CHALMERS
UNIVERSITY OF TECHNOLOGY

Data Minimization in Distributed Applications for More Privacy

Master's thesis in Algorithms, Languages and Logic

JAKOB BOMAN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016:NN

Data Minimization in Distributed Applications for More Privacy

JAKOB BOMAN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Division of Software Technology
Chalmers University of Technology
Gothenburg, Sweden 2016

Data Minimization in Distributed Applications for More Privacy
JAKOB BOMAN

Supervisor: Thibaud Antignac, Software Engineering
Examiner: Wolfgang Ahrendt, Software Engineering

Master's Thesis 2016:NN
Department of Computer Science and Engineering
Division of Software Technology
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2016

Data Minimization in Distributed Applications for More Privacy
JAKOB BOMAN
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

The presence of connected devices in our environment is increasing. These devices form a network often called Internet of Things (or IoT for short), where everything from light-bulbs to thermostats can be controlled by an app or by another device. These services make a lot of that data available to the end user but also to malicious parties due to the devices leaking more data than intended or by bad design. This puts the end user at risk, violating its privacy and leaking sensitive data. One simple and obvious way to prevent leakages and misuses of personal data is to collect less of this data, a principle known as data minimization. However, this solution is rarely used in practice because of business models relying on personal data harvest on one hand and because of the difficulty to enforce it once it is defined what is actually needed to provide a service.

Keywords: *some keywords will be added here*

Acknowledgements

First of all I want to thank David Frisk for this outstanding LaTeX-template I used for my master thesis.
of course others will be thanked as well

Jakob Boman, Gothenburg, April 8, 2017

Contents

List of Figures

List of Tables

Listings

1
Introduction

This chapter will give a brief introduction to the thesis, discuss the motivation for why this thesis is relevant and then also discuss the aims sought to be achieved at the end.

1.1 Motivation

The presence of connected devices in our environment is increasing. These devices form a network often called Internet of Things (or IoT for short), where everything from lightbulbs to thermostats can be controlled by an app or by another device. These services make a lot of that data available to the end user but also to malicious parties due to the devices leaking more data than intended or by bad design. This puts the end user at risk, violating its privacy and leaking sensitive data.
One simple and obvious way to prevent leakages and misuses of personal data is to collect less of this data, a principle known as data minimisation. However, this solution is rarely used in practice because of business models relying on personal data harvest on one hand and because of the difficulty to enforce it once it is defined what is actually needed to provide a service.
Privacy is utterly important for the development of IoT applications, Miorandi et al. gives several reasons: "The main reasons that makes privacy a fundamental IoT requirement lies in the envisioned IoT application domains and in the technologies used. Healthcare applications represent the most outstanding application field, whereby the lack of appropriate mechanisms for ensuring privacy of personal and/or sensitive information has harnessed the adoption of IoT technologies."

insert ref

1.2 Aim

This thesis will investigate ways to improve privacy in a special kind of IoT (Internet of Things) devices known as Wireless Sensor Networks (WSN). WSN are networks of autonomous sensors and actuators. The goal to enhance privacy for this kind of devices will be addressed by relying on data minimization. This means the project sought to improve privacy in distributed networks by limiting the amount of personal data being processed.
To achieve this, the project sought to accomplish the following steps:

- Construct models of a Wireless Sensor Networks that illustrates Over-Collection.
- Investigate Over-Collection and it's relation to Data Minimization, by use of the models, and how it can be prevented.

- Implement a solution from the models and analyze results.

1.3 Limitations

The project will not consider faulty behaviors of a Wireless Sensor Network , meaning that the systems and algorithms will work under the assumption that all messages sent are received and all units are working as intended without malfunctions. Only the result of the data collection will be analyzed in the sought outcome and if time complexity of the algorithm will be an issue for the project, it will not be considered as a failure should it arise. Some analysis will be done but it won't be a main focus to minifor the project.
Only the privacy aspects of collecting personal data will be considered throughout the thesis and the aspect of storing and managing it will be outside the scope of this thesis.
Any model properties related to time (in the sense that they can be measured numerically) will be treated on an abstract level or be disregarded.

this section is borrowed from another MT where a quote was also included from Ben-Ari's book from 2008. I thought it was relevant for me aswell

1.4 Thesis Structure

saving for later when the thesis has shaped up

2
Background

In this chapter we introduce the basic concepts that are the focus of this thesis, such as Wireless Sensor Networks and *Data Minimization*.

2.1 Wireless Sensor Network (WSN)

A Wireless Sensor Network is an improvement from the traditional sensor networks, made possible by advances in micro-electro-mechanical systems (MEMS) technology making sensor nodes that are smaller, multifunction and cheaper in comparison to previous sensors.

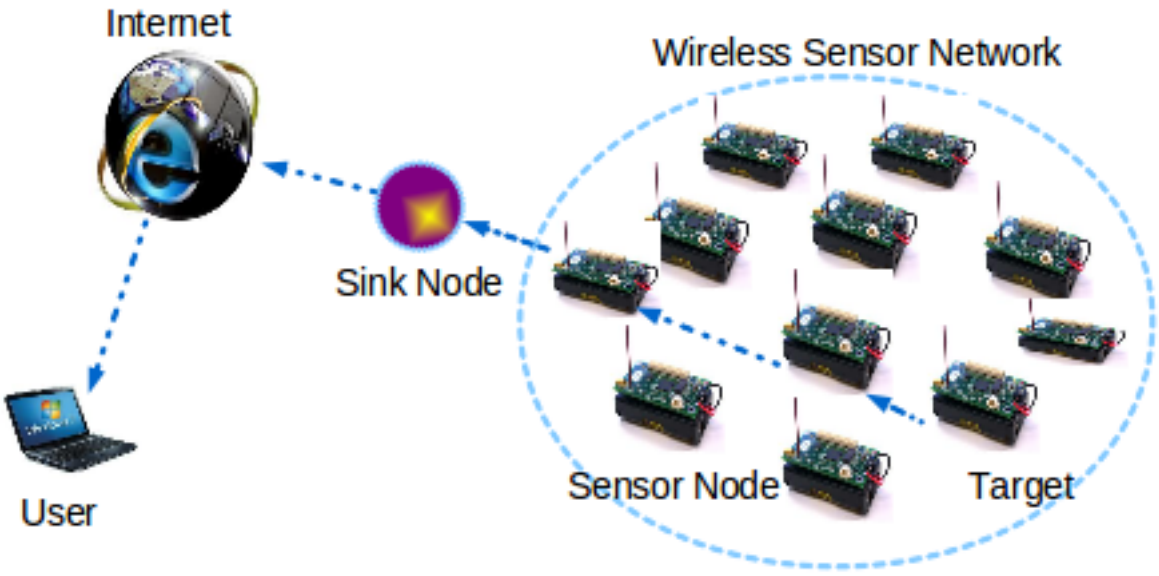

Figure 2.1: An illustration of a Wireless Sensor Network

Traditional sensors have two ways of being deployed; 1) They were positioned far away from the actual *phenomenon* (e.g. something known by sense perception) which required large sensors using complex techniques to distinguish the targets from surrounding noise. 2) Several sensors were deployed that only performed sensing and their communication topology had to be carefully engineered and they transmitted time series of the data to the central nodes which performed the communication.Wireless Sensor Networks on the other hand, is constructed by deploying a

large number of sensor nodes close to the phenomenon and their position doesn't need to be engineered or predetermined.[1]
The features that Wireless Sensor Network has supports has a wide range of usages. For example, the city of Chicago plans to installWireless Sensor Network for tracking information on urban conditions. The sensors are planned to be low-resolution cameras, infrared cameras and microphones for analyzing urban areas and sending the information back to a secureremote server. The article stresses that even though the sensors would have a low-resolution cameras, the video could still be used to identify individuals. Which could cause a privacy concern.To prevent this, the project will also define privacy policy before installing the sensors, through public hearings and also informing the public what kind of information and how the information would be gathered.

present in another way?

ref?

mention this in motivation

2.2 Privacy Issues in WSNs

When adversaries can access to sensitive information in Wireless Sensor Network , it's a privacy issue. This can either be achieved by accessing sensor data or eavesdropping on communications in the network. For example, if an adversary gains access to data from sensors monitoring a home on both the inside and the outside, they could derive information regarding the inhabitants' behaviours or private activities.
Furthermore, there are many issues related to privacy. In a paper by Haowen and Perrig[**?**], they state that the main problem related to privacy is that the information from sensor nodes are aggrevated and can be accessed remotely. This makes it easier for adversaries to access information in a low-risk and anonymous manner. Also making it possible for just one adversary to monitor multiple sites at once.
Finally, a major issues was mentioned in an paper by Al Ameen, Liu and Kwak[**?**], as privacy is a major concern in healthcare applications. They said that if the privacy issues aren't honestly debated, there is a risk for public backlash that will result in mistrust and might make available technology not used. This can happen either if the data is obtained with or without the consent of the person, since the damage can happen either way.

2.3 Data Minimization

As defined by the EDPS (European Data Protection Supervisor); "The principle of data minimization means that a data controller should limit the collection of personal information to what is directly relevant and necessary to accomplish a specified purpose."[2]
With the world becoming more and more digitalized, the collection of personal data from users becomes a growing concern. Today, data processing entities automatically makes decisions based on data analysis which can impact the lives of individuals, which in turn makes the need for protecting their personal data is even greater.[3]
Another problem which arises from being more connected is that individuals, whose data is being collected, are often unaware of the consequences of the data processing

that comes after. Also, legal repercussions for infringement of data protection obligations is usually only due to a breach or a misuse that has already occured. There are systems that support this thinking, that requires users to know what data is being collected, such as Privacy Enchaning Information Management Systems (PEIMS) which will be discussed later on in this chapter.

In a paper by Pfitzmann, Andreas and Hansen and Marit, a combined terminology for the aspects of Data Minimization was defined. The main definitions they used were: *Anonymity*, *Unlinkability*, *Undetectability* and *Unobservability.*[4] These definitions will help broaden the explanation of data minimization for the sake of this thesis and therefore we will go through them more thoroughly. To give these definition some context, we will use the same terminology as in the paper, where the two most important ones are *subjects* and *Items of Interest* (IOI). For privacy reasons, being that we wish to maximize it for human beings, subjects are mainly users in a system. But in the generalizations that follows, it can be a legal person or even be a computer. An Item of Interest is a generalization of what can be seen as information, e.g. the contents of a message, the name or the pseudonym of a user or even the action of a user sending a message. All of these can be an Item of Interest, in the following list are some definitions to expand the meaning of Data Minimization:

Anonymity means that for a subject to have anonymity, it has to be indistinguisable in a set of subjects. Meaning that if a subject has a set of attributes defining the subject, there always has to exist an appropriate set of subjects with potentially the same attributes, in other words: "Anonymity of a subject means that the subject is not identifiable within a set of subjects, the *anonymity set.*", where the anonymity set is the set of all possible subjects. The opposite of anonymity is called *Identifiability.*

Unlinkability is the relation between IOIs in the system. If several IOIs become compromised, an attacker should not be able to distinguish whether these IOIs are related or not. The opposite of unlinkability is called *Linkability.*

Undetectability is an attribute for an IOI. It means that an attacker should not be able to distinguish whether the item exists or not. The opposite of undetectability is called *Detectability.*

Unobservability is an attribute which is combination of *anonymity* and *undetectability.* It means that, in regards to an IOI, that neither if a subject is involved in the IOI or not, he or she should be aware of the other involved subjects. The opposite of unobservability is called *Observability.*

2.4 Modelling Concurrency

In this section we will present some different approaches to modelling concurrency. These will give an insight into the other tools available, the chosen approach for this thesis (Model Checking) will not be described here but will be thoroughly explained in the next chapter.

2.4.1 Petri Nets

In 1962, Carl Adam Petri disserted his work on a more graphical way of modelling concurrency, namely using something called *Petri Nets*. A Petri Net is a directed bipartite graph. An example can be seen in Figure 2.2. Each node in the graph represent transitions, shown as bars, or places, which are shown as circles. The black dots at places are called tokens, they indicate the holding of a condition at that place. Connecting the nodes are directed arcs that describe pre- and/or postconditions for the transitions, illustrated as arrows.

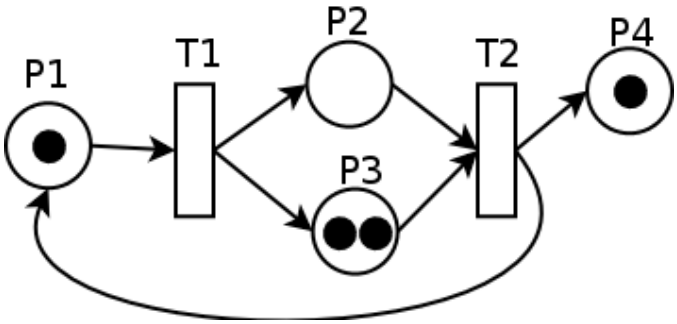

Figure 2.2: Illustration of a Petri Net

The primary rule for Petri Net theory is the rule for transition enabling and firing. It derives from the idea that many systems can be described as system states and their changes. So to simulate the behavior of a system, each state or marking are allowed to change according to the following rule:

- A transition is enabled iff each of its input places has atleast one token
- A transition can only fire if it is enabled.
- When a transition fires it removes a token from each of its input places and a token is deposited into each of its output places.

By analyzing different properties of a Petri net model of a system, such as liveness and boundedness, one can prove other properties aswell. For example, a Petri net is said to be *live*, if there always exist a fire sequence to each transition in the model, and if the model is live it's also guaranteed to be free of deadlock[5]. A Petri net is said to be *k-bounded* if for each place, there exists an upper bound k, for how many tokens that can be there simultaneously. If k is 1, then the system is said to be *safe*.

2.4.2 Process Calculi

Process Calculi is a family of related approaches for formal modelling of concurrent systems. It allows for a high-level description of the interaction, communication and the synchronization between processes and agents. Some examples of different Process Calculi are CSP, LOTOS and π-calculus. Their focus vary, for they are specialized on modelling different systems, but some features they (and other Process Calculi) share are:

referencez

- Interaction between processes are represented as communication(message-passing), rather than manipulation of shared variables.

- Processes are described as a collection of primitives and operators for those primitives.
- Algebraic laws are defined for the process operators, which allows them to be analyzed by *equational reasoning*.

find ref

Initially, to define a *process calculus*, you start with a set of channels as a means of communication. The internal structure of channels are rich and are constructed to improve efficiency, but when explained theoretically these improvements are usually abstracted away. Also, a way to form new processes from old ones is required, this also varies from the different implementations but what they have in common can be summarized the following:

- A way of expressing parallell composition of processes
- A way of specifying which channels are used for sending and receiving data
- A way to sequentialize interactions
- A way to hide interaction points
- Recursion or a way to process replication

Furthermore, an example of constructing a process calculus model can be seen in the CRC Handbook of Computer Science and Engineering[6], written in π-calculus and compared to λ-calculus.

2.5 Related Work

2.5.1 Data Erasure and Declassification

Another step to consider when ensuring privacy for users having their data collected is the aspect of releasing or removing the data. In many systems, both of these functionalities are required. For this sake, different policies for *erasure* and *declassification* need to be clearly specified so the users' privacy is protected.
In an paper by Chong and Myers[7], they propose a security policy framework where policies for both can be specified so it suits the desired application. In said framework, one would specify an erasure policy on under which conditions information must be erased. One could also state what policy that would allow data to survive erasure, since information could be allowed to still exist within a system in a restricted form. Secondly, declassification policies would define what policy should be enforced on new information, the conditions under which said information would be declassified and finally the policy it should have after declassification.
This approach covers an important aspect of privacy, namely the managing of personal data. This differs from the focus of the thesis, as we seek to minimize the collection of data to achieve better privacy.

2.5.2 Privacy Enhancing Identity Management Systems (PEIMS)

In an online setting, it's assumed that people would like to retain their anononymity.[8] To let users manually control their identities would be a cumbersome process, so instead an automated solution managing this would be preferred. Such a solution can be an Identity Management System (IMS).

An IMS is a system that allows support for "administration of information subjects". An extension of this is Privacy-Enhancing Identity Management Systems (PE-IMS) which supports "active management of personal information" which grants all parties involved flexibility and control over their personal data. A principle used for this is called 'Notice and Choice', a central aspect of data minimization, which means user-controlled linkability of personal data. This puts the responsibility on the user to make informed choices of representing and managing their partial identities.
This allows a user to be as anonymous as they wish, within the predefined limits, since a PE-IMS can be designed to offer any degree of anonymity and linkability. Applications utilizing PE-IMS would specify the range of choices available to the user. Some applications might require some authenticity from the user, e.g. government processes, and in other cases a user could be allowed complete anonymity. By allowing each application different levels of authorization, one can minimize linkability between different communication events and still maximize information exchange while preventing context-spanning profiling.

3
Theory

This chapter provides an introduction to different tools that were used in the thesis to reach the stated aim. First we explain *Formal Verification*, following it up with *Model Checking* and its' strengths and weaknesses. Finally some introduction to the model checking tool *SPIN*, Simple Promela INterpreter, and its' input language *Promela*.

3.1 Formal Verification

The act of formal verification means to make use of mathematical techniques to make sure that a design upholds a defined functional correctness [9]. This means, that if we assume we have the following: a model of a design, a description of the environment where the design is supposed to operate in and some properties we wish the design to uphold. With this information, one may want to construct some input sequences, that are in the allowed in domain of the environment, that would violate the properties stated. A common practice for finding such patterns today are random simulations or directed tests. Formal verification allows for an extended approach to this, as it allows both to search for input sequences that violates the properties but also allows to mathematically prove that the stated properties holds when no input sequences exist.

3.2 Model Checking

A traditional approach to verifying concurrent systems is based on using extensive testing and simulation to find and eliminate unwanted occurrences from the system, but this way can easily miss crucial errors when the system that's being tested has a large number of possible states[10]. An alternative technique that was developed in the 1980's by Clarke et al. is called *temporal logic model checking* or "Model Checking".
Model Checking is an automated technique to verify finite state concurrent systems, by letting a tool verify that a model holds for certain properties. The process of applying Model Checking to a design is separated into several tasks; *modeling*, *specification* and *verification*.

Modeling: First task is to translate a design into a format which is accepted by a model checking tool. This is either a compilation task or a task in abstracting certain aspects of the design to eliminate irrelevant or unimportant

details, due to limitations on time an memory.

Specification: Second task is to state which properties the design is supposed to have. This is usually done using in a logical formalism, commonly in temporal logic, which can express assertions on a system evolving over time.

Verification: The final step is allowing the tool to verify the specification on the model. This will either be a positive result, meaning the model satisfies the properties, or a negative result where the properties aren't. A negative result can also be that the model's state space is too large to fit into a computer, which will require the model to be further abstracted to be verified.

3.2.1 Model Checking Workflow

The use of model checking in practice typically follows the workflow in Figure 3.1. A design is translated into a description, that the model checker can read, and a specification of wanted or unwanted behavior is translated into a property. Then the model checker will produce a result which is either that the property is upheld or an error explaining how the property is invalidated.

Figure 3.1: Workflow of Model Checking

3.3 SPIN

The model checking tool used in this thesis is called SPIN, an abbreviation of Simple Promela Interpreter. The SPIN tool allows to create an abstract model of a system, specifying properties that the model must hold and then verify them to see if there is possible system state that invalidates it. SPIN verification models are focused on proving the correctness of process interactions.[11] Process interactions can be specified in several ways using SPIN; rendevouz primitives, asynchronous message passing, shared variables or a combination of these.

3.3.1 Promela

Promela is a specification language with its' focus on modeling process synchronization and coordination rather than computation. Therefore the language targets the description of conurrent software systems, rather than the description of hardware circuits, which is more common for other model checking applications[11]. The features in the Promela language allows for description of concurrent processes and communication through message passing over buffered or rendevouz(unbuffered) channels.

3.3.1.1 Promela Example

To give an impression of Promela's syntax, Listing 3.1 serves as an small example that captures most of the concepts used in this thesis. The example models an procedure called *environment*, receiving a message `meter` on the channel `envChan`. Then the process undeterministically choses one of the two responses in the guard statement and responds back on the same channel. Worth noting is that most part of the model is captured in an `atomic`-statement, this means that when the request is received, this process will be allowed to execute the rest of the `atomic`-statement without any interleaving. Since this process flow isn't realistic in a concurrent system, where interleaving is prone to occur, all usage of `atomic` has to be explained and carefully motivated.

Listing 3.1: Promela Example

```
1  active proctype Environment() {
2
3  Idle:
4    if
5    :: atomic {
6      envChan ? meter ->
7        if
8        :: envChan ! bigData;
9        :: envChan ! smallData;
10       fi;
11       goto Idle;
12   }
13   fi;
14 }
```

3.3.2 Properties in SPIN

Specification

In order to prove or disprove a property using SPIN, we must first state them in some formal notation. This can be done by either using `assertion`-statements, to ensure a property at a certain point in time, or using LTL to prove properties over an entire system trace. Except the operators inherited from propositional logic (*negation*, *conjunction*, *equivalence*, *implication*, etc.) LTL also provides the temporal operators such as *always*, *eventually* and *until*.

include reference

Always ($\Box$) states that a property has to hold on the entire subsequent path, e.g $\Box a$ means that the condition a always holds true. In promela this is either written as `always` or `[]`.

mention what subsequent path means

Eventually ($\Diamond$) states that a property has to hold somewhere on the subsequent path, meaning that $\Diamond a$ means that a must hold in the current state or in some future state. This is written in promela as `<>` or `eventually`.

Until (U) captures a relative behavior between two condditions, e.g. a U b means that a must hold true atleast until b holds true. In promela this is written as `U` or `until`.

For a complete description of Linear Time Logic and its' semantics in SPIN, see Holzmann (2003, p. 135-139).

Verification

Spin allows us to either prove properties that always should hold true (safety properties) or error behaviors (i.e. properties that should never hold). When verifying safety properties in Spin, instead of trying to prove that a property holds true in each possible system state, it tries to find a state in which the property is invalidated. This is intuitively a faster way of finding erroneous behavior since when the verification finds one counterexample to the stated property, it no longer needs to search other states. So when running the verification, Spin negates the specified property and then attempts to find a system trace in which the negated property holds. If this is successful, then the property can be violated. Otherwise, if no such trace exists, the property is verified to always hold true.

3.3.3 Problem space reduction

spinreferencemanualbook

There are two strategies that SPIN uses to reduce the number of states generated in verification.The aims of them are; "to reduce the amount of reachable system states that must be searched to verify properties, or to reduce the amount of memory that is needed to store each state".

One strategy to achieve this is called partial order reduction, which relies on selecting and examining only a subset of all possible execution paths. An example of this is by detecting interleaving of processes which relative ordering do not affect the final outcome of the execution, with regards to the property being verified.

mention clarke et al?

Another strategy SPIN uses is stutter equivalence. Spin's partial order reduction strategy assumes this, and by so only guarantees verification for stutter invariant properties. This makes it impossible to verify properties containing the *next*-operator.

mention meaning

3.3.4 Over-Collection

In a paper by Yibin Li et al**??**, they adressed the issues of over-collection and presented a framework to solve them. Their focus was the "smart city", an urban development vision with social facilities being connected wirelessly with information and communication technology and IoT-technology in a secure fashion to improve the efficiency of services. The system would help identify users in a smart fashion to relieve the need of ID- or credit cards. This requires that smart cities have a system that contains a lot of different users' information, which puts the users at a potential privacy leakage when accessing these features. They also sought to prevent over-collection of data, but though a privacy protection. Their definition of data over-collection was; "Collecting data more than enough on it's original function while within the permission scope". This definition is related to that their target, for which they sought to minimize over-collection, was smartphones. An example they used for explaining this definition was the following: "I take a picture and want to share it with my friends via some SNS app in my smartphone. For sharing this

photo to my friends, I have to agree the permission request from this app. After authorize the access permission to this SNS app, all my photos are available to this app, but I only want this app to access one specific photo. As a result, this app may collect data more than enough on my original requirement while within the permission scope which I authorize."
This definition, though focusing on a different target system, still held some context to our work and was used a reference point for the initial work on over-collection.

3.3.5 Decisions

In our research of over-collection, the project realized we needed some algorithmic way for a system to determine if it was over-collecting or not. To this end, we studied *decision problems*. A decision problem is a question expressed in some formal system that can be stated a "yes-no" question. And an algorithm used for solving decision problems is called a *decision procedure*, which terminates with a "yes" or "no" answer[12]. A formal system for expressing decision problems could for example be propositional logic, an example of its' syntax can be seen in Figure 3.2.

$$x_1 \wedge (x_2 \vee \neg x_3)$$

Figure 3.2: An example of a propsitional logic

show example of a decision procedure
explain the point of it

4

Modeling & Specification

In order to address the goals stated in the aim, the project first had to visualize and construct the models which would be investigated. To this end, Model Checking was used. The first two steps in model checking is to translate the sought design into models that can be understood by the model checker and defining the properties specified on the system. This chapter covers the process of these two steps, by first stating some basic concepts and then explaining the iterative process that led to the final results.

4.1 Definitions

First we will describe how a typical Wireless Sensor Network , that we sought to model, would look. The project begun with an example as simple and general as possible, having an agile development process in mind, which would then be expanded to more complex models later on.

4.1.1 Basic WSN

As mentioned in the background, a basic Wireless Sensor Network consists of a set of collection nodes (referred to as "nodes"), a central server (referred to as "Server") and finally an Environment (the observed source). An illustration of this can be seen in Figure 4.1.

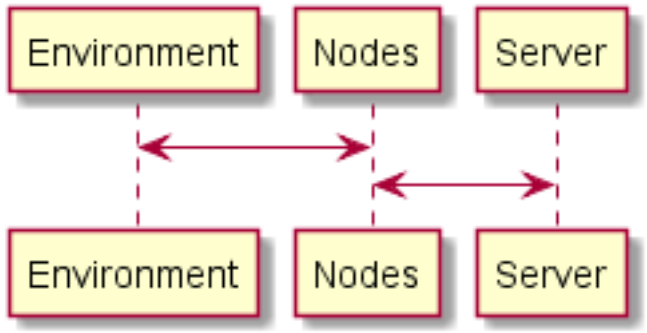

Figure 4.1: An illustration of a Wireless Sensor Network

These entities were chosen as the three pieces that our initial models would consist of. To abstract these further, the word *Actor* was used to represent that these entities

were processes acting on data. Furthermore, the project followed by defining the characteristics of each of these.

4.2 Actors

A Wireless Sensor Network is built up by several different entities that communicates data between each other. Generally a network consists of multiples of virtually the same entity, e.g. multiple collection nodes, where each of these are running the different instances of the same process.

To describe the interaction between two actors in the system, behavior models were used (e.g. Figure 4.4). Where the name of each actor is shown in the boxes at the top. The message channel used between them is shown as the arrows, where the arrow-head points to the actor receiving the message and the contents of the message is referenced above it. Finally the ordering of the messages are in a ascending order from the top, meaning the first message sent is shown furthest to the top of the figure.

At this stage the project realized that to simplify the modeling process, the environment could be considered as an actor in the system. This was an abstraction, since the observed source wouldn't act from a predefined pattern as an actor would, but to save time from having to manage concurrency with a shared resource.

4.2.1 Server Actor

The server is an actor receiving messages from nodes and storing it for later usage. A server's behavior will vary depending on the structure of the system. If the decision is taken centrally the server will be the one checking for over-collection, otherwise it will be a node. Also if the communication is managed through the server, if the nodes doesn't communicate with each other, the server will act as a repeater for the decision.

In Figure 4.2 is the behavior for a system where server makes the decision and nodes doesn't communicate with each other. First, the node sends some data, the server checks for over-collection and replies accordingly. The response will either be a "stop" signaling that over-collection has occurred and the node should stop collecting or it tells it that it can continue collecting.

In addition to the behavior model, an Finite State Automaton(FSA) were designed for the server (Figure 4.3). The initial state being `Idle_a`, which the server will stay in until some `data` is received. "Data" being either `bigData` or `smallData`; $\texttt{data} \in \{\texttt{smallData}, \texttt{bigData}\}$. The data is received in `Idle_a` and `Idle_s` and then checked in `Answ.` and `Hold`, meaning only the outgoing transitions from `Idle_a` and `Idle_s` is incoming data, the other are calculated internally. This also means the server will loop indefinetly between `Answ` and `Idle_a` as long as only smallData is received. When `bigData` is received the server will enter the `Idle_s-Hold` loop instead, which denotes the states where the server is requesting the nodes to stop collecting more data.

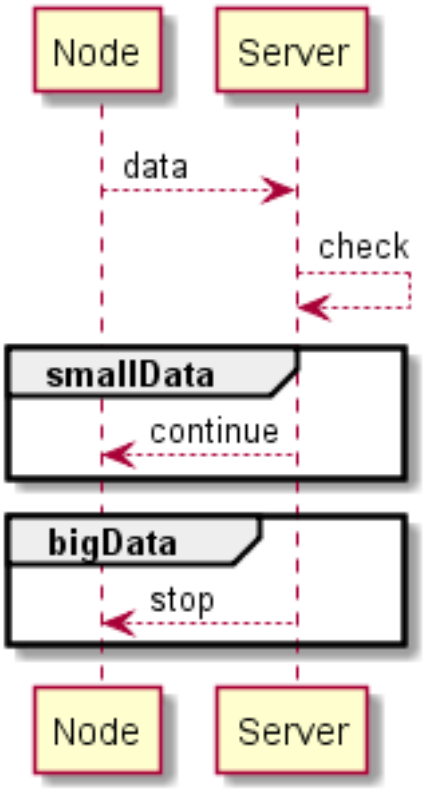

Figure 4.2: Behavior Model between Server and the Node

4.2.2 Environment Actor

The process for the environment actor had two steps:

1. Generate random data
2. Serve random data to a requesting node

As mentioned before, the first step is not intuitive for an environment since the observed source isn't randomly varying, but for modeling purposes this is a simplification made to reduce the complexity of the model. In Figure 4.4 the behavior between a node and the the environment is described.
The corresponding FSA for the environment is seen in Figure 4.5. The environment will stay in the initial state `W` (short for "Waiting"), until a node `meter` it. Then the data is "generated" in `G` (short for "generate data") and served back to the node.

4.2.3 Node Actor

As seen in the behavior model for the node actor (Figure 4.6), it captures the majority of a typical scenario for the entire system. That is intuitive since the node communicates with both of the other actors of the system and is a intermediatepart of the system. The scenario is when a node collects data, that doesn't cause a system-change, and forwards it to the server.

is this the right word?

The alternative behavior for system is described in Figure 4.7 instead. There the data collected causes the server to make the decision that the node should stop collecting.

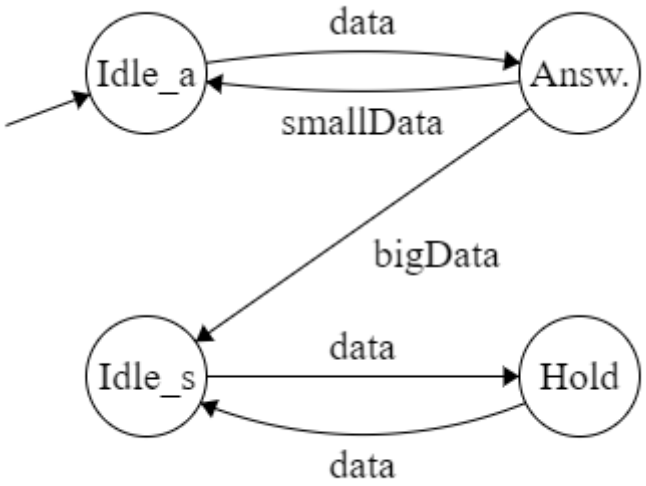

Figure 4.3: Finite State Automata for the Server Actor

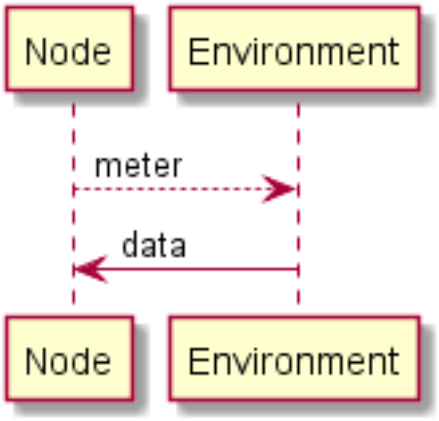

Figure 4.4: Behavior Model for the Environment

argue why I've kept the states to a minimum here?

This behavior can be described in a FSA, as seen in Figure 4.8.The node meters data from the environment and passes it forward to the system. There it waits (noted by the state `Wait`) for a response before returning to the `Idle` state.

4.3 Modeling

With this defined, the project began to construct Promela models representing these actors' behaviour. This would be the initial example which then would be expanded step by step into a complex model. This section will describe how the modeling was carried out.

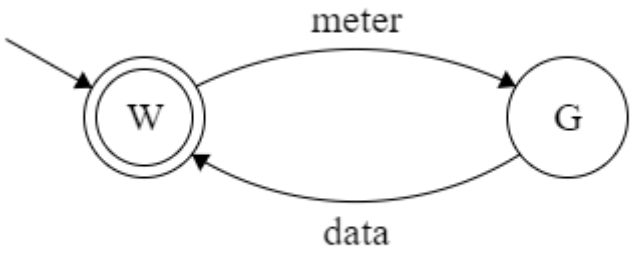

Figure 4.5: FSA for the Environment Actor

4.3.1 Initial Model

The first iteration consisted of a Wireless Sensor Network of three collection nodes, a server and an environment. The collection nodes would sent a request to the environment to receive data which would then be passed to the server for analysis. Each message being sent to the server could be the source of over-collection so to each message received, the server would respond to the nodes whether to continue collection or not. The promela representation of this can be seen in appendix *centralized_decision_multiple_nodes_8jul*.

As can be seen in the example, the network passes the data between the actors and when the server triggers the decision to stop, the message is passed through the network and the system shuts down. To verify that this functionality was achieved, a correctness property was specified as *When the decision to stop is taken, the system should shut down.*

This initial model were then analyzed, for how it could be expanded to represent a more complex example. This analysis is described following subchapters.

4.3.2 Variations

During the work on the initial model, we realized that some assumptions were made for the system and that our initial model would only work if the target system communicated in the same way. Which might not always be true. And since we hadn't chosen a predefined system to model, we aimed to keep our model as general as possible to better investigate over-collection. So to abstract our model we composed a set of different architectural variations of Wireless Sensor Networks to model, which we considered to help us achieve the sought aim.
The variations the project chosed to focus on were the following:

Centralized or Decentralized decision making
The first choice reflected how much the sensor nodes would analyze the data. Since nodes can have a processing unit, they could potentially analyze the collected data and make a decision on their own.

Conjunctive or Disjunctive decision analysis

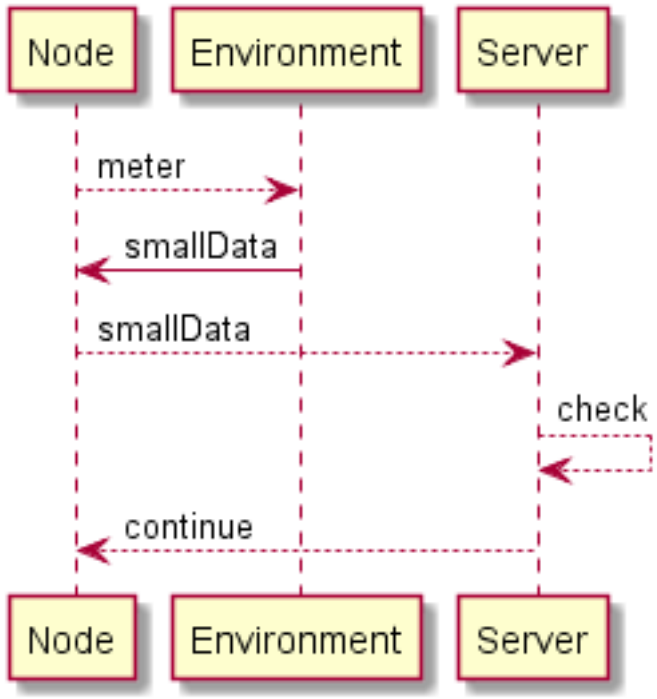

Figure 4.6: Behavior Model for a Node

The second choice reflected how the decision were processed, if the data from a single data point could trigger a decision or if the decision considered data from multiple entries in it's evaluation.

Centralized or Distributed communication

The final variation reflected how the network communicated, it was considered centralized if all communication were sent through a central unit, such as a server, or if nodes were allowed to communicate independently to each other.

This meant that our already constructed model were a model with **centralized decision making**, **disjunctive decision analysis** and **centralized communication**. From this the project continued to refine a model for each of the other variations. During the refinement of the these models, several revisions were made before the end result was reached. The following section will explain some of these intermediate revisions that led to the final result.

4.4 Revisions

The process of model checking is an iterative process, with many refinement steps before eventually reaching a final working model. As mentioned in the background, a refinement step for model checking can yield that either the models or the properties need to be changed. This section will show some examples which made refinements to the model, to give some context to the process.

The Atomic Requirement

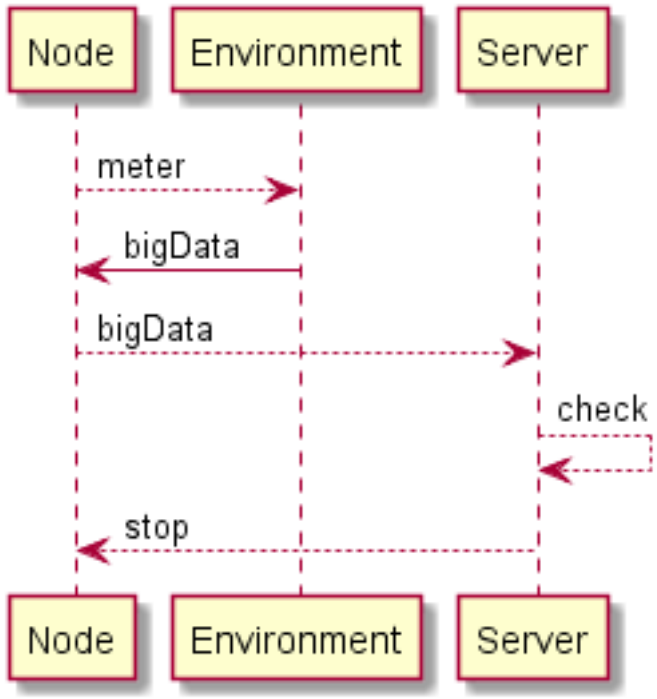

Figure 4.7: Behavior Model for a Node over-collecting

A big concern when modeling concurrency is the uncertainty of interleaving statements. To address such issues, one can use a modelling concept called **atomic** statements, which specify that several statements should be executed "at once" (without interleaving). These should be used with careful consideration, to not change the nature of what is sought to model. Our project experienced a 'state space explosions' (mentioned in a previous chapter). To reduce the state space, we introduced atomic statements at two steps in our models. One example is shown below:

Listing 4.1: Atomic statement

```
1 Waiting:
2         atomic {
3           if
4           :: out ? continue -> goto Idle;
5           :: out ? stop ->
6           fi;
7         }
```

This section also had problems with interleaving statements, it was the step where a node received the information to whether stop or continue collecting data. This change was due to that Promela uses communication channels to handle message passing and previously a shared buffered channel had been used. Though from verification with model checking, the amount of states rapidly increased with a buffered channel(*this can be referenced from the SPIN-book with the same example they had there*). So instead the project had to modify the models to use a unique channel between the server and each node. The atomic statement here handled the logic that a node that had received a message didn't wait, but instead immediately acted on the information.

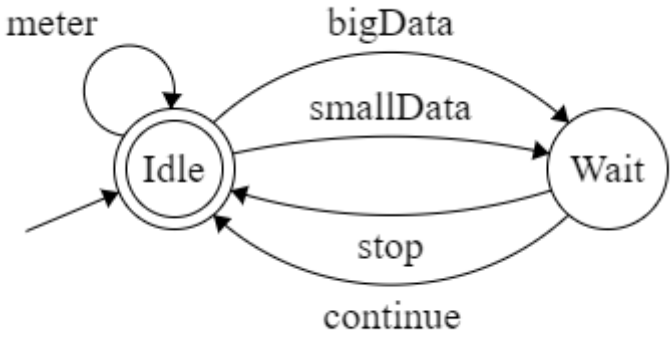

Figure 4.8: FSA for the Node Actor

Introduction of States

As can be seen in the initial model, this version handled the verification by using global variables to monitor certain values. An example is the variable `nodesDone` which represented on how many nodes were currently actively collecting data. This approach worked initially, but made the models lack certain information. Specifically for the liveness property to know whether a specific node had received data or not.

This caused the introduction of states in the models instead. Which also made the models easier compare to their sought FSAs.

The Network Channel

The Liveness Relaxation

The liveness property for the system demanded a lot of revisions. As defined, a liveness property should express that *eventually something good happens*. For this project this was a difficult thing to define,

4.5 Specification

This section presents the properties used to verify the system. The process of defining the properties were an iterative approach and several versions were considered, this section only covers the final properties were the result of the previously explained process.

4.5.1 Properties

The properties defined on the network were formulated using *Linear Time Logic* (LTL). This choice came from the fact that LTL were native to SPIN and the models

were abstracted to only focus on the relevant parts to the project, LTL could provide a simple and direct specification to that problem.

Correctness

The primary property sought of the system was that it was working as intended. This was formulated as a safety correctness property, to ensure that when decision had been taken, the system respond to the by changing its' behavior.

Definition 1. Safety Correctness
When the stop-decision is taken, the system should stop collecting.

LTL: $\square(O \rightarrow (\lozenge D))$

Where **O** and **D** corresponds to the state where the stop-decision is taken and the states where collection is stopped respectively. This captures the sought system change; whenever the system reaches the state O, eventually it will reach state D. An immediate change is not required, therefore the timing is relaxed by the eventually-operator.

Liveness

The second property was intuitive for the system since the initial models were constructed in such a way that when data is sent to the server, the first thing the server does it analyze it and respond accordingly depending on what data was sent.

mention it was verified by design?

Definition 2. Liveness (sending)
Eventually a node sends it's data to the server.

LTL: $\lozenge$ Node_Send

Where Node_Send denotes the state where the node sends the data to the server.

Definition 3. Liveness (replying)
If a node sends data to the server, eventually the server replies to the node.

LTL: $\square$(Node_Send $\rightarrow \lozenge$ Server_Reply)

server_reply doesnt exist atm.

Where Node_Send means the same as previously and Server_Reply denotes the state where the server responds to the node.

5
Design

This section covers the design aspect of the work, showing the final versions of the models discussed in the previous chapter as well as explaining how they work.

5.1 Modeling in Promela

The system consists of message channels between three different classes of procedures **Node**, **Server** and **Environment**. The node allows a dynamic number of instances to run at start-up and it's set by a predefined macro named `NUM_NODES`. As an abstraction, the project considered the data sent in the network as a set of two possibilities. Either the sent data causes a system change, the system realizes it should stop collecting to prevent over-collection, or it doesn't and it continues as before. This were noted as `bigData` and `smallData`, where `bigData` causes the system change and `smallData` doesn't.

The system procedures communicate using shared communication channels, `envChan` for the communication between the `Nodes` and `Environment` and `servChan` for the communication between the `Server` and the `Nodes`.

5.1.1 Environment

The environment is an abstraction made to simplify the work. It's considered to be a shared resource between the nodes where each node can individually meter the environment and then communicate it to the server. To achieve this the environment is constructed as an atomic statement so when a node puts up a request on the channel it's instantly handled before any other statement is executed. To handle the randomness between the outcomes (so both types of the data can be metered) an `if`-statement without guards is used.

argue translation?

Listing 5.1: Environment code

```
1  active proctype Env() {
2
3    Idle:
4      if
5      :: atomic {
6        envChan ? meter ->
7          if
8          :: envChan ! bigData;
9          :: envChan ! smallData;
```

```
10          fi;
11          goto Idle;
12      }
13      fi;
14  }
```

5.1.2 Server

The server consists of two primary states, the first being the initial state, noted below as `Answering`, where data is assumed to be collected and the second state, noted as `Stopping` where the system starts requesting that the nodes stop collecting to prevent over-collection. The states beginning with "`Idle_`" are just looping to check if a node is sending data.

Listing 5.2: Server code

```
1  active proctype Server() {
2
3  chan active_chan;
4  int i=0;
5
6  Idle_Answering:
7          if
8          :: nempty(servChan[i]) ->
9              active_chan = servChan[i];
10             goto Answering;
11         :: empty(servChan[i]) ->
12             i=(i+1)%NUM_NODES;
13             goto Idle_Answering;
14         fi;
15
16 Idle_Stopping:
17         if
18         :: nempty(servChan[i]) ->
19             active_chan = servChan[i];
20             goto Stopping;
21         :: empty(servChan[i]) ->
22             i=(i+1)%NUM_NODES;
23             goto Idle_Stopping;
24         fi;
25
26 Answering:
27         if
28         :: active_chan ? smallData ->
29             active_chan ! continue;
30             goto Idle_Answering;
31         :: active_chan ? bigData ->
32             active_chan ! stop;
33             goto Idle_Stopping;
34         fi;
35
36 Stopping:
37         if
38         :: active_chan ? smallData ->
39             active_chan ! stop;
```

```
40              goto Idle_Stopping;
41          :: active_chan ? bigData ->
42              active_chan ! stop;
43              goto Idle_Stopping;
44          fi;
45  }
```

5.1.3 Node

The node is initialized with a channel to communicate to the server with. It starts by attempting to meter the environment, then communicates it to the server and proceeds into `Waiting` to wait for an answer.

Listing 5.3: Node code

```
1   proctype Node(chan out) {
2   Idle:
3           envChan ! meter;
4           if
5           :: envChan ? bigData ->
6              out ! bigData;
7              goto Waiting;
8           :: envChan ? smallData ->
9              out ! smallData;
10             goto Waiting;
11          fi;
12  Waiting:
13          atomic {
14            if
15            :: out ? continue -> goto Idle;
16            :: out ? stop ->
17            fi;
18          }
19  DoneColl: // node will shutdown here.
20  }
```

6
Discussion

This section presents a discussion to the choices and limitations made to the thesis. Some ideas for further work is also presented.

- discuss why I used formal verification & model checking instead of traditional approaches
- discuss why I didn't build all models from the start
- discuss why I made simplifications to the initial models
- discuss why I chosed to use SPIN/Promela as a tool
- discuss the other concurrency tools in the background.
- mention that the project should had sought quantifiable means to measure collection or something else to formulate a decision on.

6.1 Model Checking

compare to a different approach and mention if any better results could'd perhaps had been achieved by doing so.
discuss the iterative constant rework of the properties and how time consuming such a task is. Initial planning estimated 2-3 weeks but in practice this took 1-2 months.

6.2 Over-Collection

overcollection was a difficult task to specify, since it's occurence varied a lot in different systems.

6.3 Data Minimization

7
Conclusion

Conclude the results of the report, did it go as expected? What progress did you make and what didn't you achieve that you had hoped? Did you reach the aim stated and did you keep yourself in the scope & limitations?
In this thesis, we aimed to investigate over-collection in wireless sensor networks. As a tool we used promela to model this and then apply data minimization as a concept to said models to reduce, or in the best case prevent, data over-collection in the systems. For modeling wireless sensor networks, a set of promela models were defined and correctness and liveness properties for the models were verified. Additional properties were discussed, such as fairness, but due to time constraints these weren't implemented.
The thesis had a difficult and abstract goal, with not as much related work to it as was hoped. If a working solution would had been achieved, we had hoped to present an approach for constructing or adapting existing Wireless Sensor Networks to achieve data minimization.
During the course of the thesis, an approach to a solution was to construct a module, not integrated into an entity in the system. Which simply could be installed and attached to the system, to prevent over-collection. This idea became an invisible goal to what would sought to be achieved.

Bibliography

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer networks*, vol. 38, no. 4, pp. 393–422, 2002.

[2] E. D. P. Supervisor, "Regulation (ec) no 45/2001," December 2000.

[3] G. Danezis, J. Domingo-Ferrer, M. Hansen, J.-H. Hoepman, D. L. Metayer, R. Tirtea, and S. Schiffner, "Privacy and data protection by design-from policy to engineering," *arXiv preprint arXiv:1501.03726*, 2015.

[4] A. Pfitzmann and M. Hansen, "A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management," 2010.

[5] C. Ramamoorthy and G. S. Ho, "Performance evaluation of asynchronous concurrent systems using petri nets," *IEEE Transactions on software Engineering*, no. 5, pp. 440–449, 1980.

[6] B. C. Pierce, "Foundational calculi for programming languages.," *The Computer Science and Engineering Handbook*, vol. 1997, pp. 2190–2207, 1997.

[7] S. Chong and A. C. Myers, "Language-based information erasure," in *Computer Security Foundations, 2005. CSFW-18 2005. 18th IEEE Workshop*, pp. 241–254, IEEE, 2005.

[8] M. Hansen, P. Berlich, J. Camenisch, S. Clauß, A. Pfitzmann, and M. Waidner, "Privacy-enhancing identity management," *Information security technical report*, vol. 9, no. 1, pp. 35–44, 2004.

[9] P. Bjesse, "What is formal verification?," *ACM SIGDA Newsletter*, vol. 35, no. 24, p. 1, 2005.

[10] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, 1999.

[11] G. J. Holzmann, "The model checker spin," *IEEE Transactions on software engineering*, vol. 23, no. 5, p. 279, 1997.

[12] D. Kroening and O. Strichman, *Decision Procedures*. Springer, 2008.