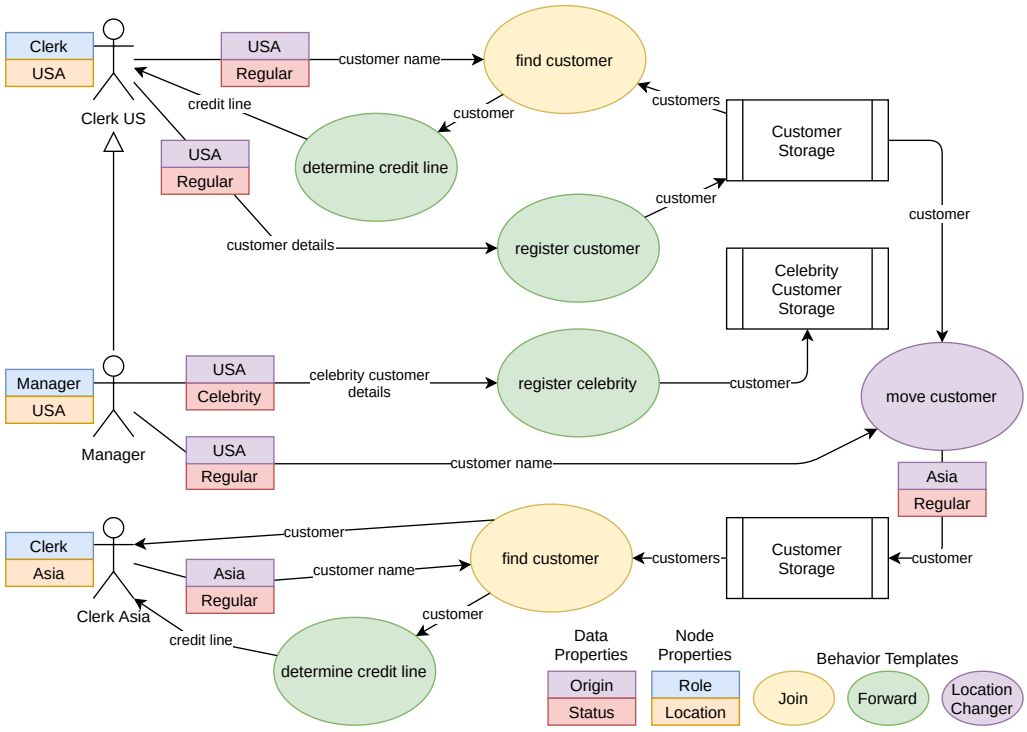


Figure 8.2: Representation of the ABAC use case data flow diagram [66].

Four properties are used for access control.

- The *Role* of an actor in the data flow model, e.g. Clerk, Manager.
- The *Location* describes the location of an actor, e.g. USA, Asia.
- The *Status* of a customer, which data is processed, e.g. Regular, Celebrity.
- The *Origin* of the customer data, e.g. USA, Asia.

Setup for Q2.1

To answer the question concerned with the accuracy in identifying issues in data flows that result from mismatched properties, we define a pair of scenarios that consist of a **S0** and a **S1** scenario. We reuse the pair of scenarios that were used for the original data flow diagram approach [66]. As these scenarios do not contain our concept of trust, trust values need to be defined for each characteristic. For this question, trust should not introduce an issue to the **S0** scenario and have no effect on the mismatched properties of the **S1** scenario. We add a *default* trust value to every characteristic. The *default* trust label represents complete trust in the validity of the property. This way, the resulting scenarios semantically match the original scenarios, as the properties are always valid and environmental factors are not considered.

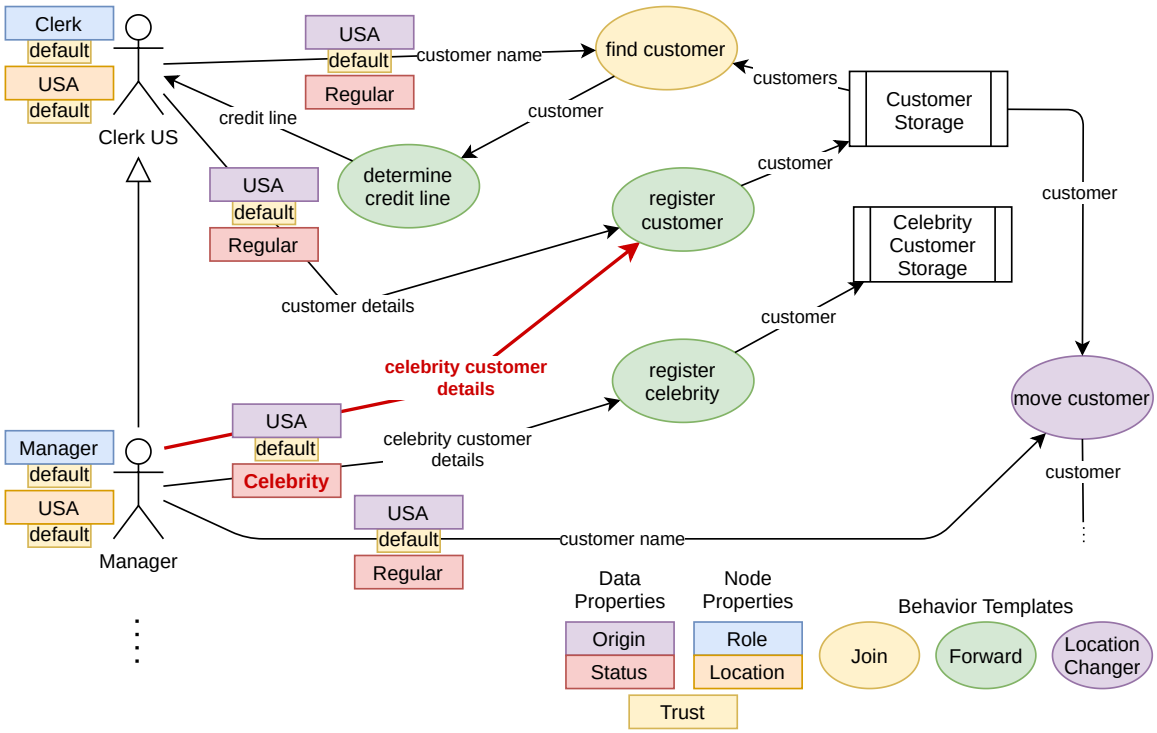


Figure 8.3: Reduced representation of the ABAC data flow diagram, with an introduced issue regarding mismatched properties.

This way, we can also ensure, that functionality of the original approach is preserved and our extension has not affected the capability of the analysis in finding issues in data flows

that result from mismatched properties. An example data flow diagram representation of the previously described ABAC use case, with *default* trust is shown in Figure 8.3. An exemplary issue that is introduced in the **S1** scenario is highlighted in red.
For the sake of clarity, we only briefly describe the remaining use cases used to answer **Q2.1**. Seifermann et al. have created a data set [67], containing detailed descriptions of the use cases, as well as **S0** and **S1** scenarios. The data set also contains the data flow diagram instances that we use as a base for this setup and add our previously described *default* trust to.

- DAC: A file-sharing system used by a family to share and view pictures. The actors *mother* and *dad* add and view pictures. The actor *aunt* only views pictures. An actor called *indexing bot* might discover the stored pictures but is not allowed to access them. To introduce the issue, a flow is added that represents a scenario where the *indexing bot* views the pictures.

- MAC: A airspace monitoring system with three levels of classification. The is used to monitor civil and military planes. The actor *clerk* uses the system to create and store weather reports. An actor *flight controller* registers civil airplanes and determines new routes based on the *clerks* weather reports and the position of other civil planes. The actor *military flight controller* registers military airplanes and determines new routes for military airplanes by taking into account the weather reports of the *clerk* and the civil airplane positions of the *flight controller*. The issue is added by creating a scenario, where the *flight controller* also takes into account the positions of military planes when determining new routes.

- ContactSMS: A user managing his contacts and sending SMS. The actor *User* can perform multiple actions on his list of contacts, including sending an SMS to a selected contact. To send an SMS, the number of the contact is extracted and sent to the actor *SMS Gateway*. To introduce the issue, the extraction of the number is skipped, effectively sending all contact information to the *SMS Gateway*.

- DistanceTracker: A service that tracks locations and running statistics. The actor *User* sends his location to a distance tracker service that stores the locations. The locations are used by the distance tracker service to calculate the distance and, with the *User*'s consent, sends the distance to a tracking service, which stores the distance. The issue is introduced by skipping the declassification process, which requires the *User*'s consent.

- TravelPlanner: A user booking a flight with a travel planner app. The app connects to a travel agency, which connects to the airline, to show and book available flights. To pay for a flight, the User's credit card information is taken from a banking app. To introduce an issue, the declassification of the User's credit card information is bypassed, allowing for unwanted banking information to be sent to the travel agency and airline.

Setup for Q2.2

To answer the question concerned with the accuracy in identifying issues in data flows that result from a lowered trust, we define a pair of scenarios that consist of a **S0** and a **S2** scenario. To properly cover all aspects of the used ABAC use case, we split the **S2** scenario into two sub-scenarios. The first scenario (**S2.1**) covers the trust in the *Role* properties of the actors, the second scenario (**S2.2**) covers the trust in the *Location* property of the actors. We further define an analysis query that is applied for both scenarios.

Role The scenario that introduces a trust issue in the *Role* of an actor focuses on the age of the role information. Similar to the definition of the ABAC access control model [32] we do not directly address authentication mechanisms or other aspects of identity management, as the field is very wide, and we can not create a representative use case for real-world scenarios without applicable case studies. We thus assume that the actors in the ABAC use case are bound to identity providers. Identity providers are services that verify that an actor has a certain role by using a user name and password. A role and IP address pair is saved in a database, which the access control system uses to determine if an actor has a certain role. After a set amount of time, the role and IP address pair is removed from the database, which requires the actor to reverify his role with the identity provider before executing an access that requires a certain role.
If the manager wants to register a new celebrity, the banking system requires that the last verification of the *manager* role is less than 2 minutes old. However, to move a regular customer from the USA to Asia, the last verification of the *manager* role is required to be less than 5 minutes old. The original data flow diagram representation of the banking system can only represent this requirement by adding multiple hierarchical roles, mixing the semantics of roles with the semantics of time or age. With our concept of trust, we can define decreasing trust, depending on the age of the role information. For our extended model, we define a manager role provider service. The trust in the information supplied by the service is only dependent on the age of the role information. The FIS input of role

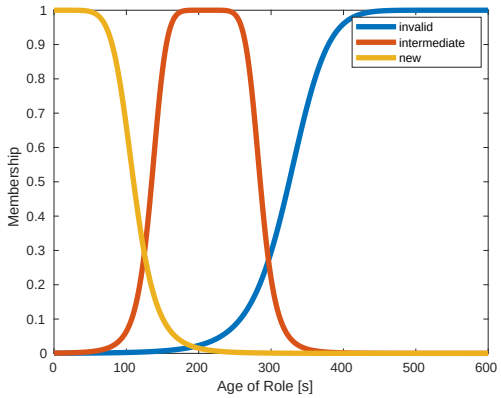


Figure 8.4: Fuzzy input of information age of the manager role provider service.

age is shown in Figure 8.4, the trust output is similar to the output defined for our running example, shown in Figure 6.4d. We chose the input membership functions to represent the prior description of the verification times for different actions of the manager. *New*

age corresponds to *high* trust, which is necessary to register a celebrity. *Intermediate* age corresponds to *mid* trust, which is necessary to move a regular customer from the USA to Asia.
For our scenario, the manager wants to register a new celebrity. The manager verifies his role with the manager role provider service and can then enter the celebrity customer details. However, the manager takes > 2 minutes to enter all of the celebrity customer details, and the action is aborted, effectively locking out the manager of registering new celebrities. A reduced representation of the extended data flow diagram is shown in Figure 8.5.

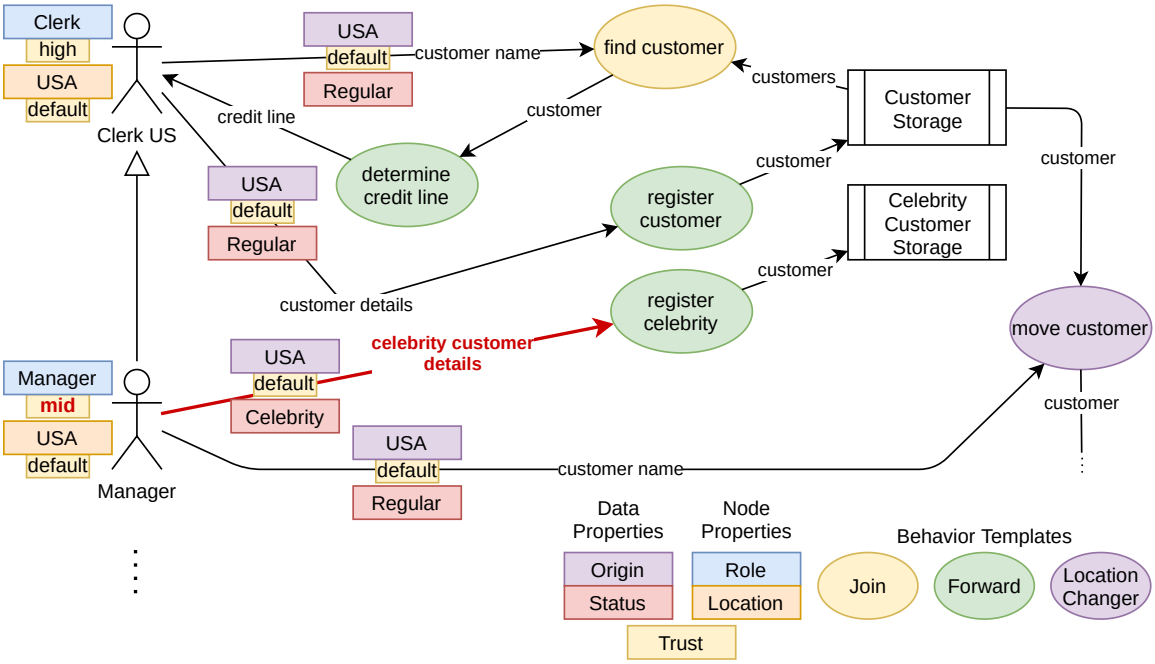


Figure 8.5: Reduced representation of the ABAC data flow diagram of the S2.1 scenario.

Location The scenario that introduces a trust issue in the *Location* of actors focuses on the resolution of the country geolocation from the IP addresses of the actors in the system. Actual accurate information about the geolocation of a related IP is only known by the internet service provider (ISP). This data is not available for commercial applications, like the banking system of the ABAC use case. The banking system has to rely on public or commercial geolocation databases and geolocation services. These databases use specially aggregated blocks of IP addresses to match the IP addresses to geolocations.
Even though vendors of such databases claim country-level geolocation accuracy of greater than 97%, this claim can not be universally confirmed [69]. A vast majority of entries in the databases refer only to a few popular countries, e.g., USA and Russia. This imbalance in the representation of countries creates an accuracy bias towards specific countries. Depending on the database, errors in the reported position range between 200Km and 800Km [54]. When looking at router, instead of end-host IP addresses, the country-level geolocation accuracy can differ by nearly 50% between databases [25].
For our scenario, we make the locations of the ABAC case study more specific. Instead of

the regions *USA* and *Asia*, we narrow the locations down to the specific countries *USA* and *Hong Kong*.
As a cost-saving measure, the banking system switches the geolocation database, which provides the location information, from the commercial NetAcuity [48] database to the free MaxMind-GeoLite [45] database. For the US, this change only reduces the country geolocation accuracy by a negligible amount. However, for countries like Hong Kong, this change reduces the country geolocation accuracy by more than 40%.
This low accuracy could result in incorrect system behavior once deployed and opens up the possibility for attacks utilizing the inaccurate country geolocation resolution. In our example, this issue results in the clerk from Hong Kong sometimes not being able to access the Hong Kong customer storage, as the access control system in place would block all data flows.
The original data flow diagram representation of the banking system has no way of representing this change. Consequently, analyzing the data flow diagram does not yield any insight into the changed situation. With our concept of trust, this change in databases can be represented in the model. A reduces representation of the extended data flow diagram is shown in Figure 8.6. The low accuracy directly impacts the trust in the location. This way, the analysis can identify that using the MaxMind-GeoLite database results in a trust issue in multiple data flows.

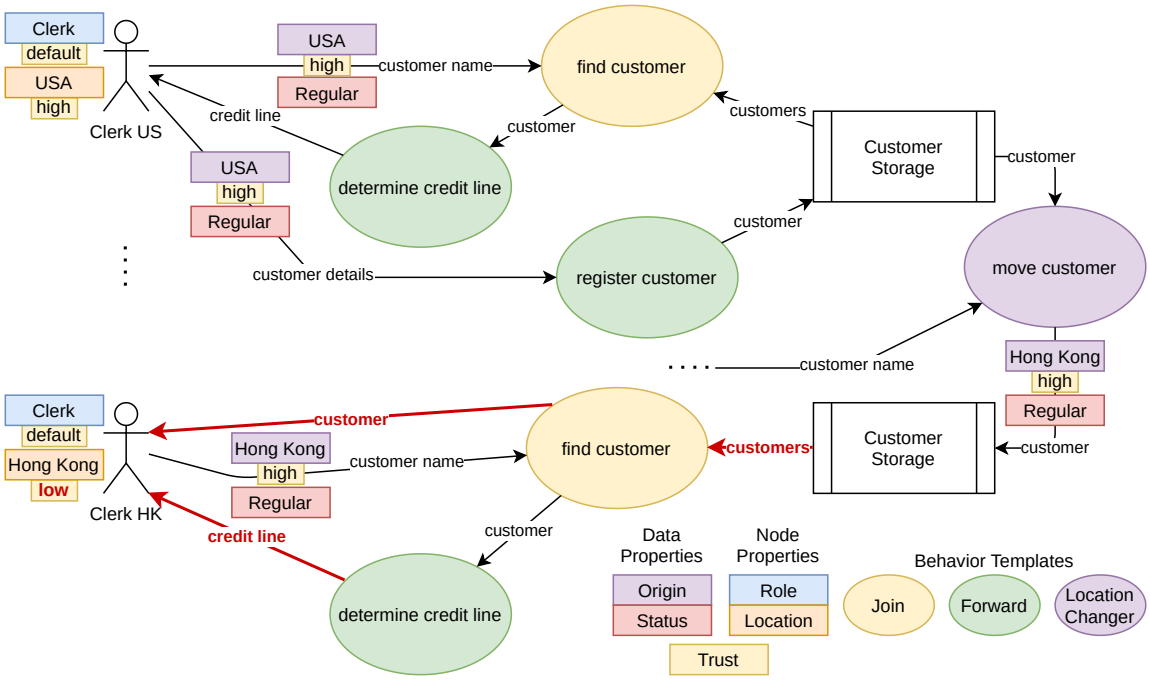


Figure 8.6: Reduced representation of the ABAC data flow diagram, using the MaxMind-GeoLite database.

Query We analyze *both* scenarios with the Prolog query, shown in Listing 8.1.
The query evaluates whether at least one of three kinds of data flows exist in the model. In line 1 of the query, we first ensure that node `A` is an actor node. Line 2 ensures that the pin `PIN` is considered an actual input pin of the actor `A`. The facts in lines 3 and 4 check the characteristics of the actor `A`, while the facts in lines 5 and 6 check the characteristics of the data that can flow to `PIN` of actor `A`. Lines 8, 9, and 10 define a disjunction, where each line contains a conjunction of facts that refer to the value or trust of the characteristics that are checked in lines 3 to 6.
Line 8 defines that a data flow is considered illegal, if the *Location* `LOC` of the actor is equal to the *Origin* `ORIG` of the customer nodeLiteral(CV, N),whose data is in the system, but the trust in the actors *Location* `LOC_TRUST` is not equal to the trust in the customer *Origin* `ORIG_TRUST`. With this statement issues like the one described in scenario **S2.2** can be identified.
Lines 9 and 10 define the use case that has been described for scenario **S2.1**. The facts in line 9 define that a data flow is illegal if the actor *Location* `LOC` of the actor and the *Origin* `ORIG` of the customer data are different and the actor does not have the 'Manager' role or trust in the 'Manager' role is 'low'. This means that only an actor with the 'Manager' role that is at least trusted 'mid' is allowed to move customers. Line 10 defines that a data flow is illegal if actors handle customer data with the status 'Celebrity' without the 'Manager' role or trust in the 'Manager' role is 'mid'. This means that only an actor with the 'Manager' role that is trusted 'high' is allowed to register celebrity customers.

Listing 8.1: The query used for the analysis of the two S2 scenarios.

```
1  actor(A),
2  inputPin(A,PIN),
3  nodeCharacteristic(A, 'Location', LOC, LOC_TRUST),
4  nodeCharacteristic(A, 'Role', ROLE, ROLE_TRUST),
5  characteristic(A, PIN, 'Origin', ORIG, ORIG_TRUST, S),
6  characteristic(A, PIN, 'Status', STAT, STAT_TRUST, S),
7  (
8    LOC = ORIG, LOC_TRUST \= ORIG_TRUST;
9    LOC \= ORIG, (ROLE \= 'Manager'; ROLE_TRUST = 'low');
10   STAT = 'Celebrity', (ROLE \= 'Manager'; ROLE_TRUST = 'mid')
11 ).
```

Setup for Q2.3

To answer the question regarding the accuracy in identifying issues that result from mismatched properties and lowered trust, we define a pair of scenarios that consists of a **S0** and a **S3** scenario. To create the **S3** scenario, we reuse the **S1** scenario of the ABAC use case, that we use for **Q2.1** and combine it with the **S2.1** scenario, that we define for **Q2.2**. The **S0** scenario is shown in Figure 8.2. Figure 8.7 shows the **S3** scenario. The 'Manager' role's reduced trust introduces trust issues in the data flows to the 'register celebrity' and 'move customer' nodes, while the added data flow to 'register customer' introduces a

property issue. For the analysis, we use the same query that has been used for the setup of Q2.2, shown in Listing 8.1.

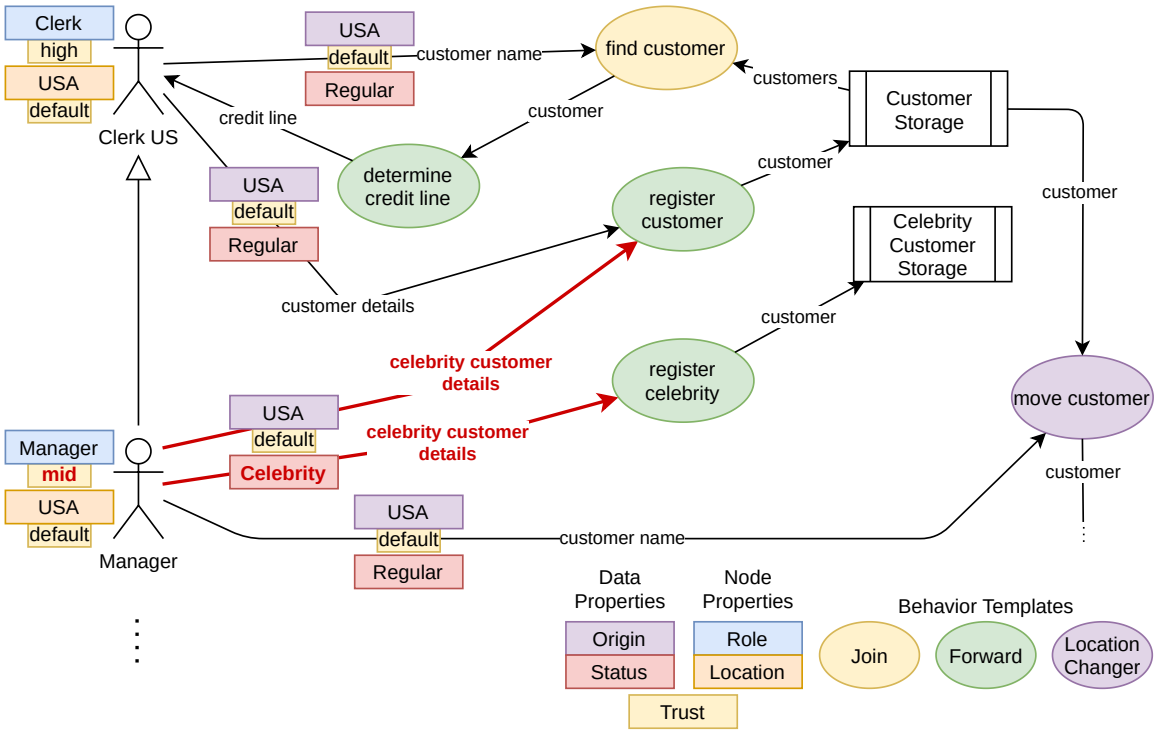


Figure 8.7: Reduced representation of the ABAC data flow diagram of the S2.3 scenario.

8.2.3 Setups for Evaluating Scalability

Each setup is run with increasing numbers, starting at 1, doubling the value till 2048 is reached. Each of these setups is executed ten times to reduce the side effects of the JVM and get a good variance when averaging the values.
All scalability setups are run on the following configuration:

- Intel i5 5200U (2 cores, 4 threads) @ 2.20 - 2.70 GHz
- 8 GB RAM
- 500 GB SSD hard drive
- Ubuntu 20.04 LTS, 64-Bit
- Java 11 (OpenJDK 64-Bit, version 11.0.11)

Setup for Q3.1

The first question regarding scalability focuses on the time required for calculating the trust labels, depending on the number of environmental factors. We utilize the FIS of the high sensitivity GPS location service we already created for the running example, see

Chapter 5, as a base.
We increase the number of inputs by adding randomly created FuzzyficationFunctions to the existing FIS. To ensure that the randomly created FuzzyficationFunctions are well-formed, we limit the FuzzyficationFunction only to be made up of between two and five triangular membership functions. We also limit the range of the FuzzyficationFunctions to start at 0 and end at a random number between 10 and 1000.
Rules are generated by combining a randomly chosen membership function of each FuzzyficationFunction. We limit these rules to only use the AND RULE_Operator. To cover every possible membership function once, we generate five rules. If every membership function of a FuzzyficationFunction with less than five membership functions is already used once, a random membership function of the FuzzyficationFunction is chosen for subsequent rules. The DefuzzyficationFunction is the same as the one we use in the running example, shown in Figure 6.4d. It is made up of three Gaussian membership functions, each representing a trust label. A random membership function of the DefuzzyficationFunction is chosen for each rule.
We initialize the FIS with the MIN AND_Operator, MAX ACCU_Operator, and center of gravity as defuzzification method.

Setup for Q3.2

The second question regarding scalability focuses on the time required to transform the model to Prolog code, depending on the number of information services. This question explores the trade-off in time associated with a more fine granular division and representation of different information services.
To be mapped to Prolog, an information service has to be linked to at least one TrustedCharacteristic. In the sense of the access control analysis, a TrustedCharacteristic represents a property that may be used in access control. This means that additionally to the increased number of information services in the model, we consequently increase the number of properties.
Based on the running example, see Chapter 5, we add information services with randomized input values. To minimize the influence of the FIS on the time needed to map to Prolog, each information service references the same FIS of the high sensitivity GPS location service of the running example. This should reduce the impact of the FIS to a constant.
For each added information service, we add a TrustedCharacteristic with the 'Location' TrustedCharacteristicType. To add the TrustedCharacteristics to a node, we assign them to each of the actors of the running example. The node behavior in our running example is only made up of the forward behavior template so that the behavior can remain unchanged.

Setup for Q3.3

The third question regarding scalability focuses on the time required to map the model to Prolog code, depending on the size of the trust enumeration of a TrustedCharacteristicType. This question explores the trade-off in time associated with more fine granular trust handling using many trust labels.

To isolate the effect of the size of the trust enumeration, we only create a single TrustedCharacteristic and InformationService. The FIS of the InformationService is made up of the tree FuzzyficationFunction inputs of our running example from Chapter 5. To match the increasing number of trust labels, we at the same time add a corresponding membership function to the DefuzzificationFunction. As we scale to a maximum of 2048 trust labels, the DefuzzificationFunction is generated with a range interval of $[0, 10240]$. This way, each membership function can cover a range of 5 without overlapping each other too much. We add Gaussian membership functions with $m = (i * 5) + 2.5$ and $\sigma = 1.5$, with i being the index of the corresponding trust label in the trust enumeration. An example of this membership function, with a reduced range of $[0, 50]$, is shown in Figure 8.8. Rules of the FIS are randomly generated in the same way described in the Setup of Q3.1. The goal is not to generate rules that cover every possible membership function of the DefuzzificationFunction, but only to generate a functioning FIS that can be evaluated to a trust label. As the FIS of our running example has three inputs with three membership functions each, we generate 27 rules, covering every combination of the membership functions. The input values of the InformationService are randomly chosen within the range of the randomly created FuzzyficationFunctions.
The remaining data flow model is based on our running example of Chapter 5. For an enumeration to be regarded in the mapping to Prolog, it needs to be assigned to a TrustedCharacteristicType. We create a new TrustedCharacteristicType, with the 'Location' enumeration of the running example as *type* and the generated trust enumeration as *trust*. We set a randomly chosen location label as the *value* and the newly created TrustedCharacteristicType as *type* of the single TrustedCharacteristic and assign it to each actor. In our running example, we only use the forward behavior template so that the node behavior can remain unchanged.

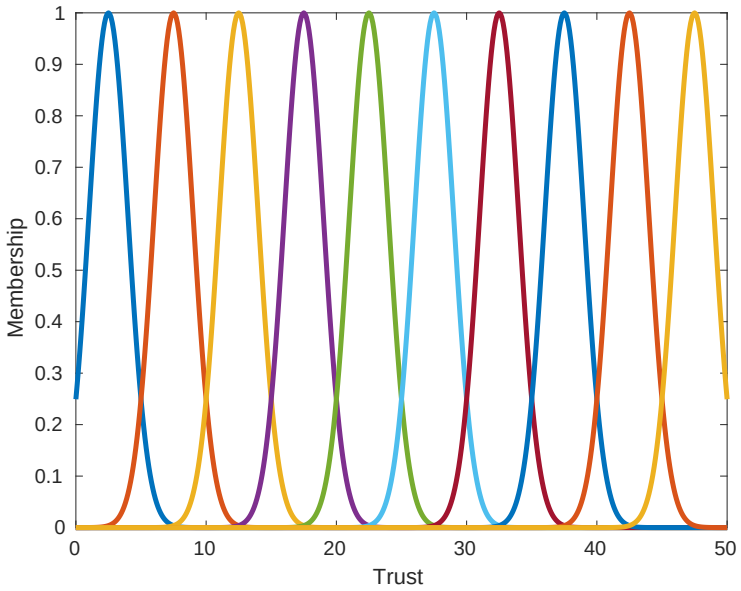


Figure 8.8: Membership function example of the fuzzy ouput regarding the setup of Q3.3.

Setup for Q3.4

The last question regarding scalability focuses on the time required to map the model to Prolog code, depending on the number of different types of properties used for access control. We utilize the data flow model of our running example as a base. For each increase in properties, we add a duplicate of the existing 'Location' TrustedCharacteristicType. For each added TrustedCharacteristicType, we duplicate the TrustedCharacteristic, which are assigned to each actor. We modify these TrustedCharacteristics to be of the added TrustedCharacteristicType. As the TrustedCharacteristicType is equal to the already existing one, the TrustedCharacteristics information services and values do not need to be changed. The node behavior in our running example is only made up of the forward behavior template so that the behavior can remain unchanged.

8.3 Evaluation Results

In this section, we present and discuss the findings of our evaluation that was defined in the previous sections.

8.3.1 Discussion on Applicability

As our evaluation regarding applicability was mostly argumentative, we discuss every aspect of the corresponding questions and try to come to a conclusion that answers the question.

Discussion on Q1.1

To answer if the information to apply our proposed concept of trust is available during design time, we focus on the availability of the information during the very early stages of the iterative development described in Section 8.2.1.
We reckon that information about sensors and environmental factors can initially be assumed and refined during the iterative development process. As we have shown with our running example (see Chapter 5) and the ABAC scenarios (see Section 8.2.2), it is possible to make some general assumptions about environmental factors from existing literature. Although the described ABAC scenarios describe a situation where the system is already deployed, we still came up with the scenarios without using actual run time details. We only used the existing ABAC DFD model of Seifermann et al. [66] and existing literature which is exactly what might be available during design time.
More precise measurements, e.g., regarding the signal damping inside the actual buildings, could be conducted, or technical specifications of actually used sensors become available during the subsequent process. This increasingly more precise knowledge can be used to refine the calculation of trust in the model. Through the iterative refinement of the model, the results of an analysis using the model can increasingly reflect the real-world situation in which the system is to be deployed.
If the DFD approach of Seifermann et al. [66] is already used to validate system requirements regarding access control, our concept of trust can be added once system requirements

change in a way that makes it necessary to handle this uncertainty. Our approach does not aim to replace the existing DFD approach but rather represents an optional step that incorporates more information in order to be able to verify requirements more accurately to the real world.

Discussion on Q1.2

To answer whether our concept of trust adds value for already deployed systems, we discuss the availability of information and describe a what-if scenario, which showcases that issues that result from changes in trust can be identified.
The handling of uncertainty we propose with our concept of trust relies on information regarding the service used to get access control properties. In order to calculate the trust in a service, there needs to be knowledge about the source, e.g., sensors and the environmental factors that influence the service. During run time, this information is already present. This enables the software architect to create a very detailed model representation of the system, which includes our concept of trust. These detailed models can be used to investigate the impact of changes from what-if scenarios on the access control of the deployed systems before implementing them.
The ABAC use case, we have described in Section 8.2.2 presents a situation where using our concept of trust is especially worthwhile for deployed systems.
In the described situation, changes in the environmental factors do not introduce new potential issues with confidentiality but rather introduce usability issues. Changes to the database used for country geolocation affect the usability of the banking service deployed in Hong Kong. The accuracy in resolving the country geolocation from the IP addresses of the actors in Hong Kong decreases drastically due to the change. With the reduced accuracy, the location property is not always properly set to Hong Kong. As a result, the access control system, which is in place, might block most data flows to the actors, effectively locking them out of accessing the Hong Kong customer storage. The original DFD approach of Seifermann et al. [66] is not able to identify this issue, as the information about the databases that are used to resolve IP addresses to geolocations is not taken into account.
With our concept of trust, this information is added to the DFD representation of the system. The changes to the database result in lower trust in the location property and therefore enable the analysis to identify the issue beforehand, preserving the usability of deployed systems.

Discussion on Q1.3

As we have described in Section 2.3 and 6.1, we use fuzzy inference systems to describe calculation rules to utilize multiple different inputs and derive a trust value. To evaluate whether FISs are suitable for the proposed use, we focus on the knowledge required to create, understand and extend a FIS:
In the book "Fuzzy sets and fuzzy logic: theory and applications," George Klir and Bo Yuan [39] address significant topics of fuzzy set theory and fuzzy logic, including fuzzy inference systems. They describe that through fuzzification, an enhanced ability to model

real-world problems is gained, lowering overall solution cost. The use of fuzziness also serves to achieve "greater capability to capture human common-sense reasoning, decision making, and other aspects of human cognition [39, pages 32f.]."
When setting up a calculation rule in general, the system's properties and environment are abstracted and simplified. As a result, information about the inputs and their influence on the result is lost. A FIS conserves more knowledge about the inputs and their influence on the calculated trust value than a conventional mathematical function by mapping inputs to membership functions and working with natural language concepts.
Additionally, a FIS "has the capability to capture and deal with meanings of sentences expressed in natural language [39, pages 32f.]," which enables, e.g., a software architect or security expert to more easily map statements or requirements about the influence of environmental factors on trust, to calculation rules. The additional knowledge, easy-to-understand linguistic concepts, and captured human decision-making allow for more informed changes or extensions to the calculation rule. Fuzzy control and fuzzy decision-making are also already relevant topics in industrial engineering. Domain experts that are involved in the creation process might already be familiar with the concept of fuzzy control/inference.
However, the quality and accuracy of the knowledge contained in a FIS depends heavily on the ability to create appropriate membership functions. As the membership functions capture linguistic concepts, their definition can be vague and context-dependent. This makes it hard to define meaningful membership functions. George Klir and Bo Yuan address this topic by describing various methods to create membership functions that carry appropriate meaning, given their context. These methods include direct and indirect methods involving one or more experts, as well as the creation of membership functions based on sample data [39, pages 280ff.]. These methods make it possible to iteratively create and increase the quality of the membership functions for the environmental factor inputs and trust output of our proposed FIS. Although the ability to iteratively create and improve the quality of the membership functions integrates well with the iterative nature of the development process (see Section 8.2.1), the methods described are rather cumbersome, time-intensive, and require good input of data or expert knowledge. However, while the initial effort to create such FISs may be higher, their understandability and extensibility make them a viable option in their proposed use.

Discussion on Q1.4

To answer if the expressiveness of the original DFD approach of Seifermann et al. [66] is affected by the addition of our concept of trust, we evaluate the expressiveness of our extended DFD approach in the same way it was done for the original approach and compare results.
To evaluate the expressiveness of their approach regarding access control, Seifermann et al. [66] define two questions:

- Is the proposed syntax capable of representing systems and their behavior relevant for access control?

- Are the proposed analysis semantics capable of defining analyses of access control violations?

To answer these questions, they select six known use cases, three of which represent the RBAC model and are taken from previous work [65]. The three remaining represent the common access control models *DAC*, *MAC* and *ABAC*. By creating instances of these use cases using the original DFD syntax, they answer the first question.
We can answer the first question in the same way, by using the *default* trust approach, we have also used to answer question **Q2.1** in Section 8.2.2. We modify the DFD instances of Seifermann et al. to use our extended DFD while still representing the identical use case. The *default* trust approach adds our concept of trust to a DFD instance but does not change the semantic of the DFD instance, effectively emulating a DFD without trust labels. Table 8.1 gives an overview of the size of the DFD instance for each use case. To

Case	Nodes	Edges	Behaviors	Characteristic Types	Labels
TravelPlanner	17	19	8	2	3
DistanceTracker	8	9	6	2	3
ContacSMS	9	12	7	2	3
DAC	7	7	6	4	4
MAC	15	22	7	2	3
ABAC	13	18	6	4	6

Table 8.1: Metrics of access control cases realized in the original DFD syntax used to evaluate expressiveness [66].

answer their second question, Seifermann et al. [66] define Prolog queries for each use case. The query for a use case can be used with the result of the transformation of the corresponding DFD instance. As we have only slightly modified the transformation to Prolog to handle our trust labels, we can replicate the queries almost exactly. Instead of the original `nodeCharacteristic` and `characteristic` facts, we use the corresponding facts we have extended with a trust label (see Section 7.2).
Being able to answer both questions in the same way as the original DFD approach suggests, that adding our concept of trust to the DFD and analysis did not impact the expressiveness. The added concept of trust, on the other hand, enables properties to incorporate information about environmental factors. This allows for more fine granular definition of access control properties and rules. The original approach allows for $\prod^{CharTypes} \#value!$ possible property combinations, where *CharTypes* is the set of characteristic types, *#value* is the number of *value* labels of the characteristic type. Including trust increases the number of possible combinations to $\prod^{TrustedCharTypes} \#value^{\#trust}!$, where *TrustedCharTypes* is the set of trusted characteristic types, *#value* is the number of *value* labels of the characteristic type and *#trust* is the number of *trust* labels of the characteristic type.
A similar result could be archived by implementing many fine granular property hierarchies. However, these properties would mix the semantics of the original property with the semantics of trust, making them less comprehensible. Through trust, the environmental

factors have a direct effect on the expressive power of the corresponding property, generally increasing the expressiveness of our approach, compared to the original approach from Seifermann et al. [66].

8.3.2 Findings and Discussion on Accuracy

In this section, we present and discuss our findings regarding the accuracy of our extended analysis. We calculate the proposed precision and recall metrics based on the findings of each question and briefly discuss the meaning of the values of both metrics.

Findings on Q2.1

We executed the model instances for the **S0** and **S1** scenarios of every use case we have described in Section 8.2.2. We correctly did not identify an issue in the six **S0** scenarios, which contain no issue. For the six corresponding **S1** scenarios, we were able to identify the contained issues correctly. Table 8.2 shows the number of issues introduced to each use case to form **S1** scenarios, compared to the number of these issues we were able to identify.
For our described metrics the number of identified issues in the **S1** scenarios represent the true positive value $t_p = 6$. We did not identify non existing issues in the **S0** scenarios resulting in the false positive value of $f_p = 0$. This results in a precision of $p = \frac{6}{6+0} = 1.0$. As shown in Table 8.2, we did not miss any of the introduced issues in the **S1** scenarios, so the false negative value amounts to $f_n = 0$. This results in a recall of $r = \frac{6}{6+0} = 1.0$. As

	S1 issues	
Case	introduced	identified
TravelPlanner	1	1
DistanceTracker	1	1
ContacSMS	1	1
DAC	2	2
MAC	3	3
ABAC	3	3

Table 8.2: Evaluated metrics for Q2.1.

we have described, our S1 scenarios and queries are semantically equal to the scenarios that were defined by Seifermann et al. when evaluating the accuracy of the original DFD approach [66]. When comparing our results to the results of Seifermann et al., we can see, that the addition of our concept of trust did not impact the ability of the DFD approach to correctly identify access control issues, that result from mismatched properties. This shows, that while we extend the ability of the DFD approach to represent and identify issues that result from trust, we did not impair the original accuracy.

Findings on Q2.2

For question Q2.2 we only base our evaluation of accuracy on the ABAC use case, described in Section 8.2.2. We divided the **S2** scenario in the two sub-scenarios **S2.1** and **S2.2**. **S2.1** introduces a trust issue regarding the role property. **S2.2** introduces a trust issue regarding the location property. We correctly did not identify an issue in the corresponding **S0** scenario, which contains no issue. For the scenarios **S2.1** and **S2.2**, we where able to correctly identify the introduced issue.
We calculate the metrics similarly to how they are calculated for the findings on Q2.1. The number of correctly identified issues in the **S2** scenarios is $t_p = 2$. We did not identify non existing issues in the **S0** scenarios and did not miss any of the two introduced issues, so the false positive value amounts to $f_p = 0$ and the false negative value amounts to $f_n = 0$. This results in a precision of $p = \frac{2}{2+0} = 1.0$ and a recall of $r = \frac{2}{2+0} = 1.0$
We did expect similar results, as property issues and trust issues both result from mismatched labels. In our approach, the way trust labels are compared is equal to how the normal property labels are compared. Bad accuracy for S1 scenarios should result in bad accuracy in S2 scenarios and vice versa.
However, as we were not able to define equivalence classes for the entirety of trust issues and test them individually, there can exist situations where this might not be the case.

Findings on Q2.3

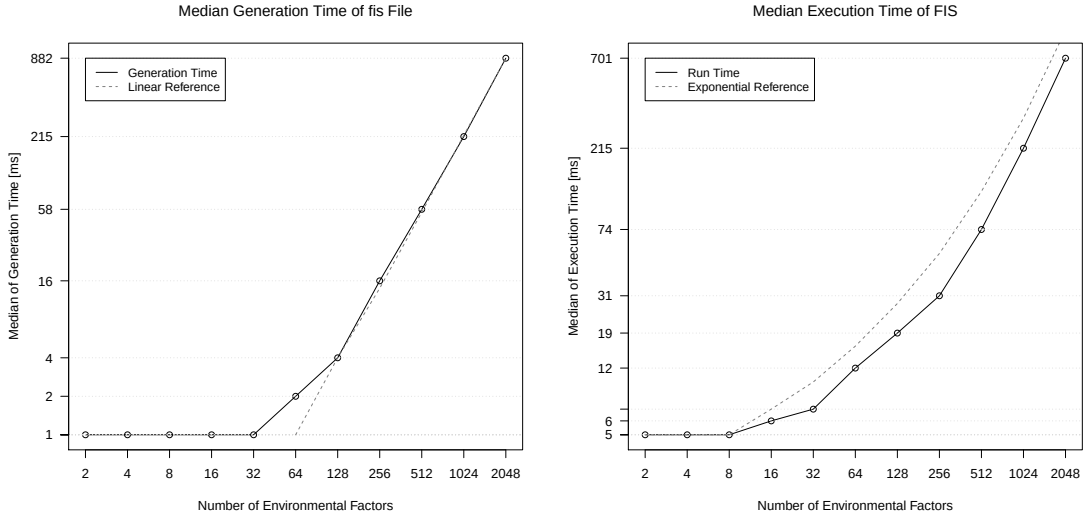
For question Q2.3 we combine the **S1** scenario of the ABAC use case with the **S2.1** scenario to form a single **S3** scenario.
As in the findings on Q2.1 and Q2.2, we correctly did not identify an issue in the corresponding **S0** scenario. In the **S3** scenario, we can exclusively identify the three issues that correspond to the **S1** scenario and the single issue that corresponds to the **S2.1** scenario. As we only use one **S3** scenario, the calculation of metrics is trivial. The precision and recall are $p = 1.0$ and $r = 1.0$.
We did expect similar results, as the analysis is already able to identify issues from mismatched properties and trust independently. As we use the same query that was used for the ABAC situations of Q2.1 and Q2.2, solutions that were found for Q2.1 can still be found, and solutions that were found for Q2.2 can still be found, effectively creating the union of both results.

8.3.3 Findings and Discussion on Scalability

In this section, we present and discuss our findings regarding the scalability of the transformation from DFD to Prolog. In each question, we scale a different aspect of the model, which we added as part of the contribution of this thesis. The measured execution times are gathered and plotted to give a better visual representation of the prevalent execution time behavior that results from the scaling the model aspect.

Findings on Q3.1

We divided our results regarding **Q3.1** into the generation of the *fis* file and the execution of the FIS. Both combined make up our process of calculating a trust label from environmental factors. Our findings are plotted, and the results are shown in Figures 8.9.
Figure 8.9a shows the time needed to generate the *fis* file. With an increasing number of environmental factors, i.e., the inputs to the FIS, we can identify a tipping point in generation time at 32 environmental factors. At this point, the nearly constant generation time behavior changes to linear behavior.
The execution time behavior of the FIS, shown in Figure 8.9b, shows exponential growth behavior with an increase of environmental factors. While the generation of the *fis* file has been fully implemented by us, we rely on third-party fuzzy inference libraries to execute the actual FIS, in our case FuzzyLite [56]. Utilizing libraries from other parties might result in varying execution time behavior. Additionally, when focusing on the actual median times of the highest iteration of our testing, we can see that the time needed to generate the *fis* file still is higher than the actual execution time of the associated FIS. This makes the generation of the *fis* file the dominant factor in the process of executing a FIS model instance with a high number of inputs. If the generation and execution time continue to grow with the observed behavior, this might change for FISs with even more inputs.



(a) Times needed to generate fis file from FIS model instance, with increasing number of inputs.

(b) Times needed to execute a fis file and calculating trust label, with increasing number of inputs.

Figure 8.9: Findings on Q3.1.

However, we believe that over 2000 environmental factors are far beyond the number of environmental factors that might be modeled during design time. With our approach, each factor would require multiple membership functions. While membership functions might be created using sample data (see Section 8.3.1), extensive rules that utilize them as inputs sill would need to be created by hand. We reckon that due to the extensive effort

involved, information services with more than 128 environmental factors will be seldom, and most will have far less. This results in a combined time for executing a FIS of mostly less than 23 ms, which is an almost negligible influence on the general running time.

Findings on Q3.2

The times needed for the transformation to and generation of the Prolog code, with an increasing number of information services and properties, are shown in Figure 8.10.
With an increase in information services and properties, we can identify that the plotted graph grows exponentially for smaller iterations but transitions into linear growth, beginning at 32 information services. This transition can be traced back to the number of information services and corresponding constant time to calculate their respective trust value becoming the dominant factor for the mapping time to Prolog. Regarding the size of the generated Prolog code, while increasing the number of information services and properties in a model increases the overall size and complexity of the model instance itself, it does not impact the size of the generated Prolog code.

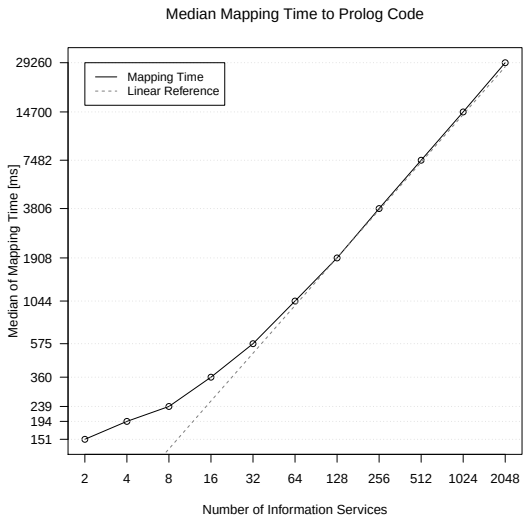


Figure 8.10: Time for the transformation to Prolog code, with increasing number information services.

Findings on Q3.3

The times needed to transform model instances with an increasing number of trust labels to Prolog code are shown in Figure 8.11. Similar to our findings on Q3.2, we can identify a point in the plotted graph where the superlinear growth in time transitions to linear growth. A big difference to our findings on Q3.2 is that the size of the trust enumeration impacts the size of the generate Prolog code radically. This was expected, as each added label is transformed to its own `characteristicTypeTrust(CT, T, I)` Prolog fact, as we describe in Section 7.2. Each additional trust label also adds lines to the Prolog representation of the behavior of the nodes, as each assignment of each behavior

adds a `characteristic(N, PIN, CT, V, T, S, VF) :- ...` rule for each possible value and trust combination of each characteristic type.

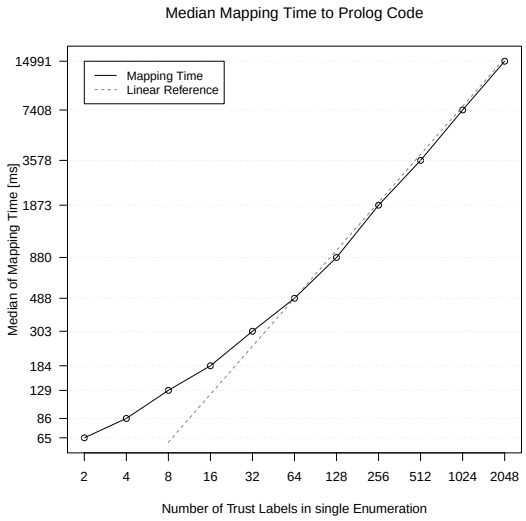


Figure 8.11: Time for the transformation to Prolog code, with increasing size of trust enumeration.

Similar to what we describe in our findings on Q3.1, we believe that defining over 2000 individual trust values for a single characteristic type is far beyond what might be modeled during design time. However, as the number of trust labels only influences the execution time in a linear manner, even a very fine-grained definition of trust labels should not result in excessive execution times of the mapping.

Findings on Q3.4

The times needed for the transformation and generation of Prolog code, with an increasing number of types of properties, are shown in Figure 8.12. Looking at Figure 8.12, we can identify, that overall the execution time behavior of the transformation to Prolog shows linear growth when increasing the number of instances of *TrustedCharacteristicType*.
Similar to our findings on Q3.3, the size of the Prolog code is impacted radically, as each instance of *TrustedCharacteristicType* is transformed to the Prolog facts described in Listing 7.1. For our setup, with three value labels and three trust labels, each added TrustedCharacteristicType adds seven additional Prolog facts.

Generally, with the findings for Q3.2, Q3.3, and Q3.4, we can conclude that the model aspects we add as part of our approach only influence the execution time of the mapping to Prolog in a linear manner. We believe that this linear influence will not negatively impact the overall execution time behavior of the original mapping to Prolog. The original mapping also increases the size of the mapped Prolog code for each additional model element, which should also result in a linear execution time behavior.

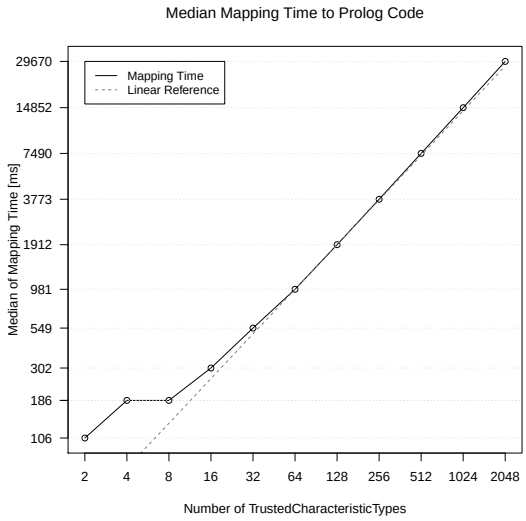


Figure 8.12: Time for the transformation to Prolog code, with an increasing number of types of properties.

8.4 Threats to Validity

As the evaluation of our approach is partly evaluated with case studies, we discuss the internal validity, external validity, construct validity, and reliability of our contribution, as characterized by Runeson, Höst, Austen, and Regnell [61].
As our evaluation goals are structured in applicability, accuracy, and scalability, we discuss our threads to validity for each of our goals separately.

Internal validity ensures that causal relations are valid, i.e., the factor that is expected to have an influence is the only influencing factor.
The main threat to the internal validity of our evaluation of applicability is whether or not our chosen questions and discussed results hold enough weight to make a proper statement about applicability. The discussed additional value of our contribution for run time systems in question Q1.2 highly depends on the system and its area of application. However, as we have discussed in the evaluation, we still were able to create multiple problematic real-world situations, using actual values of environmental factors taken from existing literature. This also mitigates the thread regarding Q1.1, as we were able to identify applicable environmental factors for various situations from existing literature, using only the described use cases as a base.
As the setup of our evaluation of accuracy is strongly based on the evaluation of the DFD approach of Seifermann et al. [66], the same factors that influence the internal validity still hold: Analysis queries or DFDs that are incorrect can still yield the expected result, which might be an identified violation or not. We address this issue by defining three distinct types of issues in our S1, S2, and S3 scenarios as well as the corresponding S0 scenarios without issues and analyze them with the same query. Queries that are overly customized to find a specific issue, e.g., by encoding the violation that is to be identified directly in the query, can positively impact the accuracy. To combat this threat, we use

the same queries for the evaluation of Q2.1 that have been used by Seifermann et al.. To avoid these overfitted queries, for questions Q2.2 and Q2.3, we extend the access control query of the ABAC use case to directly take our concept of trust into account and use this distinct query to evaluate the three situation pairs of questions Q2.2 and Q2.3. A threat to the internal validity of our scalability evaluation using different base models instances for each scaled aspect of a model. Some base model setups can positively impact the execution time on some test cases because they omit the parts of the transformation algorithm that are impacted by the scaled aspect of the model. Also, using different base model instances makes it impossible to properly compare the resulting execution time behavior of different scaled model aspects. To combat this threat, we use the same model instance as a base for Q3.2, Q3.3, and Q3.4 of our scalability evaluation. With a uniform base, the influence of the remaining aspects of the model remains constant and does not affect the overall observed behavior. We also focus on a single aspect for each question, explicitly only scaling the model elements necessary for the scaled model aspect to be taken into account by the transformation algorithm.

External validity ensures that the findings are only generalized if they can validly be applied to other situations, groups, or events.
For our evaluation of applicability, a threat to validity is the author's bias towards his own contribution and his own limited experience regarding real-world access control setups. However, we base most of our statements on existing literature. Moreover, our evaluation of applicability already contains a discussion about external validity, as the proposition of applicability already covers how far something can be generalized or holds value for others.
The main threat to external validity of our accuracy evaluation stems from the small number of S2 and S3 scenarios we cover when answering questions Q2.2 and Q2.3. As we have discussed multiple times throughout this thesis, at the time of writing this thesis, there is little existing literature that describes use cases that cover all details that are necessary to apply our concept. This lack of usable information makes it impossible to derive meaningful equivalence classes for trust and evaluate the accuracy of our analysis in finding issues with trust in a structured way. We still try to partly mitigate this thread by defining our S2 and S3 scenarios using real-world environmental factors and measurements from exiting literature to create appropriate use cases, removing our bias towards using solely made-up values that positively influence the accuracy. Additionally, the S2 and S3 scenarios are based on the ABAC use case of Seifermann et al. [66]. As we have described in Chapter 4, our concept of trust is based around the idea of trust relationships and attribute assurance, described in the NIST publication defining ABAC [31], [32]. By adding our concept of trust to the ABAC use case, we can show that we are not only able to represent but also accurately identify issues in trust chains during design time, as well as analyze whether the availability of appropriate attributes is assured.
Another threat to external validity are the selected accuracy values we base the setup of question **Q2.2** on. We take the geolocation accuracy values for the S2.2 scenario from the publications [25], [54], [69]. The geolocation database data used by these publications is from 2016 or older. According to the statement on the website of MaxMind-GeoLite [45], their databases are updated weekly. As we have no way of reevaluating the results of our

referenced publications with current databases, the problem described in this setup might not exist anymore or be less drastic than described. However, the described situation still presented an issue in 2016, and issues of a similar nature might still occur.

Construct validity assures that our selected metrics can answer our research questions and that the questions contribute to the defined evaluation goals. For our metrics, we chose a discussion to rate applicability, precision, and recall to rate accuracy and the execution time to rate scalability. Using metrics to summarize the applicability of an approach is not sufficiently possible, as the applicability is generally very dependent on variations and limitations of the actual system an approach is applied at. We still discuss the applicability during two general phases of software development, design time and run time, which every production system undergoes. The included discussion about expressiveness is fully based on the evaluation of the original DFD approach our contribution is based on. Consequently, we share the same threads to validity, which have already been discussed by Seifermann et al. [66].
The precision and recall metrics are common for evaluating accuracy and are used evaluate related work [4], [66].
Measuring the execution time for tasks with increasing size and discussing the observed execution time behavior is also a common way to evaluate the scalability. Especially for approaches in the domain of system architecture modeling, where the formulated problems can vary in size drastically, for example [27]. In our evaluation of scalability, we only focus on the mapping of DFD instances to Prolog code and not the actual execution of an analysis by running a query on the resulting Prolog code. As we have already briefly discussed in Section 8.1.3, the execution time of the Prolog code depends on other factors that are more closely related to the original DFD approach of Seifermann et al. [66] than to our approach of this thesis, like the general structure of the data flow diagram instance, the kind of executed query, the Prolog library and optimization techniques that were used. As we only base the implementation of our approach on the DFD approach of Seifermann et al., we believe that evaluating the impact of these more general factors on the scalability of running the analysis is out of scope for this thesis.

Reliability assures that other researchers can repeat the evaluation and come to the same results. As discussed before, our discussion regarding applicability is based on existing literature. Still, our discussion but might still leave room for further interpretation, which is an issue we can not directly address. To mitigate any remaining threads regarding the reliability of our evaluation, we publish our implementation and model extension as well as all model instances of every analyzed scenario in our data set [9]. This allows others to reproduce the results of our accuracy and scalability evaluation.

8.5 Assumptions and Limitations

We distinguish between our assumptions and limitations of our proposed concept of trust, as well as of our evaluation:
Regarding our concept of trust, we make some assumptions regarding the environmental factors and their influence in the calculation of trust. Generally, we assume that developers

have to know which environmental factors influence trust of a service or have a process of identifying environmental factors and their corresponding influence on trust. While adding newly emerging environmental factors to the calculation can be done with little effort, as we have discussed in Section 8.3.1, knowledge about the environmental factors has to already exists. We also assume that all environmental factors are quantifiable and can be taken into account by the trust calculation. Additionally, linguistic concepts or membership functions for environmental factors have to already exist, or it is always possible to apply one of the processes described in Section 8.3.1 to derive fitting membership functions, i.e., either expert knowledge or sample data exists and can be used.
A limitation of our approach is that there is no way to draw conclusions based on a trust label about which exact environmental factors of which InformationService have led to the trust label. Different information services can provide the same property, e.g., a location in a building, but be influenced by different environmental factors and in different ways. The characteristics associated with these information services have the same characteristic type, which makes the, e.g., location properties and trust comparable with each other. It is therefore not necessarily possible to tell based on a trust label from which InformationService it originated and thus which exact environmental factors had which influence.
We make two assumptions in our evaluation that do not directly pose a threat to validity but need mentioning for the sake of completeness:
For our evaluation, we assume that the architectural model always describes the software or run time system adequately. For our approach, this means that the model includes the correct specifications for every information service and that changes in the software or run time system are correctly propagated to the model. If this is not the case, transferring conclusions from an analysis based on the model to the software or runtime system could lead to errors.
With existing approaches which use system monitoring data to align architectural models with the run time system, such as iObserve [27], this assumption could be achieved with less effort.

8.6 Data Availability

All data of this thesis is made publicly available in our data set [9]. We include our data flow diagram extension, extended Prolog transformation algorithm, and all test implementations or model instances used for the evaluation.

9 Conclusion

To conclude this thesis, we summarize our contribution in Section 9.1 and give an outlook on future work in Section 9.2. Finally, we express our acknowledgments in Section 9.3.

9.1 Summary

In this thesis, we proposed our approach to handle uncertainty in access control during design time. We defined a characterization of uncertainties in access control on the architectural level to provide a better understanding of the kinds of uncertainty that can be present. By defining our concept of trust, we raise awareness about existing uncertainty, enabling the handling of the corresponding uncertainty. Adding additional information to design time software architecture models and providing a way to analyze model instances for access control violations enables software architects to further increase the quality of models and verify requirements regarding access control under uncertainty in the early stages of the software development process.
We defined our concept of trust as a composition of environmental factors, e.g., environmental conditions of nature, age of information, and used hardware of a modeled system, that impact the validity of and consequently trust in access control properties. To combine the environmental factors, we proposed the use of fuzzy inference systems as a way of calculating trust values for access control properties.
We extended the existing DFD approach of Seifermann et al. [66] to design time information flow and access control analysis. The existing approach uses a transformation algorithm to map instances of a DFD metamodel to a Prolog logic program and uses queries to analyze the DFD and detect violations. Access control properties are represented by labels, which are propagated along the flow of data. We extended the existing DFD metamodel by explicitly representing services that supply access control properties. Each information service has a trust associated with them, which different environmental factors can influence in different ways. To describe how environmental factors are mapped to trust values, we proposed the use of fuzzy inference systems as a way of defining calculation rules. We introduced trust labels as a way of representing trust in the DFD metamodel. We further extended the existing transformation to Prolog by adding additional label propagation logic for our trust labels.
In our evaluation, we discussed the overall applicability of our proposed concept of trust. Using manually created scenarios and values for environmental factors we extract from existing literature, we showed that our concept of trust can generally be applied to system models during design and run time of the system. We also showed that there can be situations even after the system has already been deployed where the addition of our concept of trust to the system model provides additional value. We show that our use of

FISs as a way of defining a mapping from environmental factors to a trust value makes it more natural for humans to define these kinds of calculation rules and holds additional information, which aids in the extensibility of existing FISs. Additionally, we compared the expressiveness of the existing DFD approach in representing systems and defining analyses for access control, with and without our extension, which showed that our extension did not impact expressiveness.
We also evaluated the accuracy of our approach in correctly identifying access control issues. We defined three classes of issues that result in an access control issue and showed that our approach could correctly identify each issue. Finally, for our evaluation, we observed the execution time behavior of the mapping from DFD to Prolog. We scaled the size of the model aspects that were added by our approach. This showed that even for large systems with very detailed trust definitions, the mapping to Prolog only exhibits linear execution time behavior.

9.2 Future Work

While working on this thesis, we identified three points of future work.
One of the biggest obstacles throughout the work on the thesis was the poor supply of publications dealing with similar problems that could be applied to our contribution. Further research regarding real-world scenarios or use cases regarding access control properties and uncertainty could improve the quality of the evaluation of applicability and accuracy of our approach. Additionally, this would not only help to provide a better base for other publications in this field but could also help to better identify and address actual real-world issues.
Our contribution still requires the software architect to create the Prolog query data flow constraints by hand and have knowledge about the structure of the Prolog program that results from the transformation of a DFD instance. Similar to our contribution, the data flow constraints approach of Hahner et al. [26] is also integrated with the DFD approach of Seifermann et al. [66]. The use of the domain-specific language (DSL) for modeling data flow constraints would simplify the process of defining constraints and reduce the need to know details about the actual mapping to Prolog.
While working on this thesis, we presented our approach in the context of a meeting of the members of the FluidTrust research project [28]. The following discussion centered around the FISs and the possibility of applying machine learning to increase the quality of the FISs overall.
For the membership functions of the inputs and outputs of the FISs, there are already existing approaches that utilize sample data [39, 290ff.], which could be adapted to machine learning. However, while it conceptually should be possible to create the rules for a FIS of an InformationService using machine learning, some major problems and limitations need to be considered. As an InformationService can generally supply any kind of property for the access control model, the data required to learn rules might be different for each kind of property. This is a problem, as machine learning is very dependent on the amount of available and prepared data. Additionally, if a new environmental factor is added to a FIS, the whole rule set needs to be relearned completely, eliminating the normally easy

extensibility of a FIS. While the described problems exist, the possibility of using machine learning with our contribution should be further researched.

9.3 Acknowledgments

I want to thank the members of the FluidTrust research project for their comprehensive feedback, which provided an outside view on our approach. I especially want to thank Stephan Seifermann for his inputs and advice regarding the creation and analysis of models using his DFD approach. Finally, I want to thank my advisors, Maximilian Walter, and Sebastian Hahner, for their continuous input, support, and feedback throughout the time of this thesis.

Bibliography

[1] Khaled Alghathbar, Csilla Farkas, and Duminda Wijesekera. "Securing UML information flow using FlowUML". In: *Journal of Research and Practice in Information Technology* 38.1 (2006), pp. 111–120.

[2] Claudio A. Ardagna et al. "Supporting location-based conditions in access control policies". In: *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, ASIACCS '06* 2006 (2006), pp. 212–222.

[3] Phillip G Armour. "The five orders of ignorance". In: *Communications of the ACM* 43.10 (2000), pp. 17–20.

[4] Steven Arzt et al. "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps". In: *Acm Sigplan Notices* 49.6 (2014), pp. 259–269.

[5] Nitin A Bansod, Marshall Kulkarni, and SH Patil. "Soft computing-a fuzzy logic approach". In: *Soft Computing* 73 (2005).

[6] Victor R Basili, Gianluigi Caldiera, and H Dieter Rombach. "The goal question metric approach". In: *Encyclopedia of Software Engineering* 2 (1994), pp. 528–532.

[7] Steffen Becker, Heiko Koziolek, and Ralf Reussner. "Model-Based performance prediction with the palladio component model". In: *Proceedings of the 6th International Workshop on Software and Performance, WOPS'07*. WOSP '07. ACM, 2007, pp. 54–65.

[8] Matt Blaze, Joan Feigenbaum, and Martin Strauss. "Compliance checking in the policymaker trust management system". In: *International Conference on Financial Cryptography*. Springer. 1998, pp. 254–274.

[9] Nicolas Boltz. *Architectural Uncertainty Analysis for Access Control Scenarios in Industry 4.0 - Data Set*. 2021. DOI: 10.5281/zenodo.5093176. URL: https://doi.org/10.5281/zenodo.5093176.

[10] Nicolas Boltz, Maximilian Walter, and Robert Heinrich. "Context-based confidentiality analysis for industrial iot". In: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE. 2020, pp. 589–596.

[11] Hugh Boyes et al. "The industrial internet of things (IIoT): An analysis framework". In: *Computers in Industry* 101 (2018), pp. 1–12.

[12] Max Bramer. *Logic programming with Prolog*. Vol. 9. Springer, 2005.

[13] Tomas Bures et al. "Capturing Dynamicity and Uncertainty in Security and Trust via Situational Patterns". In: *9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020*. Vol. October. 2020.

[14] Swati Chaudhari and Manoj Patil. “Study and Review of Fuzzy Inference Systems for Decision Making and Control”. In: *ISSN ISSN (CD-ROM American International Journal of Research in Science Technology, Engineering & Mathematics AIJRSTEM* (2014), pp. 2328–3491.

[15] Pau Chen Cheng et al. “Fuzzy Multi-Level Security: An experiment on quantified risk-adaptive access control”. In: *Proceedings - IEEE Symposium on Security and Privacy* (2007), pp. 222–227.

[16] Frédéric Cuppens and Alexandre Miege. “Modelling contexts in the Or-BAC model”. In: *19th Annual Computer Security Applications Conference, 2003. Proceedings.* IEEE. 2003, pp. 416–425.

[17] Maria Luisa Damiani et al. “GEO-RBAC: a spatially aware RBAC”. In: *ACM Transactions on Information and System Security (TISSEC)* 10.1 (2007), 2–es.

[18] Tom DeMarco. “Structure analysis and system specification”. In: *Pioneers and Their Contributions to Software Engineering*. Springer, 1979, pp. 255–288.

[19] John R. Douceur. “The sybil attack”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2429 (2002), pp. 251–260.

[20] Bernd Eissfeller, Andreas Teuber, and Peter Zucker. “Indoor-GPS: ist der Satellitenempfang in Gebäuden möglich”. In: *ZfV, Zeitschrift für Vermessung* 4 (2005), p. 130.

[21] Michalis Famelis and Marsha Chechik. “Managing design-time uncertainty”. In: *Software and Systems Modeling* 18.2 (2019), pp. 1249–1284.

[22] David Ferraiolo, D Richard Kuhn, and Ramaswamy Chandramouli. *Role-based access control.* Artech House, 2003.

[23] David F. Ferraiolo et al. “Proposed NIST Standard for Role-Based Access Control”. In: *ACM Transactions on Information and System Security* 4.3 (2001), pp. 224–274.

[24] Robert Fullér. “Neural fuzzy systems”. In: (1995).

[25] Manaf Gharaibeh et al. “A look at router geolocation in public and commercial databases”. In: *Proceedings of the 2017 Internet Measurement Conference*. 2017, pp. 463–469.

[26] Sebastian Hahner et al. “Modeling Data Flow Constraints for Design-Time Confidentiality Analyses”. In: *2021 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2021, pp. 15–21.

[27] Robert Heinrich. “Architectural runtime models for integrating runtime observations and component-based models”. In: *Journal of Systems and Software* 169 (2020).

[28] Robert Heinrich et al. *FluidTrust*. 2021. URL: `https://fluidtrust.ipd.kit.edu/`.

[29] Urs Hengartner and Zhong Ge. “Distributed, uncertainty-aware access control for pervasive computing”. In: *Proceedings - Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2007* (2007), pp. 241–246.

[30] Hilary H. Hosmer. “Using fuzzy logic to represent security policies in the multipolicy paradigm”. In: *ACM SIGSAC Review* 10.4 (1992), pp. 12–21.

[31] Vincent C Hu et al. “Attribute-Based Access Control”. In: *IEEE Computer* 48.2 (2015), pp. 85–88.

[32] Vincent C. Hu et al. “Guide to Attribute Based Access Control (ABAC) Definition and Considerations”. In: *NIST Special Publication* 800.162 (2014).

[33] Asim Iqbal et al. “An overview of the factors responsible for GPS signal error: Origin and solution”. In: *International Conference on Wireless Networks and Information Systems, WNIS 2009* 1 (2009), pp. 294–299.

[34] ISO. *ISO IEC 61131 – Part 7: Fuzzy Control Programming*. Standard. International Electrotechnical Commission, Oct. 2001.

[35] J-SR Jang. “ANFIS: adaptive-network-based fuzzy inference system”. In: *IEEE transactions on systems, man, and cybernetics* 23.3 (1993), pp. 665–685.

[36] Jan Jürjens and Pasha Shabalin. “Automated verification of UMLsec models for security requirements”. In: *International Conference on the Unified Modeling Language*. Springer. 2004, pp. 365–379.

[37] A. A.E. Kalam et al. “Organization based access control”. In: *Proceedings - POLICY 2003: IEEE 4th International Workshop on Policies for Distributed Systems and Networks* (2003), pp. 120–131.

[38] Kuzman Katkalov et al. “Model-driven development of information flow-secure systems with IFlow”. In: *2013 International Conference on Social Computing*. IEEE. 2013, pp. 51–56.

[39] George Klir and Bo Yuan. *Fuzzy sets and fuzzy logic*. Vol. 4. Prentice hall New Jersey, 1995.

[40] Heiko Koziolek and Ralf Reussner. “A model transformation from the palladio component model to layered queueing networks”. In: *SPEC International Performance Evaluation Workshop*. Springer. 2008, pp. 58–78.

[41] Ioanna Lytra and Uwe Zdun. “Supporting architectural decision making for systems-of-systems design under uncertainty”. In: *1st ACM SIGSOFT/SIGPLAN International Workshop on Software Engineering for Systems-of-Systems, SESoS 2013 Proceedings* (2013), pp. 43–46.

[42] Abhishek Majumder, Suyel Namasudra, and Samir Nath. *Taxonomy and Classification of Access Control Models for Cloud Environments*. 2014, pp. 55–99.

[43] Ebrahim H Mamdani and Sedrak Assilian. “An experiment in linguistic synthesis with a fuzzy logic controller”. In: *International journal of man-machine studies* 7.1 (1975), pp. 1–13.

[44] Inc. Math Works. *Fuzzy Logic Toolbox for Use with MATLAB*. MathWorks, 2001.

[45] *MaxMind-GeoLite2*. 2021. URL: `https://dev.maxmind.com/geoip`.

[46] Amir H Meghdadi and M-R Akbarzadeh-T. "Probabilistic fuzzy logic and probabilistic fuzzy systems". In: *10th IEEE International Conference on Fuzzy Systems.(Cat. No. 01CH37297)*. Vol. 3. IEEE. 2001, pp. 1127–1130.

[47] Philipp Meier, Samuel Kounev, and Heiko Koziolek. "Automated transformation of component-based software architecture models to queueing petri nets". In: *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE. 2011, pp. 339–348.

[48] *NetAcuity*. 2021. URL: https://www.digitalelement.com/solutions/ip-location-targeting/netacuity/.

[49] Sven Peldszus et al. "Secure Data-Flow Compliance Checks between Models and Code based on Automated Mappings". In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE. 2019, pp. 23–33.

[50] Carlos-Andrés Peña-Reyes and Moshe Sipper. "Fuzzy CoCo: Balancing Accuracy and Interpretability of Fuzzy Models by Means of Coevolution". In: *Studies in Fuzziness and Soft Computing* (2003), pp. 119–146.

[51] Diego Perez-Palacin and Raffaela Mirandola. "Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation". In: *ICPE 2014 - Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering* (2014), pp. 3–14.

[52] Marie Christin Platenius. "Fuzzy matching of comprehensive service specifications". PhD thesis. Universität Paderborn, 2016, p. 262.

[53] Marie Christin Platenius et al. "Matching of Incomplete Service Specifications Exemplified by Privacy Policy Matching". In: *Communications in Computer and Information Science*. Vol. 508. September. 2015, pp. 6–17.

[54] Ingmar Poese et al. "IP geolocation databases: Unreliable?" In: *ACM SIGCOMM Computer Communication Review* 41.2 (2011), pp. 53–56.

[55] David M. W. Powers. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation". In: *Journal of Machine Learning Technologies* 2.1 (2020), pp. 37–63.

[56] Juan Rada-Vilela. *The FuzzyLite Libraries for Fuzzy Logic Control*. 2018. URL: https://fuzzylite.com/.

[57] Qasim Mahmood Rajpoot, Christian Damsgaard Jensen, and Ram Krishnan. *Attributes Enhanced Role-Based Access Control Model*. Vol. 14. 1. 2015, pp. 16–17.

[58] Indrakshi Ray, Mahendra Kumar, and Lijun Yu. "LRBAC: A Location-Aware Role-Based Access Control Model". In: *International Conference on Information Systems Security, ICISS 2006*. Vol. 1716. Springer, Berlin, Heidelberg, 2006, pp. 147–161.

[59] Ralf H Reussner et al. *Modeling and Simulating Software Architectures – The Palladio Approach*. Cambridge, Massachusetts: MIT Press, 2016, p. 377.

[60] Vasja Roblek, Maja Meško, and Alojz Krapež. "A Complex View of Industry 4.0". In: *SAGE Open* 6.2 (2016).

[61] Per Runeson et al. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons, 2012.

[62] Mustapha Ben Saidi, AA Elkalam, and Abderrahim Marzouk. "TOrBAC: A trust organization based access control model for cloud computing systems". In: *International Journal of Soft Computing and Engineering* 2.4 (2012), pp. 122–130.

[63] Ravi S Sandhu. "Role-based access control". In: *Advances in computers*. Vol. 46. Elsevier, 1998, pp. 237–286.

[64] Stephan Seifermann. "Architectural data flow analysis". In: *Proceedings - 2016 13th Working IEEE/IFIP Conference on Software Architecture, WICSA 2016*. WICSA'16. IEEE, 2016, pp. 270–271.

[65] Stephan Seifermann, Robert Heinrich, and Ralf Reussner. "Data-driven software architecture for analyzing confidentiality". In: *Proceedings - 2019 IEEE International Conference on Software Architecture, ICSA 2019*. IEEE, 2019, pp. 1–10.

[66] Stephan Seifermann et al. "Detecting Violations of Access Control and Information Flow Policies in Data Flow Diagrams". In: *Journal of Systems and Software* (2021). Submitted.

[67] Stephan Seifermann et al. *Evaluation data set*. 2021. URL: `https://figshare.com/s/6379cecec327eb627aaf`.

[68] Minakshi Sharma and Sourabh Mukharjee. "Brain tumor segmentation using genetic algorithm and artificial neural network fuzzy inference system (ANFIS)". In: *Advances in Computing and Information Technology*. Springer, 2013, pp. 329–339.

[69] Yuval Shavitt and Noa Zilberman. "A geolocation databases study". In: *IEEE Journal on Selected Areas in Communications* 29.10 (2011), pp. 2044–2056.

[70] Nimalaprakasan Skandhakumar et al. "Graph theory based representation of building information models for access control applications". In: *Automation in Construction* 68 (2016), pp. 44–51.

[71] Graeme Smith. *The Object-Z specification language*. Vol. 1. Springer Science & Business Media, 2012.

[72] Gregor Snelting et al. "Checking probabilistic noninterference using JOANA". In: *it-Information Technology* 56.6 (2014), pp. 280–287.

[73] Kurt Stenzel et al. "Declassification of Information with Complex Filter Functions." In: *ICISSP*. 2016, pp. 490–497.

[74] İsmail Üstün et al. "A comparative study of estimating solar radiation using machine learning approaches: DL, SMGRT, and ANFIS". In: *Energy Sources, Part A: Recovery, Utilization, and Environmental Effects* 0.0 (2020), pp. 1–24.

[75] W.E. Walker et al. "Defining Uncertainty: A Conceptual Basis for Uncertainty Management in Model-Based Decision Support". In: *Integrated Assessment* 4.1 (2003), pp. 5–17.

[76] Mark J Wierman. "An Introduction to the Mathematics of Uncertainty: including Set Theory, Logic, Probability, Fuzzy Sets, Rough Sets, and Evidence Theory". In: *Creighton University. Retrieved* 16 (2016).

[77] Man Zhang, Shaukat Ali, and Tao Yue. "Uncertainty-wise test case generation and minimization for Cyber-Physical Systems". In: *Journal of Systems and Software* 153 (2019), pp. 1–21.

[78] Man Zhang et al. "Uncertainty-Wise Cyber-Physical System test modeling". In: *Software and Systems Modeling* 18.2 (2019), pp. 1379–1418.

[79] Man Zhang et al. "Understanding uncertainty in cyber-physical systems: A conceptual model". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9764. Springer Verlag, 2016, pp. 247–264.