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

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

© JAKOB BOMAN, 2016.

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, January 26, 2017

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Motivation

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:

- Define Over-Collection and it's meaning with regards to this project.
- Construct a model of an example of a Wireless Sensor Network that stops the collection when Over-Collection is occuring.
- Implement (or generate) a C-implementation from the models and analyze them.

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 recent 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.

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]

2.2 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 (PE-IMS) 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:

implement this

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.*

2.3 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.

2.3.1 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 2.1. 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.

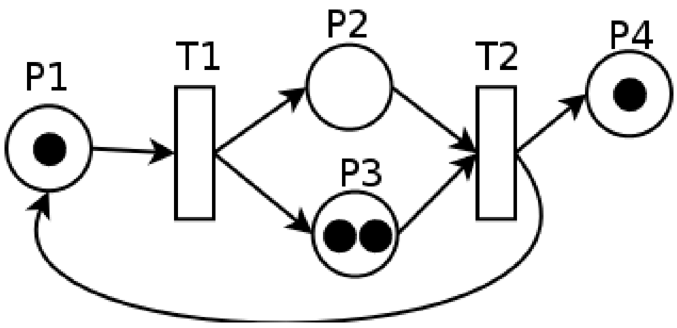

Figure 2.1: Illustration of a Petri Net

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*.

2.3.2 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.

2.4 Related Work

2.4.1 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.

2.4.2 Privacy Enhancing Identity Management Systems (PE-IMS)

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 the various concepts used throughout the thesis. 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 *insert ref here*. 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.

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 environment, receiving a `meter`-request on `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

```
active proctype Environment() {

Idle:
  if
  :: atomic {
    envChan ? meter ->
      if
      :: envChan ! bigData;
      :: envChan ! smallData;
      fi;
      goto Idle;
  }
  fi;
}
```

3.3.2 Properties in SPIN

3.3.2.1 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).

3.3.2.2 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

4
Modeling & Specification

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 explain how a typical design, for a Wireless Sensor Network we seek to model, looks. Secondly we will describe in detail how each entity, referred to as *Actors*, operates. Finally, before explaining the modeling, we will explain the meaning of some key aspects of the work such as *Decisions* and *Over-Collection*.

4.1.1 Basic WSN

A basic Wireless Sensor Network was defined as a starting point for the models. It consisted 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. Which is the basis for all the models described in the thesis. An important note on this setup is that the "environment" here is considered an entity (same as a node or a server). This abstraction was made to simplify the modeling, so the nodes instead of managing a shared resource instead can have the data communicated to them as messages.

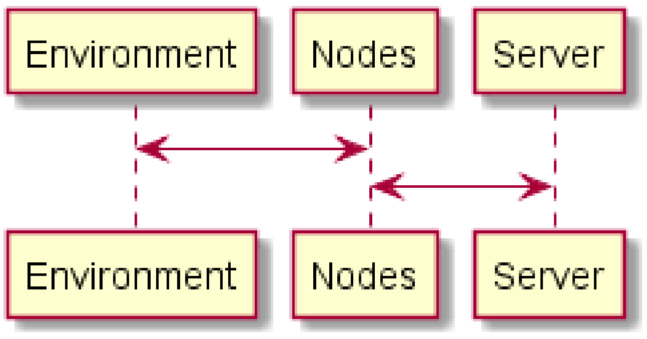

Figure 4.1: An illustration of a Wireless Sensor Network

4.1.2 Description of the 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. Throughout the thesis these entities will be referred to as *Actors*.
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.

4.1.3 Over-Collection

Over-Collection is the state or process which a system collects data beyond the scope of it's required parameters.

Definition 1. (Collecting)

Definition 2. (To Function)
if a process P yields a valid result by a specific amount of input parameters, we say it requires these input parameters

Definition 3. (Over-Collection)
if a process P collects more data than it "requires to function", we say a process is over-collecting.

4.1.4 Decisions

A *decision procedure* is an algorithm that terminates with a yes or no answer, given a decision problem.[12]

4.2 Modeling

This section covers the modeling aspect of the thesis which was one of the main goals of the thesis, first will be discuss some initial assumptions and show some faulty models that were refined to the final results.

4.2.1 Initial Models

4.2.2 Variations

- Centralized or Decentralized decision making
- Conjunctive or Disjunctive decision analysis
- Centralized or Distributed communication

4.2.2.1 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.

4.2.2.2 Decision Analysis

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.

4.2.2.3 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.

4.2.3 Initialization

An initial model was made for one variation, a model with a centralized decision making, disjunctive decision analysis and centralized communication. This was made as a starting point for other variations and also help define the properties sought of the network.
Furthermore, the project defined the individual components of the Wireless Sensor Network to three actors: *Node*, *Server* and *Environment*.

4.2.4 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

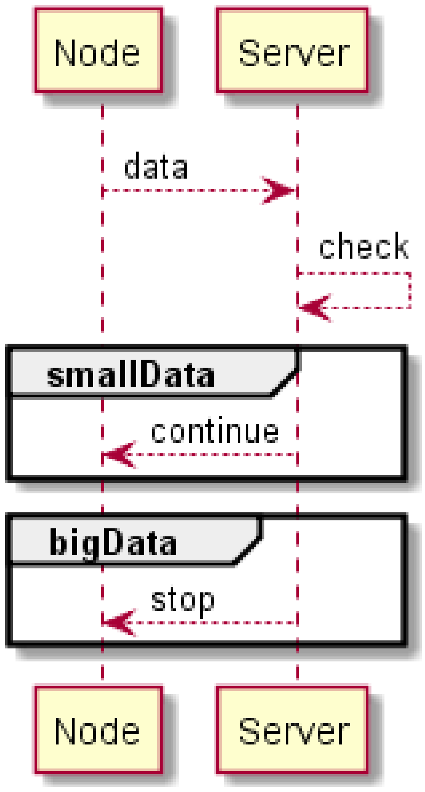

Figure 4.2: Behavior Model between Server and the Node

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.

4.2.5 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.

4.2.6 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

is this the right word?

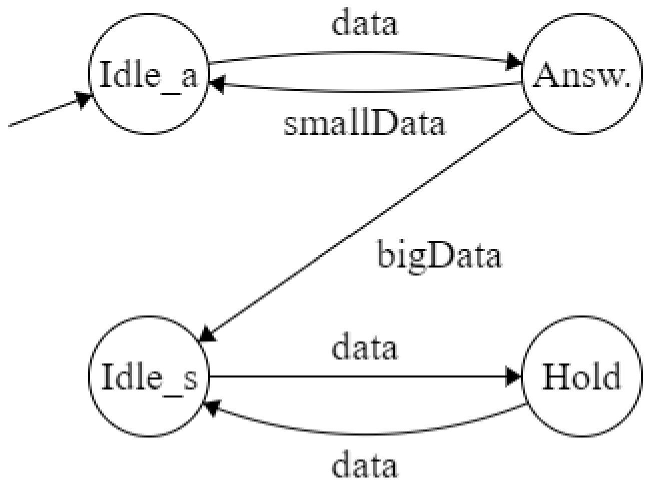

Figure 4.3: Finite State Automata for the Server Actor

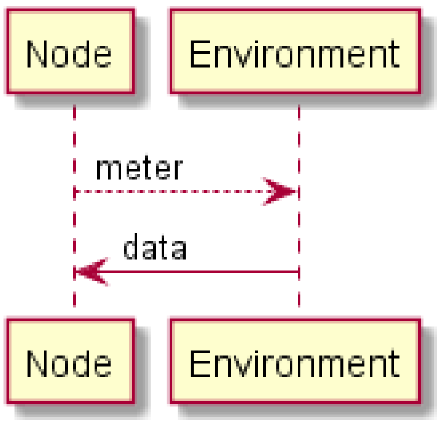

Figure 4.4: Behavior Model for the Environment

of the system. The scenario is when a node collects data, that doesn't cause a system-change, and forwards it to the server.
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.
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.

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

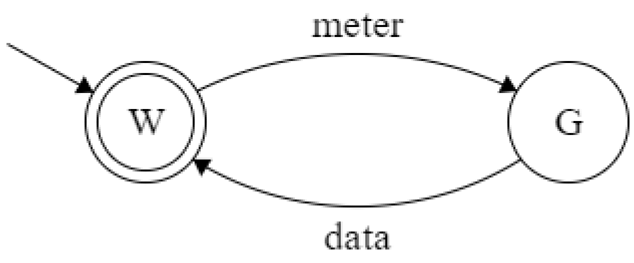

Figure 4.5: FSA for the Environment Actor

4.3 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 this process.

4.3.1 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.

4.3.1.1 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 4. Safety Correctness
When the stop-decision is taken, the system should stop collecting.

LTL: $\Box(O \rightarrow (\Diamond D))$

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 timingis relaxed by the eventually-operator.

is this the proper word?

4.3.1.2 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?

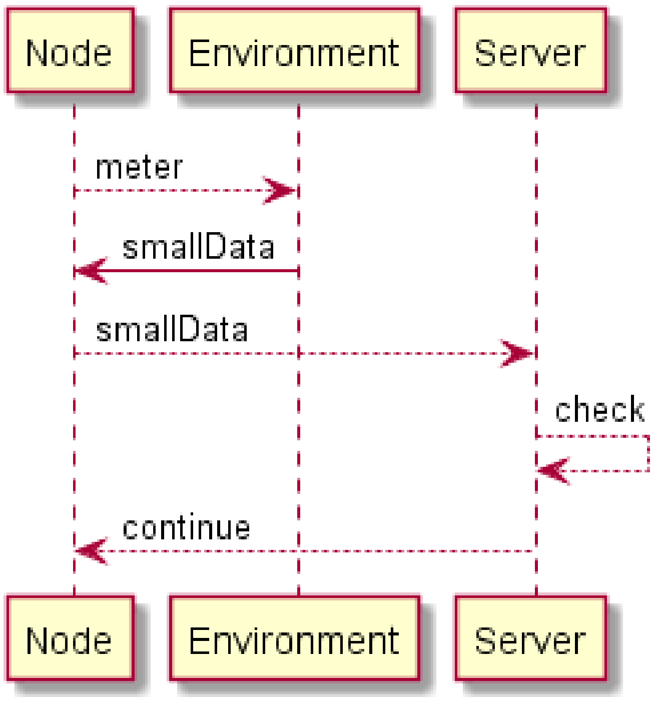

Figure 4.6: Behavior Model for a Node

Definition 5. Liveness (sending)
Eventually a node sends it's data to the server.

LTL: $\Diamond$ Node_Send

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

Definition 6. Liveness (replying)
If a node sends data to the server, eventually the server replies to the node.

LTL: $\Box$(Node_Send $\rightarrow$ $\Diamond$ 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.

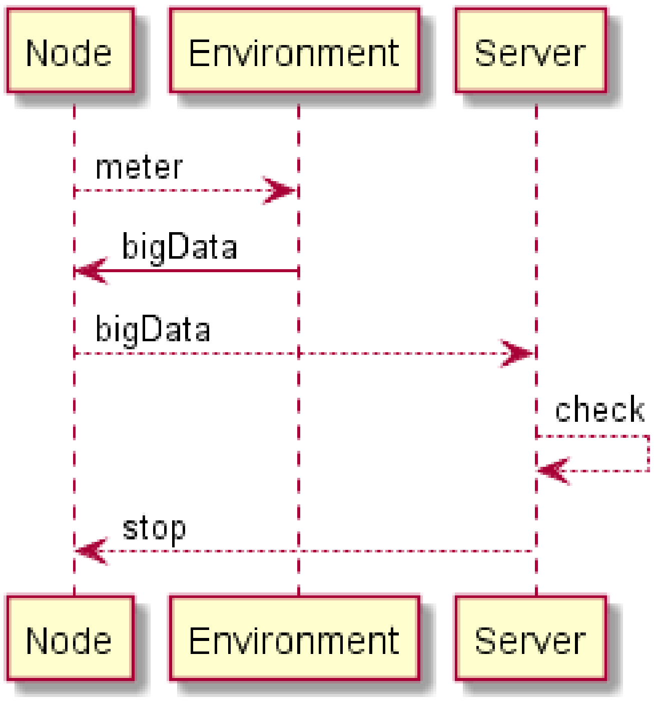

Figure 4.7: Behavior Model for a Node over-collecting

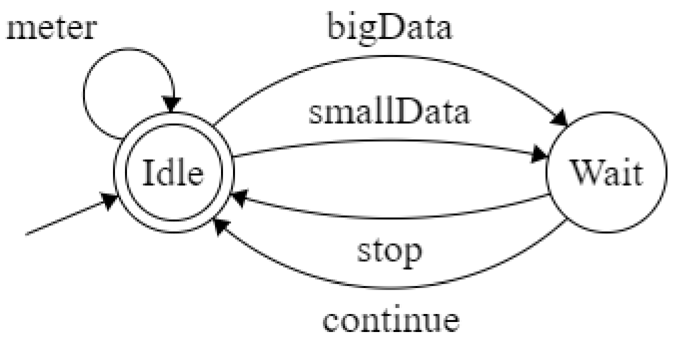

Figure 4.8: FSA for the Node Actor

5

Design

introduction-text: I seek to use SPIN/Promela for my models and first I need to justify why I did so and compare it to other tools...

Analysis of UPPAAL vs. TLC for verifying the WS-AT Protocol.
Survey regarding the NuSVM "symbolic" model checker.

Tool	**SPIN**	**UPPAAL**	**NuSVM**
specification language	promela	timed automata network with shared variables	...
necessary user's background	programming	programming	...
expressiveness of spec. language	...	restricted, communicating state machines, C-like (but finite) data structures, inductive approach	...
model checker characteristics	...	verifies the full specification language (with time)	...
modeling / verification speed	...	slower modeling, faster verification	...
verification of time/cost features	...	straightforward modeling and state-of-the-art verification support	...
parameterized reasoning	...	...	...

Table 5.1: Comparison between the model checkers SPIN, UPPAAL and NuSVM.

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

```
active proctype Env() {

  Idle:
    if
    :: atomic {
      envChan ? meter ->
        if
        :: envChan ! bigData;
        :: envChan ! smallData;
        fi;
        goto Idle;
    }
    fi;
}
```

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

```
active proctype Server() {

chan active_chan;
int i=0;

Idle_Answering:
        if
        :: nempty(servChan[i]) ->
            active_chan = servChan[i];
            goto Answering;
        :: empty(servChan[i]) ->
            i=(i+1)%NUM_NODES;
            goto Idle_Answering;
        fi;

Idle_Stopping:
        if
        :: nempty(servChan[i]) ->
            active_chan = servChan[i];
            goto Stopping;
        :: empty(servChan[i]) ->
            i=(i+1)%NUM_NODES;
            goto Idle_Stopping;
        fi;

Answering:
        if
        :: active_chan ? smallData ->
            active_chan ! continue;
            goto Idle_Answering;
        :: active_chan ? bigData ->
            active_chan ! stop;
            goto Idle_Stopping;
        fi;

Stopping:
        if
        :: active_chan ? smallData ->
            active_chan ! stop;
            goto Idle_Stopping;
        :: active_chan ? bigData ->
            active_chan ! stop;
            goto Idle_Stopping;
        fi;
}
```

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

```
proctype Node(chan out) {
Idle:
          envChan ! meter;
          if
          :: envChan ? bigData ->
             out ! bigData;
             goto Waiting;
          :: envChan ? smallData ->
             out ! smallData;
             goto Waiting;
          fi;
Waiting:
          atomic {
            if
            :: out ? continue -> goto Idle;
            :: out ? stop ->
            fi;
          }
DoneColl: // node will shutdown here.
}
```

5.1.4 Translation

this section will generally discuss the translation and the new errors that could occur, but are handled by design.

5.2 Verification

Discuss the results from the verification, present modifications done to fix any errors that might occur (perhaps show a interesting case of this).

6

Implementation

Discuss different approaches to verify the implementation and argue for the one I decided on.

6.1 Code Generation

Decide on a tool, discuss it

6.2 Satisfaction

Explain how I used my approach to verify the implementation

6.3 Analysis

Analyze the result of the generation and discuss limitations on the current models. E.g. redundancy from the generation or weaknesses in terms of security

7 Discussion

Discuss different choices made and why they were made.

- 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.

8
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?

9
Ethics

this section will discuss ethical aspects and what ethical impacts it can have.

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.