TAMPEREEN TEKNILLINEN YLIOPISTO

MIKKO ERONEN

IMPLEMENTATION OF DATA SYNCHRONIZATION OVER A CHALLENGED NETWORK

Master of science thesis

Examiner: Professor Jarmo Harju

Examiner and topic approved by the Council of the Faculty of Computing and Electronic Engineering on 24 August 2015

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
ERONEN, MIKKO: Implementation of data synchronization over a challenged network.
Master of Science Thesis, 48 pages, 7 Appendix pages
August 2015
Major: Communication Networks and Protocols
Examiner: Professor Jarmo Harju
Keywords: Internet of things, Challenged network, Mining equipment.

This thesis is related to the trend of the industrial internet of things. There exists a fair number of product and service examples where a manufacturer has a need for usage data harvesting. The gathered usage data can be used, e.g., in product development. In this thesis the product is mining equipment and its maintenance.

Sending the data straight from the mining equipment to the manufacturer is problematic, since mines often lack Internet connection. In some cases mines have local area networks, but in other cases those are not available. The only method of gathering the data can be transportating via USB flash drives or similar. The way the data is moved with the flash drive from the mining equipment to a location with Internet connection is called aided mine network. This location can be, e.g., an office building near the mining area. The core problem of the thesis is the gathering, moving, and synchronization of the usage data using the aided mine network.

In this thesis, a plan to implement the gathering of the data is developed. The solution is called DATAMiNe, i.e., Data Aggregation Through Aided Mine Network. The network consists of three parts. The parts are a Manager, an Edge Relay, and a Data Aggregator.

DATAMiNe's architecture is designed so that it supports an easy replacement of the aided mine network. Replacement can be a local area network, or an integrated Internet connection in the mining equipment. A communication protocol between the Manager and the Edge Relay is designed so that it supports the special needs of the aided mine network.

The development of DATAMiNe starts with an initial plan, which bases on the mining equipment manufacturer's vision, and use cases about unified data gathering into a single Data Aggregator. DATAMiNe is developed by ordinary software design methods, by programming a proof of concept test software, and finally by verifying a protocol with the Spin tools. With Spin, it is possible to formally check the interaction between connected state automata. All development steps play a part towards the next implementation phase. That is, the production implementation. The verification model forces attention to the details that otherwise would be ignored in the design phase. The test program implementation helps to choose the cost effective ways in the design.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
ERONEN, MIKKO: Toteutus datasynkronisaatiosta haasteellisen verkon ylitse.
Diplomityö, 48 sivua, 7 liitesivua
Elokuu 2015
Pääaine: Tietoliikenneverkot ja protokollat
Tarkastaja: professori Jarmo Harju
Avainsanat: Teollinen Internet, rajoitteinen verkko, kaivoskoneet.

Tämä diplomityö on linjassa teollisen internetin trendien kanssa. On olemassa lukuisia esimerkkejä tuotteista ja palveluista, joita halutaan kehittää niiden käytöstä kerätyn datan perusteella. Diplomityössä tämä tuote on kaivoskone ja sen huolto.

Tiedon lähettäminen suoraan kaivoskoneesta valmistajalle on ongelmallista, sillä kaivoksissa ei usein ole internetyhteyttä. Joissain tapauksissa kaivoksista saattaa löytyä langaton lähiverkko, mutta toisinaan sellaisen pystyttäminen ei ole kannattavaa. Ainut järkevä tapa hakea käyttödataa kaivoskoneista saattaa olla tiedon siirto muistitikulla. Diplomityössä kutsutaan avustetuksi kaivosverkoksi (englanniksi aided mine network) tapaa, jolla siirretään dataa muistitikulla kaivoskoneelta Internet-tiedonsiirron mahdollistavaan paikkaan. Tällainen paikka voi olla esimerkiksi kaivosalueen läheisyydessä oleva toimisto, johon tulee internetyhteys. Diplomityön ydinongelma on datan kerääminen, siirto ja synkronointi kaivoskoneista valmistajan tietokantaan käyttäen avustettua kaivosverkkoa.

Diplomityössä kehitetään suunnitelma datankeruun toteuttamiseksi. Ratkaisua kutsutaan DATAMiNeksi eli datan koostamiseksi avustetun kaivosverkon kautta (englanniksi Data Aggregation Through Aided Mine Network). DATAMiNe koostuu kolmesta osasta. Osat ovat manageri (englanniksi Manager), reunaviestijä (englanniksi Edge Relay) ja keruukanta (englanniksi Data Aggregator).

DATAMiNen arkkitehtuuri on suunniteltu siten, että se tukee avustetun kaivosverkon helppoa korvaamista langattomalla lähiverkolla tai suoralla internetyhteydellä kaivoskoneeseen. Tiedonsiirtoprotokolla managerin ja reunaviestijän välillä on luotu sellaiseksi, että se palvelee avustetun kaivosverkon tarpeita.

DATAMiNen kehitystä edistetään osin ohjelmistosuunnittelun, osin testiohjelman toteutuksen perusteella. Testiohjelman lisäksi DATAMiNen protokolla verifioidaan käyttämällä Spin–sovellusta. Sovelluksen avulla voidaan tarkastaa tilakoneiden keskinäistä vuorovaikutusta toisiinsa formaalisti.

DATAMiNen kehitys alkaa suunnitelmasta, jonka lähtökohtana ovat kaivoskoneiden valmistajan toiveet ja käyttötapaus datan keräämisestä yhteen tietokantaan. Suunnitelmaa hiotaan osin tekemällä testiohjelma, osin verifioimalla osasien välistä tiedonsiirtoa. Molemmat tavat jalostavat alkuperäistä suunnitelmaa. Verifiointimallin kehittäminen pakottaa vastaamaan kysymyksiin, jotka helposti tulee muuten suunnitelmavaiheessa ylenkatsottua, ja testiohjelman implementaatiotyö auttaa valitsemaan kustannustehokkaat menetelmät. Diplomityön lopussa todetaan, että DATAMiNe on valmis oikeaa implementaatiota varten.

ACKNOWLEDGEMENTS

This M.Sc. thesis would not have been possible, if not for the brilliant supervisors, Prof. Jarmo Harju, and Dr. Juhana Helovuo. Their guidance and perseverance was, and is well appreciated. Without such assets, this thesis would not be anywhere near as good as it is now. I humbly want to thank my supervisors for their laborious efforts.

Thanks also go to my employer, Atostek Ltd. which played big role in establishing the topic of the thesis. Atostek also provided me time for writing the thesis, not to mention the positive push by all the colleagues.

I would like to thank the department of pervasive computing and the whole staff of TUT. The whole campus and its people forms unique compound of sense of community and learning. This thesis ends my journey in the academic field, at least for now. I'm very happy, that the time I spent studying at university degree, was specifically at TUT.

Special thanks go to Mr. Väinö Halla-aho and M.Sc. Jussi Suomi who contributed to my career choice, during, and after upper secondary school. I thank these people for influencing me at the pivot points of my life. Without them, I most probably would not be here, where I am now.

Finally, I would like to thank my family and my girlfriend. My parents, Mr. and Mrs Risto and Anneli Eronen for my personal traits of intent and ambition. B.Sc. Elina Lukkarinen for her limitless support for me. Your indirect influence has been at least as prominent to my graduation as my own direct actions.

Tampere, August 24th, 2015

Mikko Eronen

CONTENTS

LIST OF TERMS, SYMBOLS AND ABBREVIATIONS

Aggregator	A computer software that aggregates a specific type of information from multiple online sources.
DATAMiNe	Data Aggregation Through Aided Mine Network. The system developed in this thesis.
F#	Programming language developed by Microsoft. F# supports both functional and imperative programming techniques.
GUI	Graphical User Interface. Way of using computers via graphical interface elements, such as icons.
Huffman encoding	Type of optimal prefix code. An algorithm for finding such an optimal prefix code was developed by David A. Huffman.
IDE	Integrated development environment. An intergrated collection of programming tools that enables design and implementation of a program.
IoT	Internet of Things. Network of physical things that communicate with various network actors on their own.
iSpin	Graphical user interface for Spin. ISpin is written with Tcl/Tk.
PaaS	Platform as a Service. A category of cloud computing services that deliver applications over the Internet.
PROMELA	Protocol Meta Language. Verification modelling language introduced by Gerard J. Holzmann.
SLA	Service Level Agreement. A contract that defines a service level from a service provider to a customer.
Sneakernet	Transfer of electronic information by moving removable media physically.
Spin	Verification tool for analysing the logical consistency of concurrent systems in formal manner. Uses PROMELA.
Tcl	Tool Command Language. Interpreted programming language by John Oustenhout.
Tcl/Tk	Combination of Tcl and Tk as a GUI tool kit.
Tk	TCL add on, designed for creating graphical user interfaces. Library includes widgets, e.g., windows, buttons and lists.
WPF	Interface rendering system developed by Microsoft. It uses XML to define and link interface elements in GUI.
XML	Extensible Markup Language. Document encoding language.
.NET	Software framework developed by Microsoft. .NET libraries include, e.g., database connectivity and cryptography features.

1 INTRODUCTION

This thesis covers an implementation plan for gathering, aggregating, and synchronizing of work machine data that can be later used in business development. General outline of the problem is that a mining equipment manufacturer wants to gather usage data from the machines they have sold to their customers. This thesis aims to pave the way towards a more efficient data gathering. This document is part of a work that also includes protocol verification and a proof of concept implementation. This chapter presents the preliminary use case, the motivation for the development of this kind of system, and the structure of the thesis.

Figure 1 illustrates how the data gathering scheme works. Usage data is synchronized between mining equipment and manufacturer's data aggregator. Regardless of the customer, usage data is gathered to the same place. Additionally, the synchronization process guarantees that manufacturer's information about the equipment is up to date as long as the communication is uninhibited. The manufacturer can use the gathered data in its analysis, e.g., in research and development, or problem diagnosis.

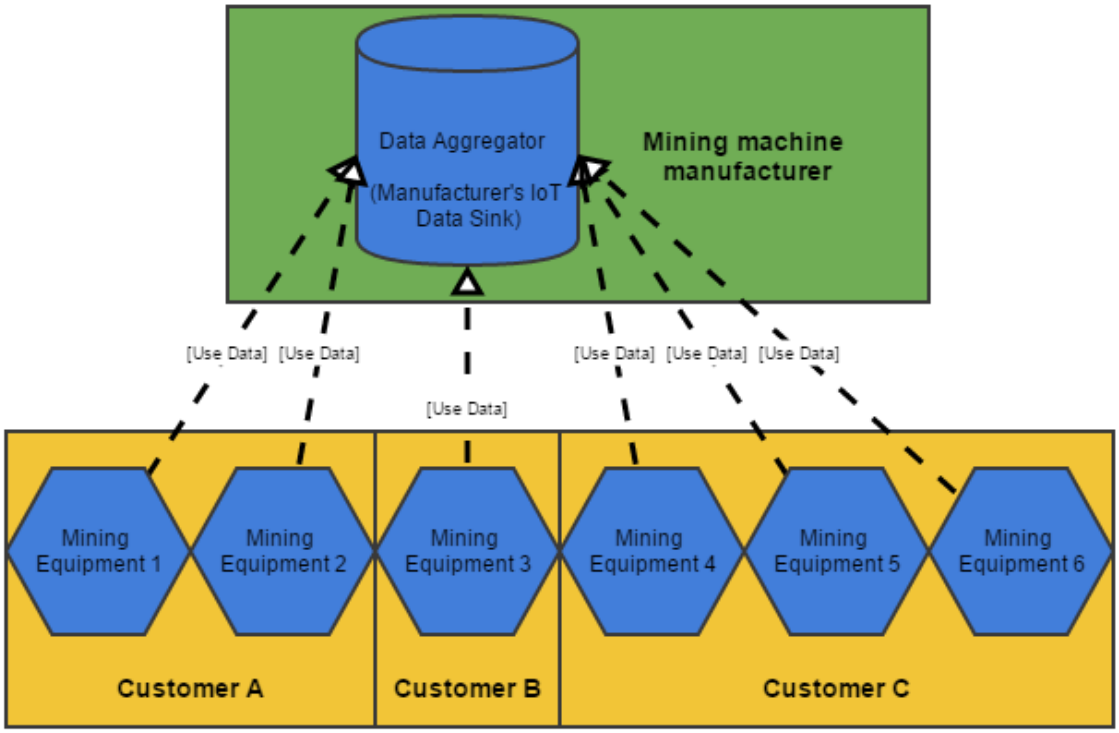

Figure 1: mining equipment usage data is gathered into manufacturer's data sink.

1.1 Motivation for the development

Industries will continue to adopt techniques related to the internet of things. In his article, Chris Preimesberger predicts industrial IoT to take the spotlight in the year 2015. [3] Buzzwords like machine learning and advanced analytics are engaging because technical solutions for thorough data gathering have ripened enough and because business intelligence has been seen as an advantage in competition. "Enterprises that know their customers and potential customers will know success." [24]

However, value-added services cannot work without data synchronization. [2] A data synchronization scheme that gathers mining equipment use data from the field is a preliminary step in order to make more advanced analysis on customer behaviour and machine usage. A moderate use for the data is to, e.g., predict customer's maintenance needs, or to assist in remote troubleshooting. In this sense, the data aggregator in Figure 1 can be seen also as manufacturer's IoT data sink. One of the core tasks in this thesis is the development of an aggregation and synchronization scheme that can be used as a basis for future implementations. The thesis tries to prepare the data gathering so far that production implementation can be started.

1.2 Target environment

The preliminary use case is that mining equipment sometimes works outside of the Internet perimeter, as illustrated in Figure 2. In such a case, data from the equipment is carried out from the mines inside flash drives by human operators. However, some mines are equipped with WLAN connections. Additionally, some equipment operate above ground and they are equipped with internet connections via, e.g., a 3G modem. In this thesis, the scope is focused on the scenario where the mining area is outside of the Internet perimeter, since it is the more challenging one.

If the equipment operates without internet connections, the operators must carry the data in flash drives somewhere, where an internet connection exists. Usually this place is an office building near the mine. There an operator can upload the data from the flash drive into the manufacturer's cloud service. Substituting data networks with human operators brings in many problems.

Mining sites are placed around the globe. Usually, these sites are in remote areas with limited infrastructure. The global aspect is one of the reasons why a cloud platform is considered as the final destination for the synchronizable data. Buying PaaS also releases the manufacturer from the burden of maintaining its own data centre.

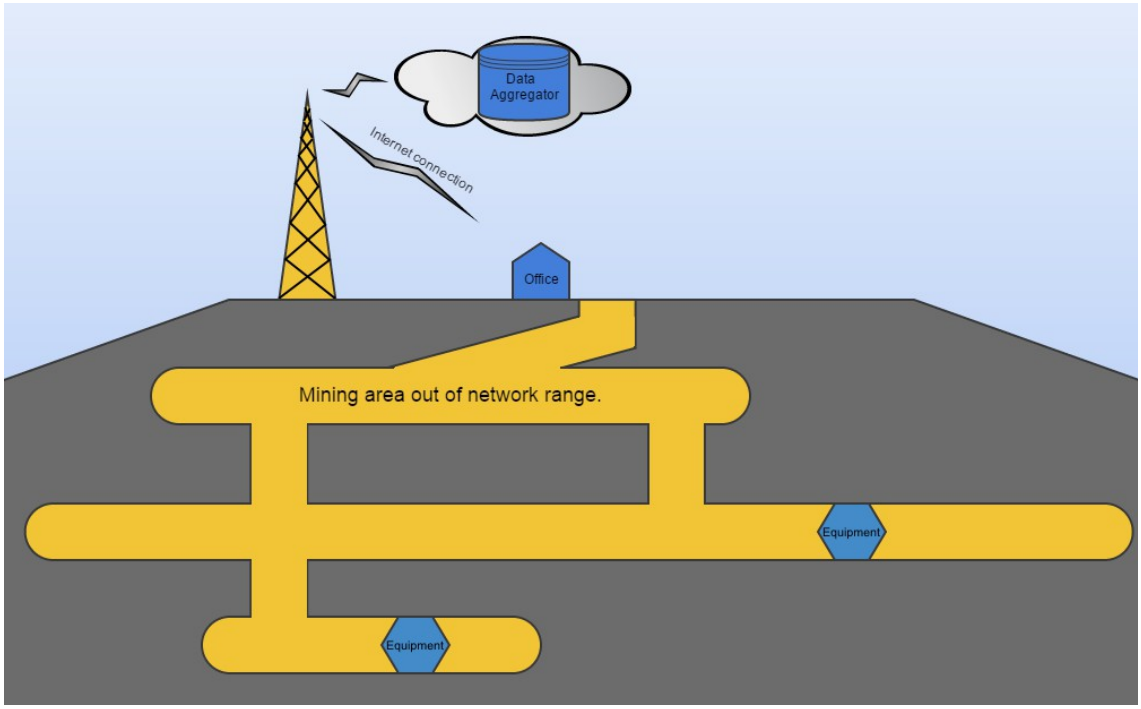

Figure 2: Underground mines can be out of conventional network range.

1.3 DATAMiNe

The main content of this thesis is the design and implementation of a data aggregation system which is called Data Aggregation Through Aided Mine Network, abbreviated as DATAMiNe. DATAMiNe is currently on the proof of concept level. It carries out data synchronization from mining equipment database to another database, where the latter represents a data aggregation end point. In the solution's name, the mine network is said to be aided. That refers to the possible human operator, which can replace the lack of mine network infrastructure.

DATAMiNe software is implemented in the F# programming language. The core functionality is an application layer data synchronization protocol. In its current state, the synchronization works only upstream, i.e., from mining equipment to data aggregator.

1.4 The structure of this thesis

Chapters two through four discuss the development and implementation of the proof of concept software. Chapter two focuses on what are the main problems and related technologies in the development. Chapters three and four tell how a DATAMiNe software is designed, implemented, and how the designed protocol is verified.

Chapters five through six discuss evaluation, results and further development. In Chapter five the proof of concept implementation is assessed, protocol verification results are discussed, and future plans are introduced. Chapter six summarizes the results, and some key development ideas.

2 BACKGROUND

It has been predicted that curiosity towards industrial IoT increases [24]. Enterprises want to know their customers since it is seen as a part of successful business. Topics like advanced analytics and machine learning can lead to better services, but the prerequisite is that the data is gathered from existing customers. Mining equipment vendors are not idle bystanders in reforming their business models.

The general problem in this thesis is how to gather data from mining equipment so that the vendor can use the data for product and service development. Initial goals, placed by the orderer of this work, were comparison of data synchronization algorithms and comparison of cloud service providers. More precisely, the thesis will concentrate on techniques that relate to data synchronization, transmission protocols, and also a concise assessment on how to choose between said cloud service providers.

A unique aspect of this problem set comes from the use case where mining equipment cannot be connected to the Internet. There already exist many platforms that can be used for the synchronization between the cloud and a computer that is directly connected to the internet. In DATAMiNe, synchronization must be done even if the time between messages is very long, or the network is unreliable.

2.1 Data synchronization

Data synchronization is not a new problem but it is still significant. The reason for synchronization is often the same as with replication: fault tolerance in the case of storage location unavailability [27]. Synchronization solutions have become a necessity with consumer mobile devices [2]. Heavy industry will need the same kind of solutions in the near future. Many enterprises, universities, and government departments have built their information systems to be standalone. Problems emerge when operations are being streamlined and dynamic updating of the data among the dispersed databases is not possible [17]. Another use case is the synchronization between web services and mobile devices. Those devices need to run local databases in order to achieve stable data processing in case the connection to the server-side database is lost. [3] The fundamental problem is the same when we want to gather use data from client instances into the manufacturer's aggregation database. The synchronization problem should not be mixed up with database consolidation or system integration problems. Data synchronization does not demand that the data storages are the same in composition.

Data synchronization is a process where the data in one storage is convoyed to another as an identical copy. It can be seen as a treatment process to maintain data consistency [17]. Data synchronization can be divided into two categories: file synchronization and database replication. [5] The synchronization scheme can be tweaked towards certain goals. For example, data that is synchronized does not have to be the whole content of the storage. Synchronized storages can be arranged into some hierarchical structure, or into a more decentralized structure. Synchronization groups can be static or very fluid.

Generally, synchronization needs to be seamlessly integrated, it has to have error recovery and collision detection. Synchronization should also be customizable when needed. These aspects are listed by Coleman relating to the needs of the user. [5] On the other hand, Albright argues that "Synchronization succeeds when it disappears", meaning that best case for the user is when he or she need not pay any attention to the synchronization process. [2]

2.1.1 Middleware and rule base

When looking for general approach to data synchronization, the following example exists: In their conference paper [17] "Research for a Data Synchronization Model Based on Middleware and Rule Base.", JianJun and XiangYun present their model of generic synchronization model for databases. Its three main components are a working file, a rule base and a middleware.

The working file consists of information about database connection parameters, synchronization timing and frequency parameters, and data security parameters. The rule base is an aggregation of synchronization rules, including data transformation, filtering, and mapping rules. The middleware uses a synthesis of working file and rule base information. It has a multi-tier architecture with the following tiers

1. Data access module.
2. Data capture module.
3. Data conversion module.
4. Data transmission module.
5. Security verification module.

The middleware captures the needed data from several sources and forms an XML file, which can be transmitted to the target database. The source can be a single database or a set of databases.

2.1.2 Secured message digest and synchronization through SQL transactions

One example of database synchronization is the following. In the conference paper “An efficient database synchronization algorithm for mobile devices based on secured message digest.” [3] Balakumar and Sakthidevi suggest using only standard SQL queries in the synchronization transactions. This is to create a more general and compatible data synchronization solution compared to most database vendor tools which usually work only with a certain product.

The authors' implementation of this algorithm is called Improved Synchronization Algorithms based on Message Digest (ISAMD). ISAMD does not rely on timestamps when deciding which version of data is correct. Instead, it needs an extra table on each database. That extra table holds columns for primary keys from data tables, relating values, an inconsistency flag, and a mobile device identification code.

2.1.3 OMA DS and Huffman encoded data transmissions

Open Mobile Alliance (OMA) is an acknowledged forum for mobile service enablers. [21] OMA DS is a platform-independent data synchronization standard formerly known as Synchronization Markup Language (SyncML)

Kee-Hyun and Ju-Geon try to optimize transmission packet content in their paper “Efficient Transmission Method for Mobile Data Synchronization Based on Data Characteristics” [18]. The Basis is that Open Mobile Alliance Data Synchronization (OMA DS) encodes its elements by XML or WBXML. WBXML is WAP Binary XML, which was originally developed together with Wireless Application Protocol (WAP) to provide a more compact presentation of XML. The paper's authors use modified Huffman encoding instead, and achieve good results.

2.1.4 Correctness versus availability, and optimistic versus pessimistic strategies

Details of a problem often dictates the software's behaviour. To decide appropriate guidelines in software behaviour, it is not trivial to scale requirements and implications for correctness. In their very fundamental paper “Sacrificing serializability to attain high availability of data in an unreliable network” [9], Fischer and Michael state that constraints on availability are a way to protect consistency. However, availability is a principal reason for deciding to replicate data in the first place. Serializability means that the same transaction sequence results always the same outcome. “The results of transactions should be the same as if they had been performed on some serial order.” Serializability ensures the consistency of the database. Availability dictates that every node can work on their local copy of a data even when the network fails. Serializability on the other hand allows only one such node to work after network failure. Work over data must be balanced between serializability and availability.

In another paper titled “Consistency in Partitioned Networks” [8], Davidson, Garcia-Molina, and Skeen expand on what is the relation between consistency and availability. In their terminology, words “correct” and “consistent” are interchangeable.

The topic is reflected on a situation where synchronizable data is scattered over multiple nodes in a network. Nodes can each make changes to the data. Network is then partitioned. A partitioned network here means that some nodes can be temporarily disconnected. In such a situation data consistency becomes problematic to maintain.

Authors introduce concepts of optimistic and pessimistic strategies in synchronization. Synchronization strategy is pessimistic when system sets data unavailable due to a possible inconsistency. Absolute correctness demands that only one instance works on the data in case of partitioning. Constraints are an easy way to achieve consistency, but often unavailability is unacceptable. Compromising constraints and keeping them effective at the same time requires extended knowledge of the handled information and its nature. At the other end of the spectrum is an optimistic strategy. Optimistic strategy does not limit availability and therefore conflicts on the data can occur.

Authors also give some guidelines on how to choose between strategies mentioned above. For example, if network failures exist over long periods of time, a pessimistic strategy is a better choice than an optimistic one. If the transactions are short and the work load variance is small, an optimistic strategy works better.

2.1.5 Tombstones and reconciliation

Synchronization solutions that enable data object modification from multiple nodes, need to consider situation where synchronization is slow and alterations are made virtually simultaneously. In his paper titled “A Java Framework for Mobile Data Synchronization” [4] Cohen writes about a peer-to-peer synchronization framework for mobile devices written in Java. The implementation makes synchronization decisions or reconciliation based on data object history. Each synchronizable data object has history information, which is carried within. When a data object is altered somewhere in the network, new versions of the data object are created. A conflict emerges if two changes into same data object have been made and if they have versions neither of which is later than the other.

Tombstones are data objects that are deleted in some but not in all synchronization nodes. Tombstones are necessary because without them a synchronization process may try to update an object that does not exist any longer. However, accumulating tombstones can become a problem. Algorithms that rely on informing all synchronization nodes in order to remove the tombstones are not well suited for weakly connected mobile devices.

2.2 Challenged network and other special conditions

One could see cloud as a simple answer to the synchronization problem. However, communication between mining equipment and the edge of the Internet falls outside of any cloud perimeter. Even when using a cloud as data sink or data aggregator, the challenged nature of a network between the cloud and mining equipment needs some separate solution from a general cloud solution.

A network described as challenged can be reduced to a sneakernet if an underground WLAN is not installed. There are successful examples of how to install wireless local area networks into mines [12] [28], but not all mining companies invest into WLAN infrastructures. For this reason, the use case on which the implementation introduced in this thesis is built for, expects the mining equipment to be unreachable by any internet services. Supplementing a method of delivering data between mining equipment and the Internet has been transporting flash drives with the operators, hence the sneakernet. This method must be sufficient for data synchronization also.

Human error brings new layers to the problem, since latency can extend even into weeks, flash drives can be lost, or they can change arriving order during the transportation. By data transfer terms, these circumstances can result in a large packet loss rate and long delays.

2.3 Protocol engineering

Protocol is originally a diplomatic term but in communication technology it portrays a specification for network communication. A protocol can describe low-level details linked with some type of physical transmission, or it can describe high-level information such as the possible set of messages between network nodes [6]. Therefore, communication protocol engineering can be about creating a set of rules that defines viable messages and error handling, in order to enable robust information sharing between two or more separate entities in a network.

Entities in a network have shared connection over a medium that can be anything physical and modulatable. On top of connectible medium, layers of protocols can be established. Two popular protocol layer models are OSI model and TCP/IP stack model. Figure 3 shows how the two models map to each other. TCP/IP model is less detailed than the OSI model but it is more usable when working with the hugely popular TCP/IP suite, which is the foundation of the modern Internet.

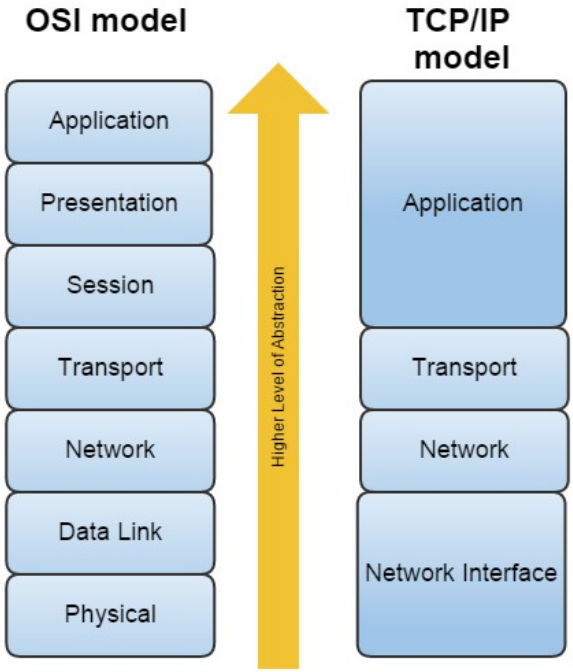

Figure 3: Popular protocol layer models, OSI and TCP/IP. Level of abstraction rises towards higher tier layers.

As with software engineering, protocol engineering also benefits from building higher level abstractions on top of each other. In protocol layer models the communication abstraction level rises when moving from Physical or Network Interface layer up towards the Application layer. A higher abstraction system means that it needs services from a lower layer systems to operate, but that it can implement a more advanced service.

Considering general approach to the protocol engineering, John Spragins lists seven aspects or problems to consider. [26]

- Resolving incompatibilities of equipment
- Coordination of sender and receiver
- Maximizing reliability and freedom from errors
- Optimizing performance
- Minimizing cost
- Network management

These problems can be encountered at different stages of design. Classifying the problems is good way to form mental image of mutual traits in protocol design.

Protocols that operate in multi-agent environments and often only above Transport layer, are called interaction protocols. The purpose of an interaction protocol is to enable agents to complete their tasks through exchange of information, co-operation and coordination. Interaction protocol engineering is based on communication engineering but it is more specialised on its field of problems. [16] Marc-Philippe Huget and Jean-Luc Koning suggest a process for interaction protocol design, which is similar to the waterfall model [25] in software engineering. The model consists of five steps that must be walked through: Analysis, Formal Description, Validation, Protocol Synthesis and Conformance Testing. Figure 4 illustrates the suggested model.

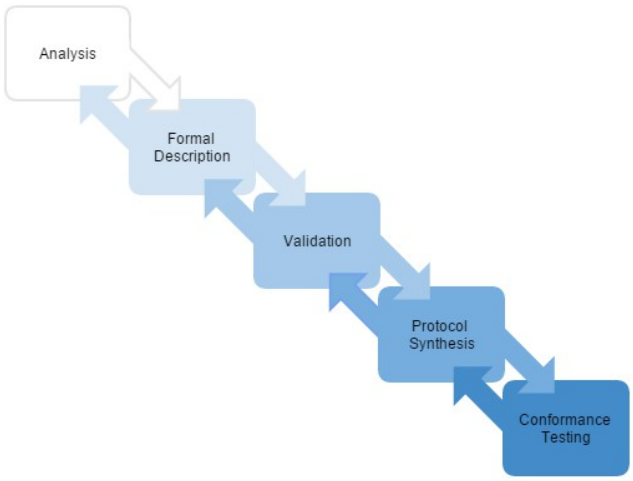

Figure 4: Marc-Philippe Huget's and Jean-Luc Koning's suggestion for interaction protocol design stages.

This model is more rigid than the continuous integration models often used in modern software development, but that is well justified. Traditionally, continuous integration and rapid deployment have not been included in protocol engineering toolbox. Protocols are often seen as a critical links between systems, and therefore deployment of an insufficiently validated protocol can halt production and cost great amounts of money.

2.4 Validation and verification

Because protocols can act in critical roles in systems, it is natural that the protocols must be tested before putting them into the production. Central tools for this are verification and validation. Both are tools to confirm that a software fulfils the desired functionalities. In literature, it is not always clear what an author means precisely when using these two terms. For the sake of accuracy, the terms are defined here.

Haikala defines the terms as follows [11]. In verification, the software is tested against specification, i.e. “are we building the product right”. Validation on the contrary, tests the software against its purpose, i.e. “are we building the right product”. Verification can be defined easier, its scope does not have to cover the whole system, and it can often be executed in separated parts. The value that the verification creates, depends highly on the specification and its quality. For example, poorly defined terms can make the verification impossible. Validation is executed on more complete software in more authentic environment. This makes validation more thorough, but also more laborious operation. In Haikala's book, verification is done after every phase in the software development process. Validation is reserved for end phases, although, validations can be executed earlier with, e.g., prototypes.

2.5 Selecting a cloud provider for Data Aggregator

DATAMiNe's Data Aggregator needs a platform. Cloud services are an attractive solution since they offer automatic scalability and outsourced hardware maintenance. Costs for provisioning new hardware can be lower and unplanned disruptions to the live services can become rarer due to seamless relocation [22]. The service prices are kept low by offering large volumes [19], and customer interfaces are often web pages, providing automated set of services. Besides the obvious advantages of cloud computing, one should consider a few aspects of the market.

Today, there are many players on the cloud provider market. In the future, the market may become an oligopoly of a few big players, but the consolidation of the market dominance will take time [10]. Currently, no single company has the advantage over the market [13]. In this situation, deciding on which provider might be the best, can be very complicated. Wayne Pauley tries to solve this problem in his article “Cloud Provider Transparency: An Empirical Evaluation” [23]. Pauley's method of assessment is to use data from published web sources. He scores the cloud providers by use of standards, best practices, policies, procedures, and contractual service level agreements (SLA). Also, his preassessment includes a question whether or not each cloud provider has over five years of experience from the market. In the assessment, Savvis, Terremark and Microsoft were scored highest with their cloud products.

If a company plans to use a cloud service for an extended period of time, choosing the provider should be done so that the migration effort from one provider to another will be kept minimal. Otherwise, cloud provider should be trustworthy to continue their business. Assessing provider's services is problematic in the same way than assessing cloud provider's business in general, and it is particularly hard in a relatively new market [10]. Other strategy can be to avoid vendor locking and look for providers with open APIs, or to make sure that migrating the data and porting applications can be done when needed [22].

Pauley's cloud provider assessment takes service level agreements into consideration. However, it does not consider whether or not the customer can enforce those agreements. Bernt Ostergaard points out in his article “How to select the best hybrid cloud provider for your firm” [22] that "for the customer it is safer to store data with a well funded provider with the resources to make your SLA guarantees enforceable." In other words, when running into problems in the quality of cloud service, financially sound provider is more likely to be able to reimburse the resulting losses.

2.6 Data security

Data security should be considered when talking about systems that handle confidential or otherwise valuable data. With the DATAMiNe use case, there are multiple critical points that need special attention.

One such point is the human factor that is necessary when moving flash drives in and out of a mine. In their article “Surviving Attacks in Challenged Networks” [7] Cucurull et al. propose a security framework for disaster situations where cryptographic methods are energy-wise too expensive and trust establishments are too hard to achieve. Even though the environment in DATAMiNe's case is not a disaster area, trust establishments cannot be used there either. Especially the human element in taking the flash drives in and out from the mine can be seen problematic. The assisted network can be seen comparable to a hastily formed network (HFN). The suggested framework in the article bases on observations which can be used to detect, diagnose, and finally mitigate unwanted behaviour inside the network.

From equipment user's point of view, there is a possible information security hazard in sending out distinctive mining machine use data. For example, with services like Dropbox, one should be advised not to share their business critical information [20]. Things like these should be paid attention to in DATAMiNe.

2.7 Industrial Internet and the prospective evolution of IoT

As industrial Internet of Things applications are taken increasingly into use, the private players on the field try to guarantee the longevity of their business. Industrial Internet Consortium or IIC for short, is a joint forum for promoting best practices, as well as accelerating growth of IoT in industrial use cases. [1] Their vision is to combine the Information Technology or IT, with Operational Technology or OT. OT being, e.g., traditional industrial production facilities.

IIC has released Industrial Internet Reference Architecture document which tries to "initiate a process to create broad industry consensus to drive product interoperability and simplify development of Industrial Internet systems that are better build and integrated with shorter time to market, and at the end better fulfil their intended uses." The main points of the document are stakeholder viewpoints and their relation to security issues. Listed viewpoints are the following:

- Business Viewpoint that considers business vision, values and objectives.
- Usage Viewpoint that considers activities of human or logical users.
- Functional Viewpoint that considers components, their interrelation and structure.
- Implementation Viewpoint that considers technologies needed to implement components.

Throughout the document, topics are mirrored with the viewpoints. Security is one of the main concerns and that is justified by the fact that OT security often relies on physical security, isolation of the system and the obscurity of proprietary communication protocols. Industrial Internet systems, on the other hand, are often connected, distributed, and deeply integrated with enterprise systems. The document argues that security cannot be treated as a separate concern.

The Industrial Internet Consortium tries to avoid fragmentation and a loss of interoperability in the IoT market in the future. It is not yet certain whether or not the market acknowledges the IIC's vision. [1]

3 DESIGN OVERVIEW

With DATAMiNe, the preparatory work before synthesis of the results is twofold. After the initial analysis of the concept, test program is created to act as the proof of concept. Additionally, the protocol used in the aided network is verified formally. Figure 5 illustrates the idea.

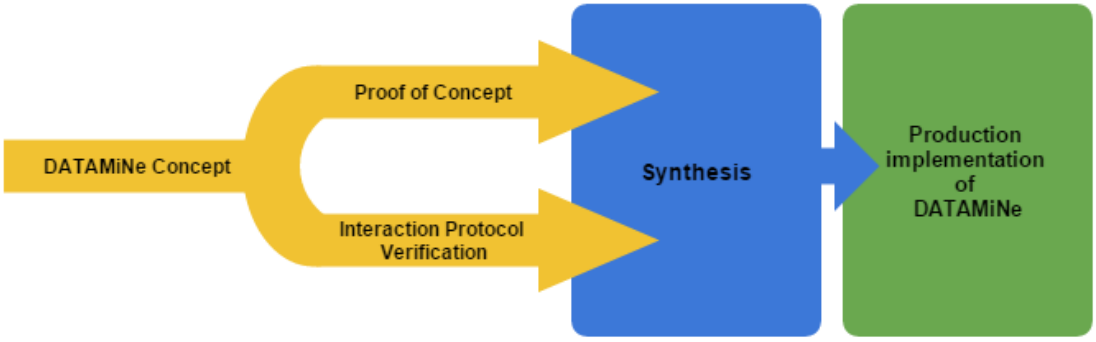

Figure 5: The approach to develop DATAMiNe includes proof of concept and interaction protocol verification.

The synthesis in Figure 5 does not mean the same thing as in the Huget's and Koning's *Protocol Synthesis* stage [25]. In their model, the *Protocol Synthesis* means implementation of a protocol. In Figure 5, the synthesis means the composition of the lessons learned during the creation of the proof of concept and the interaction protocol verification.

The general guidelines for the DATAMiNe concept come straight from customer requirements and the use case. The proof of concept software's aim is to be able to operate in the use case scenario. The proof of concept nature affects how different aspects are emphasized in this stage.

One emphasized aspect is to minimize the data that might be left in portable drives. Portable drives are necessary when substituting WLAN or Ethernet but due to human error data might be left lingering in those drives. This can happen e.g. if a drive has been lost, changed, or abandoned without proper reformatting and overwriting. Data left lingering into the transport medium is rarely considered a problem in data communication but when the medium is a mass storage device, obvious problems arise, e.g., the data can fall into wrong hands.

Another emphasized point is data security. Unauthorized access to flash drives is easy. Even if a flash drive is not misplaced, a human transporting the drive might not be authorized to read the data, and therefore it must not be in plain text.

Relating to networking views in the mining industry [12] [28], the messaging paradigm must be such that it abides to developing data communication networks. In practice, when new mining equipment are expected to work with, e.g., WLAN or 4G, DATAMiNe should drop the flash drive aided data transportation. However, the target is not to do this dynamically. Rather, DATAMiNe should be configurable in this aspect. Another configurable aspect should be whether the synchronization should be triggered by polling, or triggered programmatically from other parts of the system beside DATAMiNe.

DATAMiNe needs two modules that can operate as separate agents. The first one is called Manager and it works within mining equipment software to generate synchronization packets. The second is called Edge Relay and it consumes Manager-generated packets and uploads them into Data Aggregator. Data Aggregator is the service that hosts manufacturer's IoT data sink, and it can be considered to be a third module. The Edge Relay can be placed on the flash drive, where it can run itself with auto run, or it can be placed beside the mining equipment software and Manager, if an internet connection is available. The Edge Relay can be seen as a mobile middleware, which means, that it is not tied to any single operation location.

3.1 Data

Target data consists of mining equipment usage data, which is originally written on databases inside the equipment. DATAMiNe is a system that gathers this data into one place where it is accessible by the manufacturer. The place where DATAMiNe gathers the data is generically called Data Aggregator. Within Data Aggregator, there is a database which includes all the mining equipment databases. Each included database has a state table, which is used to coordinate data that is yet to be synchronized into the Data Aggregator database. Figure 6 illustrates the composition.

Combined data inside the aggregator database is segregated by individual equipment identification. Segregation means that the equipment database contents do not mix when combined. It also means that the aggregator database is slightly different from mining equipment databases because it needs a structure, which enables the segregation.

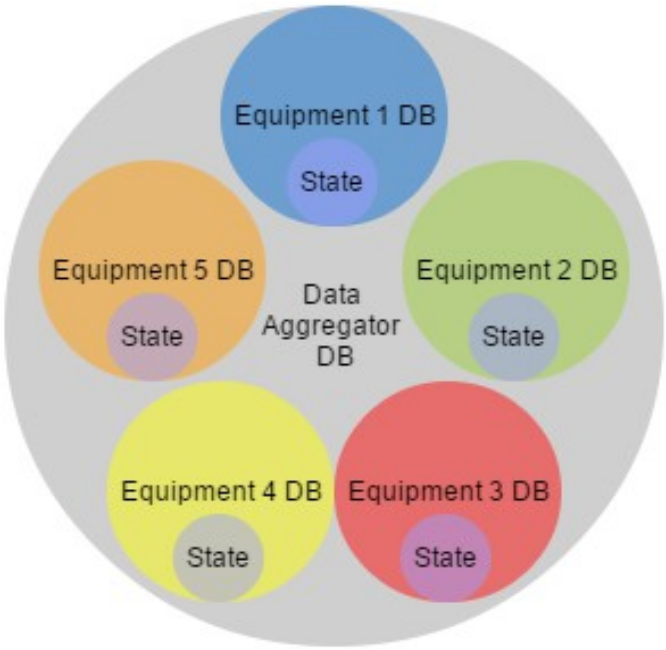

Figure 6: Data Aggregator holds database that is supposed to hold all the data of each equipment database. Each equipment database holds information about synchronization state.

State of each database includes information about the last batch of synchronization. State information content is settled to be based on the mining equipment clocks. Basically, state is designed to be a three-column table within all equipment databases. The columns are:

1. Equipment identification.
2. Last initiated synchronization timestamp. (Mining equipment database state.)
3. Last successfully ended synchronization timestamp. (Data Aggregator state considering to this certain equipment database)

This state structure can be used to ensure that only relevant data is sent through the communication medium at any given time. Using mining equipment's internal clock brings some risks. For example, if the clock resets for any reason, extra data is bound to be sent through the communication channels. In earlier stages of DATAMiNe development, implementation was tried to be designed without timestamps as suggested by Balakumar and Sakthidevi [3]. It was evaluated however, that advantages were insignificant compared to the complexity that a design without timestamps required. Data tables should be marked for synchronization in the endpoint where data is generated, i.e., Manager's code. This is the same as filter rules in Jianjun L's and XiangYun Zheng's model. [17]

Data that is transmitted over a medium is inevitably morphed somehow. As Kee-Hyun Park suggested in his article, there are benefits to be gained if the transmitted packets can be kept compact [18]. However with DATAMiNe, data integrity is more valuable.

3.2 Software components

Main function of the Manager is to generate synchronization packets. The packets are consumed by Edge Relay. Edge Relay picks up the packets and uploads them into Data Aggregator. Protocol wise, DATAMiNe's functionality is limited in OSI model's three top layers. Figure 7 illustrates the DATAMiNe interaction in OSI model. The OSI model is used here, because it differentiates between tasks above the Transport layer in a way that is relevant here.

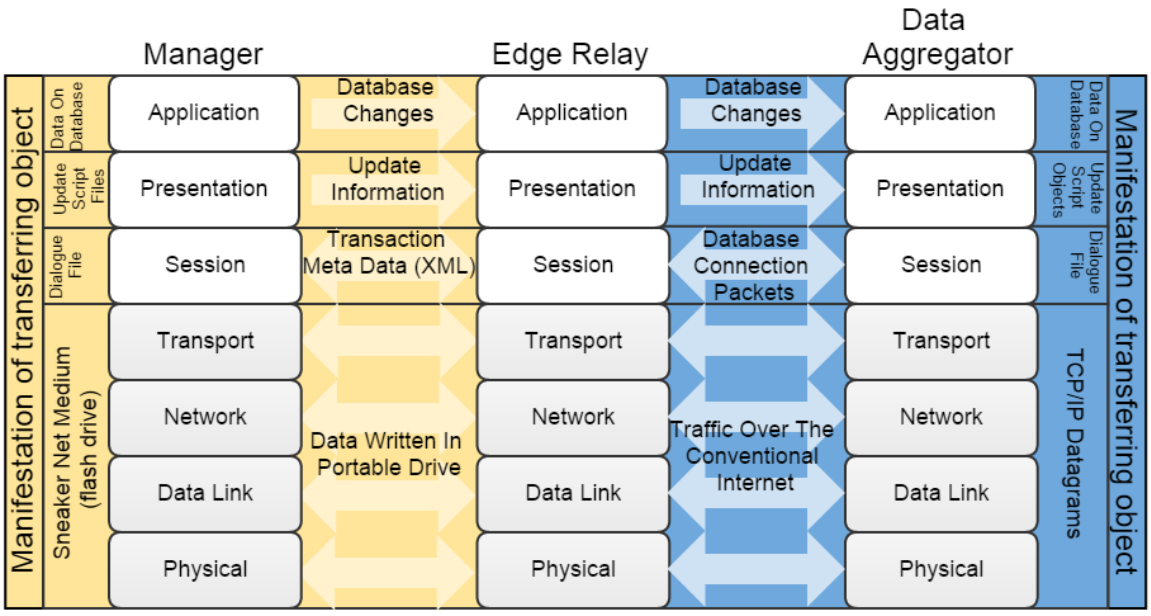

Figure 7: DATAMiNe consists of Manager, Edge Relay and Data Aggregator which fulfil upper layer roles of the OSI model.

Up to the Transport layer, DATAMiNe communication is done by a portable flash drive between Manager and Edge Relay. Between Edge Relay and Data Aggregator, the same level of communication is done with the TCP/IP suite.

Manager uses a file named Dialogue to establish a connection between itself and Data Aggregator and in the OSI model it fills the Session layer role. Dialogue is transported with other data in the flash drive. Dialogue is written in XML and it contains a meta section about active mining equipment. Edge Relay uses Dialogue to make checks. If certain equipment is not listed in meta section, acknowledgement packets concerning that equipment instance will not be sent with that flash drive. In Dialogue there is also a list of update files and a short trace of transactions, where activities concerning that particular flash drive can be read. The design of Dialogue file is illustrated in Figure 8. Session layer between Edge Relay and Data Aggregator covers database connection.

```
<?xml version="1.0"?>
<Dialogue xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd=
"http://www.w3.org/2001/XMLSchema">
  <metasection>
    <EndPoint>
        <machinename>testMachine</machinename>
        <machinestate>
            2015-01-26T12:48:46.725099+02:00
        </machinestate>
        <dbstate>0001-01-01T00:00:00</dbstate>
    </EndPoint>
  </metasection>
  <filehashes>
    <UpdateFile>
        <filename>
            testMachine1986-04-13_00-00-00TO2015-01-23_15-26-28
        </filename>
        <hash>NOTREALHASH</hash>
        <intervalbegin>
            2015-01-23T15:26:29.4276525+02:00
        </intervalbegin>
        <intervalend>
            2015-01-23T15:26:29.4276525+02:00
        </intervalend>
    </UpdateFile>
  </filehashes>
  <trace>
    <Transaction>
        <localtime>
            2015-01-23T15:26:29.4336577+02:00
        </localtime>
        <direction>UP</direction>
        <msgtype>DATA</msgtype>
        <payloadname>
            testMachine1986-04-13_00-00-00TO2015-01-23_15-26-28
        </payloadname>
    </Transaction>
  </trace>
</Dialogue>
```

Figure 8: The structure of Dialogue file

As seen in Figure 7, the presentation of the database change is update information when travelling through DATAMiNe communication and manifestation of that update information is an update script file between Manager and Edge Relay. Edge Relay uploads update script file onto Data Aggregator's memory, which handles it as an object. Originally OSI model's presentation layer was reserved for syntax and semantics mappings between formats. The way it is used here is unconventional but useful. Presentation layer here confirms that the update information can be translated into database changes.

On the application level, Edge Relay is invisible. Edge Relay just relays the database changes that are presented as update information from Manager to Data Aggregator.

3.3 Protocol functionality and logic

The protocol between DATAMiNe nodes can be very optimistic when considering availability versus correctness dilemma. Since data flows only in one direction, synchronization scheme becomes very simple. In many protocols, two channels of communication can be discovered. The first one is control channel which informs what information must be transferred and the second channel is for the actual data transfer [27]. In DATAMiNe, those two channels exists as well. The control channel's medium is the Dialogue and the update script files are medium for the data transfer channel.

Figure 9 shows the Edge Relay's logic as a control flow diagram. General guidelines of this model persist in the proof of concept implementation and the protocol verification phase. After initialization, the Edge Relay goes through a polling loop, which aims for update file upload from flash drive to the cloud Data Aggregator. Polling consists of two checks. First is internet connection check using ping, second is checking the existence of uploadable files using the Dialogue. If either of these checks does not produce beneficial outcome, a wait timer is run.

If the needed prerequisites are fulfilled, the Edge Relay proceeds to upload i.e. import the upload files. After finishing, Dialogue is updated to carry acknowledgement back to the mining equipment database. After updating the Dialogue, Edge Relay does garbage collection and removes successfully imported update script files. Next, the loop is restarted.

One restriction of this design is that the flash drive that carries the acknowledgement must have committed new data to the Data Aggregator. This is because every Dialogue is always bound to a single flash drive. The Edge Relay cannot relay acknowledgements without knowing, which equipment the current flash drive is associated to. In other words, flash drives must be first initiated at the Manager's end point.

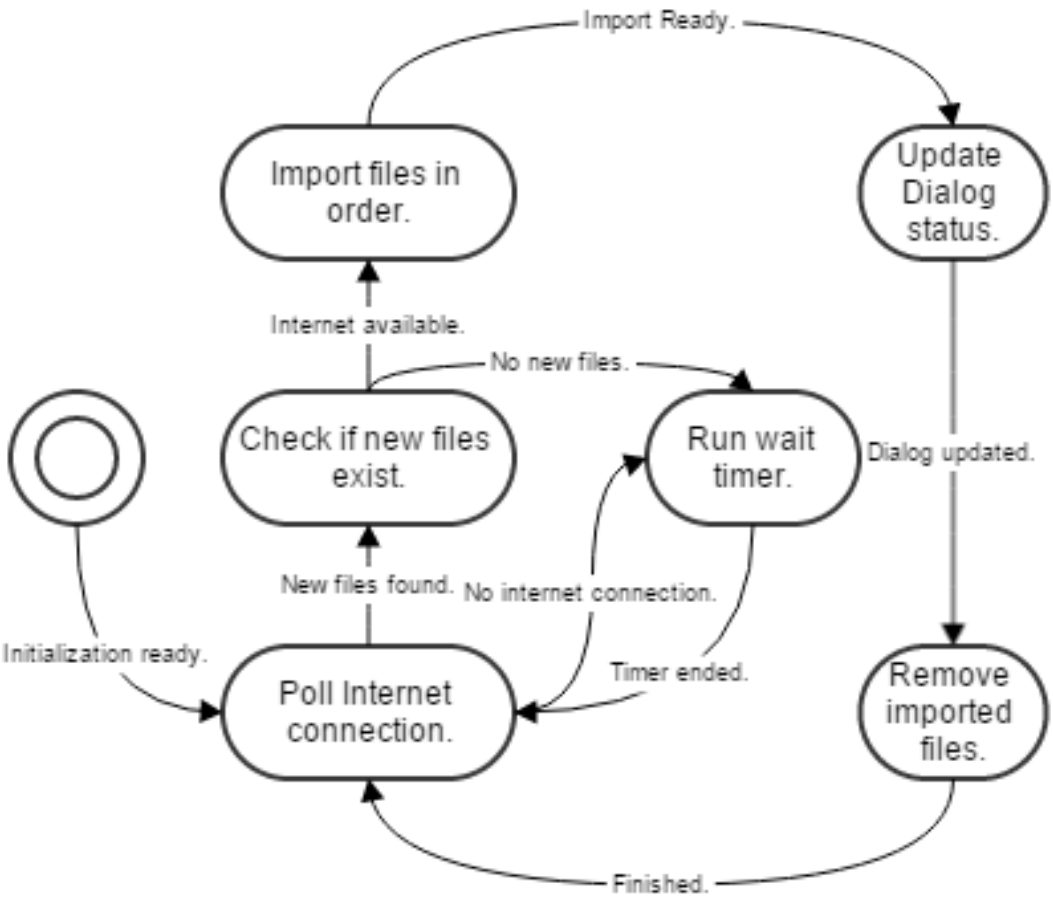

Figure 9: Edge relay's control flow.

Figure 10 Shows the Manager's control flow. The main loop of the Manager starts with checking the Dialogue for finished transactions. If acknowledgements are found, the program removes the respective files from a temporary folder where the update script files are gathered. After this, the Manager checks if new data has been inserted into the mining equipment's database. If no changes are found, a wait timer is run and the loop restarts.

When database changes are found, new update scripts are written. New script files are moved into the temporary folder, and the Dialogue is updated to include the transaction of the new files. Afterwards the loop is restarted.

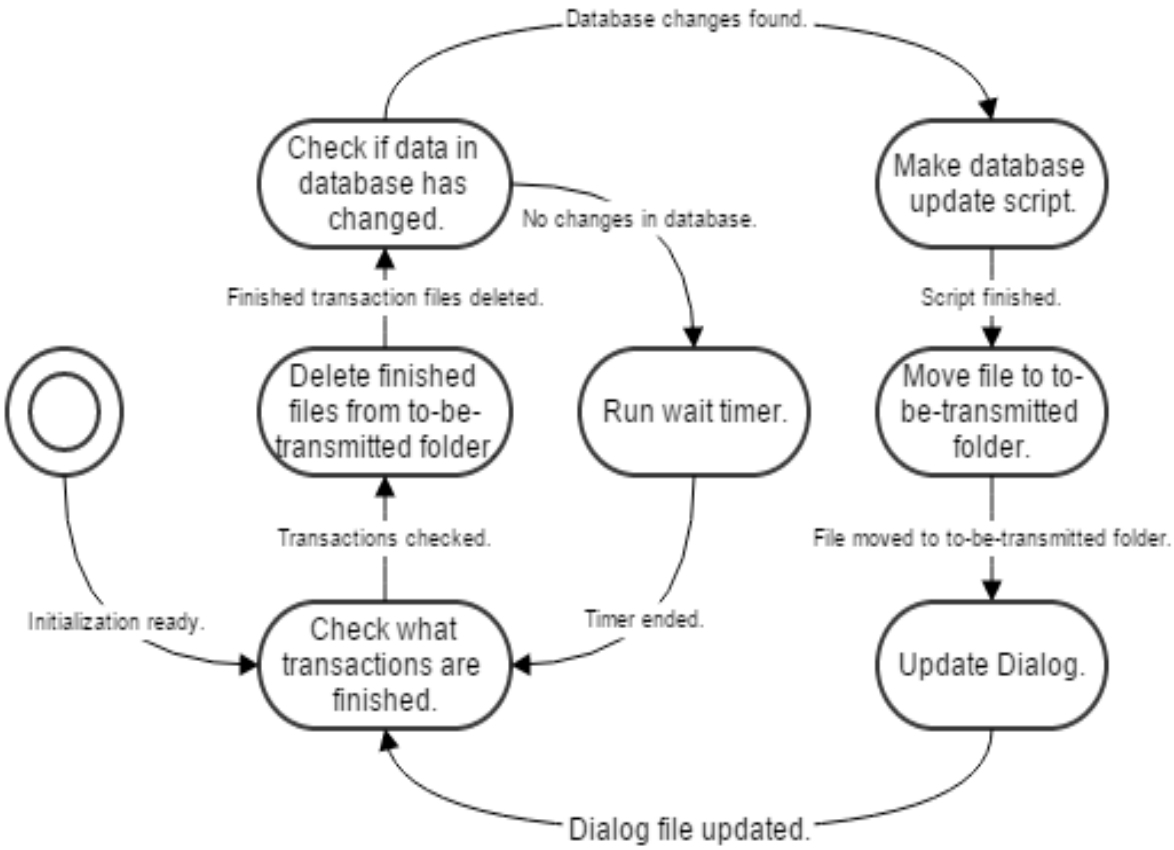

Figure 10: Manager's control flow

Finally, Figure 11 shows how the previously presented Edge Relay and Manager interact, if they both are active simultaneously, in the same device, and host computer is connected to the Internet. This kind of configuration of DATAMiNe can exist when the mining equipment is supplied with an Internet connection. The configuration eliminates the need for flash drives, and therefore the need for aided network as well. The figure shows that both agent software are polling the same way as they would when separated. Eliminating the polling would be more efficient, but for its simplicity, minimal configurability is devised. One of the few changes that this kind configuration makes, is a relocation of transaction folder. The folder would normally be located on the flash drive, but in this case it is on the host computer's hard drive. This configuration is the reason why files, especially the Dialogue file, must be locked when one agent module is handling it. Incomplete files cannot be opened for reading or writing by more than one agent at a time.

With the aided network the sequence diagram, i.e. Figure 11, would be little different. In the diagram, the Edge Relay is started at the same time as the Manager. With the aided network the activity of the Edge Relay would be restricted only to the moments when the flash drive visits the host computer with the Internet connection. Also, the arrows pointing from the Manager to the Edge Relay and back would form steeper slopes in case of the aided network. In other ways, diagram would stay the same.

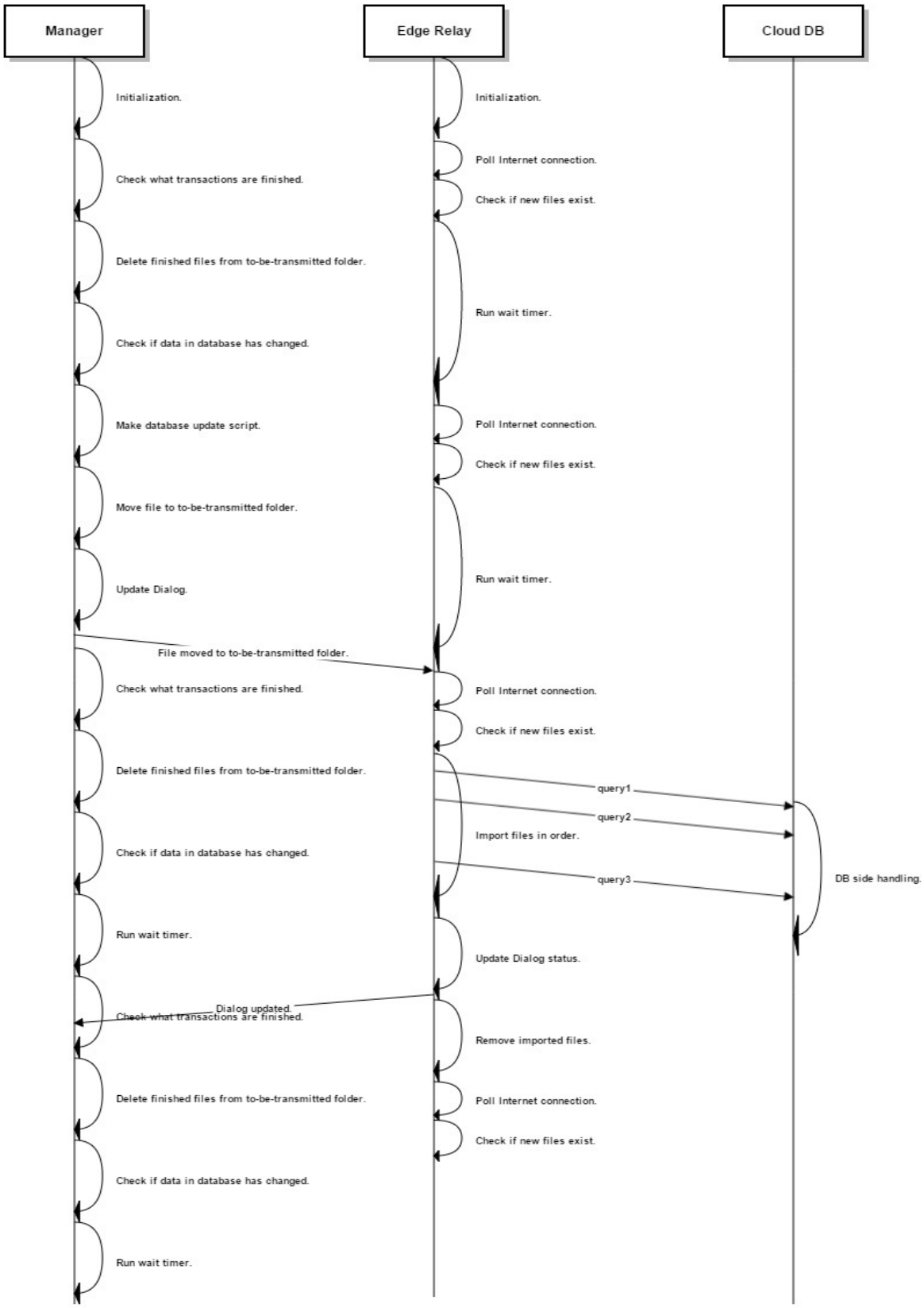

Figure 11: Sequence diagram between Manager, Edge Relay, and Cloud DB

4 CONCEPT TESTING AND VERIFICATION

This chapter explains how the initial design was put into use in the proof of concept implementation and during the verification phase. What techniques and technologies were used, and how the design was evolved during the development.

The first two sections include meta information about techniques and technologies used. Third section talks about a tool called Spin. It is meant to be an introductory information preceding the verification section. Last two sections introduce the details of the proof of concept implementation and the verification, respectively.

4.1 Used techniques

As a workflow, the development of DATAMiNe follows quite well Marc-Philippe Huget's and Jean-Luc Koning's suggestion for interaction protocol design presented in Section 2.3. Considering verification of DATAMiNe protocol, it would fit into phases two and three i.e. “Formal Description” and “Validation” respectively. The proof of concept implementation is disconnected from this workflow since it is parallel to the verification as Figure 5 in Chapter 3 shows, but it has no formal description and it is not validation either. Workflow technique in this case would best be described as an ad-hoc combination of what seems to be essential to build a data synchronization system.

Considering Section 2.4 and difference between verification and validation, it is not explained why the phase's name is *validation* and not *verification.* Marc-Philippe Huget's and Jean-Luc Koning's paper uses both terms seemingly interchangeably [16]. This thesis uses the term verification when talking about the formal testing, because the real environment and the formal test environment are arguably very different. However, arguments exists for using either term.

In addition to the graphs presented in Section 3.3, there is a module graph, Figure 13, about the proof of concept implementation of DATAMiNe in Section 4.4. Other than that, there is no technique worth mentioning that is related to the proof of concept.

Protocol verification is done using a formal and exhaustive method. The verification is machine assisted.

4.2 Used technologies

With the proof of concept implementation, Microsoft Visual Studio 2013 was used as the integrated development environment (IDE). F# 3.1 was chosen as the programming language. Target framework was .NET 4.5.2. Windows Presentation Foundation (WPF) was used in a test user interface. Additionally, a few extensions were used with Visual Studio. Extensions included Visual F# Power Tools, F# Outlining and Visual F# 3.1.2. Mercurial was chosen as the version control system.

With the protocol verification, Spin model checker was used. Model was made with PROMELA and user interface was iSpin.

4.3 The Spin model checker

Spin model checker is a generic modelling and verification system for asynchronous process systems. Spin can verify interactions between state machines defined with PROMELA. PROMELA (protocol meta language) is a language for building verification models. It includes primitives for process creation and primitives for interprocess communication [14].

The principal PROMELA developer Gerard J. Holzmann writes that models written in PROMELA language differ distinctly from models that are written in programming languages. A verification model represents an abstraction of a design. It leaves out everything extra. A verification model also often contains features not found in the implementation, such as environmental behaviour or correctness properties. One of the key elements in Spin and PROMELA are branching blocks, where the course of execution is decided randomly between multiple possibilities. [14] However, randomizations in these cases are not purely random. For example, if there are three enabled branches in *do*-block, Spin first randomizes if the execution proceeds to the first branch or not. If the latter is picked, then a second randomization is done between the second and third branches. In this way the probabilities between the three branches are 50%, 25%, and 25%. [15]

Spin offers features for PROMELA syntax checking, interactive simulation, and verifier generator. Interactive simulation lets the user pick simulation paths by hand. Verifier generator goes through the whole space of possibilities that can happen in the simulation. In this way the user can be sure that model has been verified completely. If the Verifier generator comes up with a problem branch in the tree, it will produce a counter-example that can be run again in the interactive simulator. With these features, Spin can spot deadlocks, non-progressive loops, and any assertion failures that are expressed with the PROMELA model. [14]

4.4 Proof of concept implementation

Proof of concept implementation consists of working Manager and Edge Relay modules, which can be controlled from a shared user interface for testing. The test interface has buttons for initialization, launching and shutting down the modules. In addition, it has text area which is filled with actions that different modules and components make. Figure 12 shows a screenshot of the test user interface.

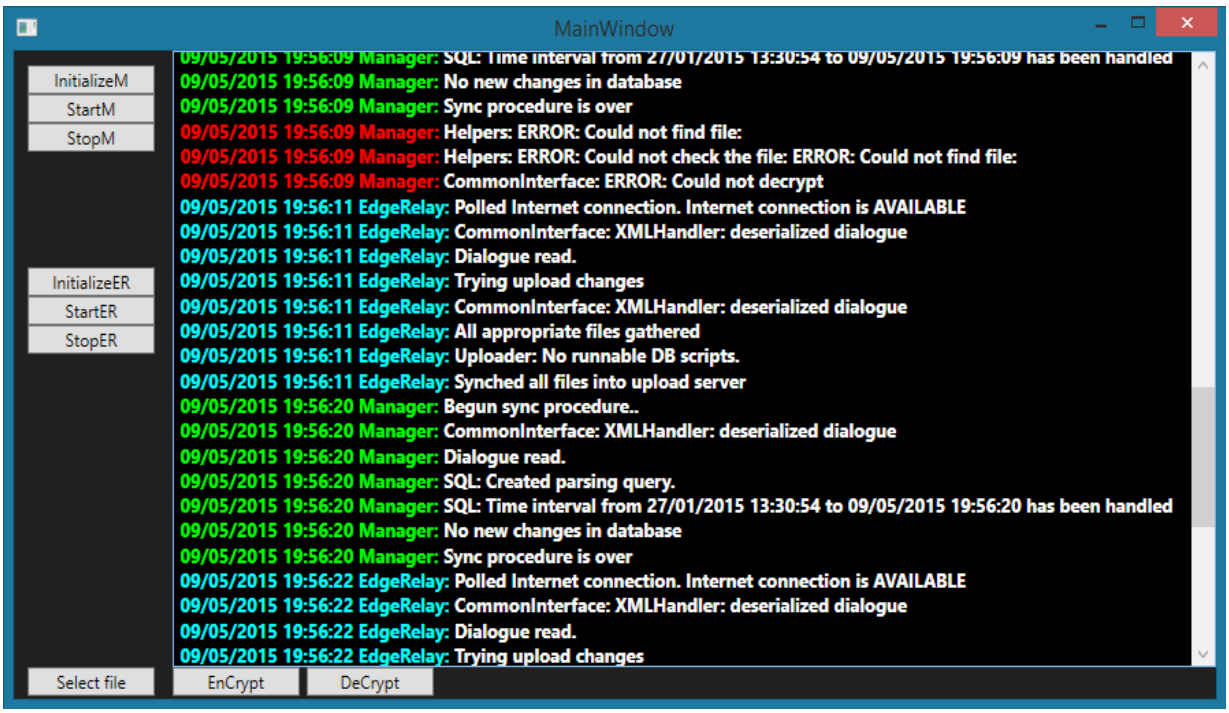

Figure 12: Test user interface for DATAMiNe proof of concept.

The aim of the proof of concept implementation was primarily to show that creation of data synchronization is relatively easy to do. The core functionality of the software is to synchronize data from one database to another. Interfacing a cloud service or encryption was not implemented, although the user interface has encryption related buttons attached.

The inner modules of the proof of concept implementation are presented in Figure 13. The main modules are *M* and *ER* i.e. Manager module and Edge Relay module. Additionally, *Common* module is used by both, *M* and *ER* modules. Lastly, since the proof of concept implementation software is also for testing purposes, it needs a GUI, and for this reason *TestForm* module exists.

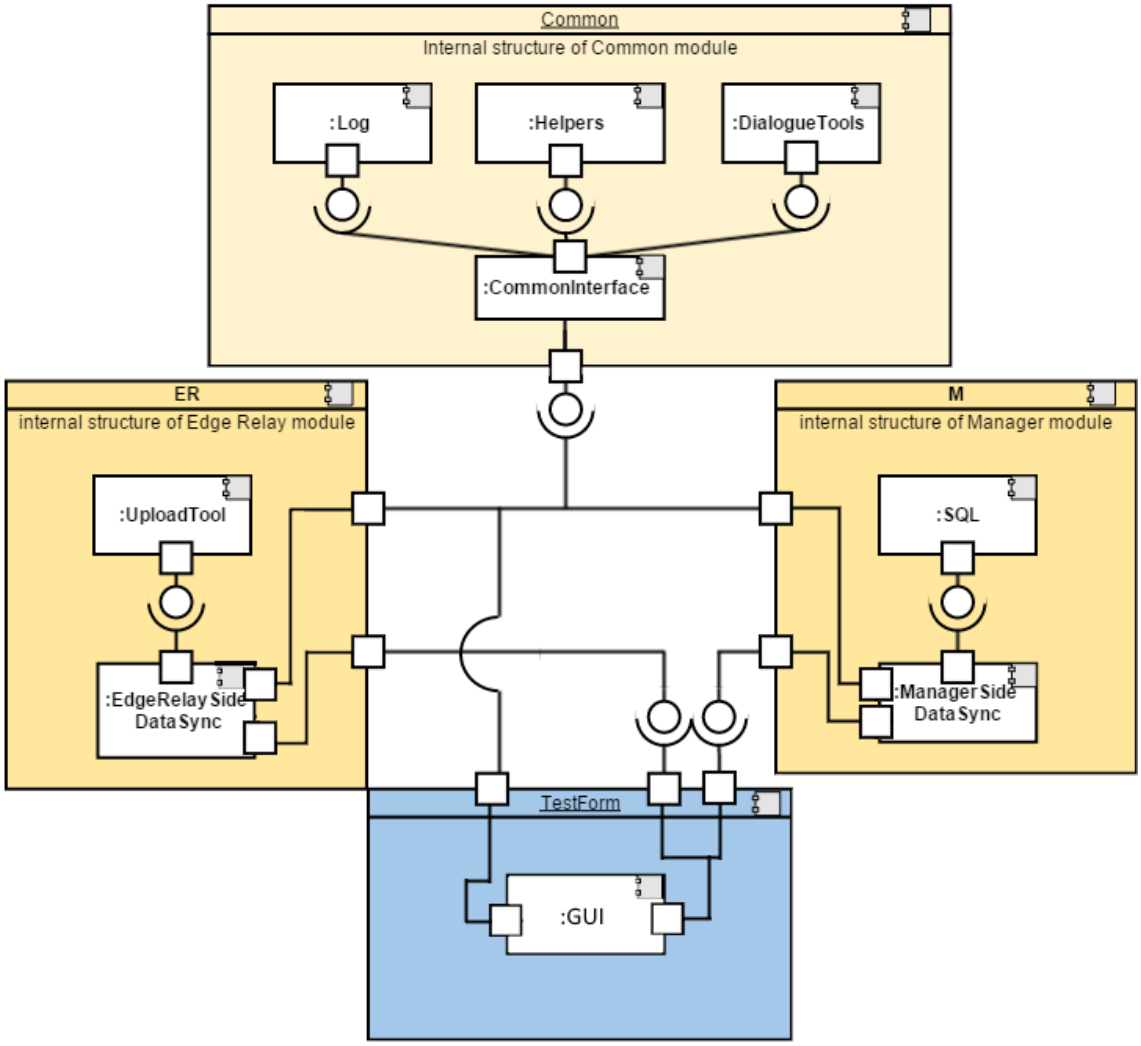

Figure 13: Inner components and modules of the proof of concept implementation.

Common module has three components, *Log*, *Helpers*, and *DialogueTools*. The *Log* component implements logging into external files on a hard drive. The *Helpers* component includes miscellaneous and generally useful functions. The *DialogueTools* component has all the needed functions to edit Dialogue files. Interfaces of these components are used by *CommonInterface* component, which wraps them up as a new interface. This is shown outside of the module. The *Common* module is used by all three other modules.

The *ER* module includes all functionality that is unique to the Edge Relay. *EdgeRelaySideDataSync* component has most of the functionalities of the module, excluding communication with Data Aggregator. That is separated into its own component called *UploadTool*.

The *M* module is designed in the same way. *ManagerSideDataSync* component handles most of the functionality, excluding communication with the mining equipment database. That functionality is separated into component *SQL*.

The *TestForm* module does not contain anything but user interface made with WPF. It uses interfaces of all three other modules.

4.5 Verification

Communication protocol used by DATAMiNe was verified using PROMELA and Spin. The actual code for meta model can be read from Appendix A. The verification was done using iSpin, which is graphical user interface for Spin made with Tcl/Tk. Figure 14 shows the edit view in iSpin.

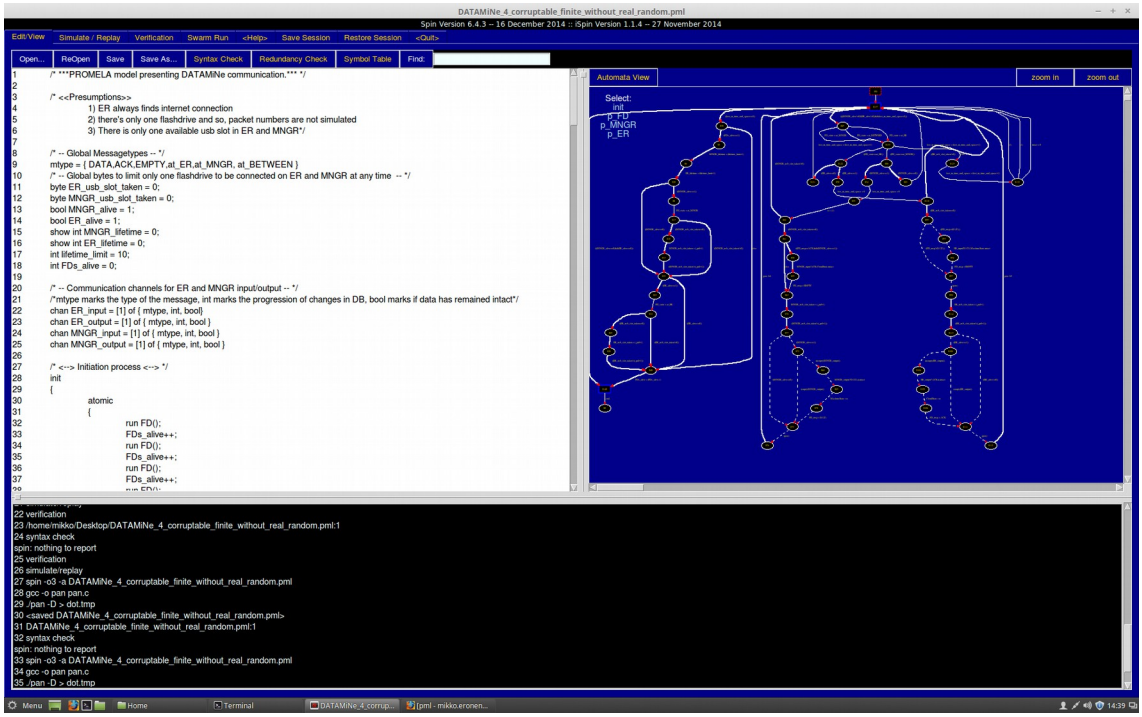

Figure 14: iSpin GUI

PROMELA was used to model the interaction between Manager and Edge Relay state machines. In addition to these two state machines, a third was made to simulate the medium between those two agents, i.e., a flash drive state machine. The flash drive state machine, however, is not the best description, because the state machine also simulates the travelling, corruption, and misplacement of the flash drive. This chapter will go through models of the verified state machines.

4.5.1 Overview of the meta model

Five different versions were made to test the PROMELA model from different aspects. The versions were more complicated with each step. Steps included following features.

1. Simulated Manager and Edge Relay state machines with one flash drive state machine moving between them. Running time infinite.
2. As Step one, but running time was limited to a finite length.
3. Multiple flash drives over infinite timespan. Also, adding possibility that flash drive can be lost forever.
4. As step three, but running time was limited to a finite length.
5. Step four with possibility of data corruption.

Verification covers reliability of protocol between Manager and Edge Relay. Actual Spin listings from verifications can be found from Appendix B. Verified aspects guarantee the following:

- The protocol does not end up in a non-progressing loop, i.e., live lock.
- Data Aggregator state always progresses forward, i.e., protocol does not end up in deadlock.
- Data within lost packets are written into the next ones until acknowledgement has been received from the Edge Relay.
- Corrupted packets are considered as lost.
- Synchronization eventually succeeds as long as there exists a transferring flash drive and the packets are not infinitely being lost.

This chapter explains the contents of the PROMELA meta model in the final version, i.e., step 5. Note also that the Appendix A metamodel is from the same step. Preceding version steps can be deduced as subsets from step 5 functionality. In addition to the state machines, there exist message channels, message types, and a set of global variables that map the progression of synchronization. In PROMELA, declarator *mtype* can define names of numeric constants. Often these constants are used to differentiate between interaction packet types or other enumerations. In DATAMiNe verification the following enumerations were defined.

- DATA
- ACK
- EMPTY
- at_ER
- at_MNGR
- at_BETWEEN

The first three are communication packet types. The rest are simulated locations where flash drive can exist in any given time during its lifespan. Deciding between these three locations is a core functionality of the flash drive state machine, which is the same as FD state machine. Figure 15 illustrates this behaviour. When *at_ER* or *at_MNGR* is picked as simulated location, a flash drive state machine will interact with Manager state machine, which is the same as *MNGR* or with Edge Relay state machine, which is the same as *ER* respectively.

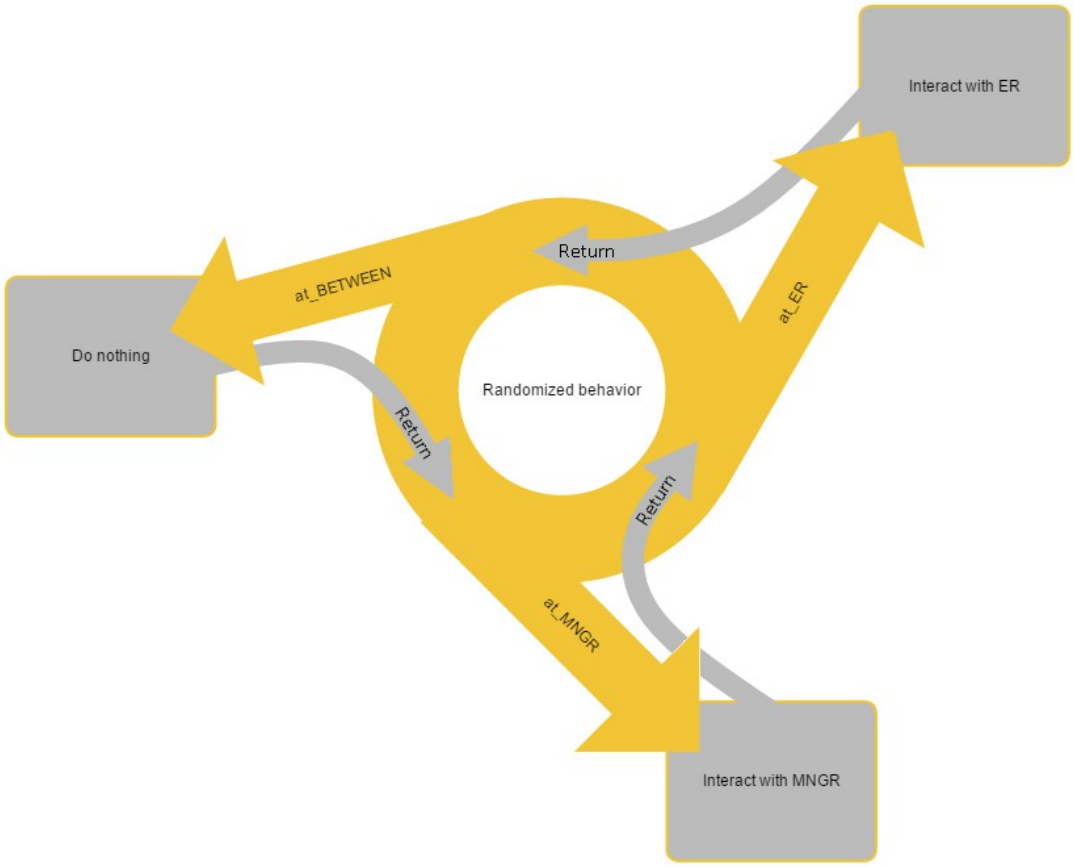

Figure 15: Randomized location of the simulated flash drive is a key functionality of FD state machine.

Manager and Edge Relay state machines have two designated communication channels each. The Manager has its input and output channels, respectively called *MNGR_input* and *MNGR_output*. Edge Relay similarly has its input and output channels respectively called E*R_input* and *ER_output*. Every channel packet is formed in the same way. First, a packet contains a packet type from *mtype*, i.e., *DATA* or *ACK.* Second, a packet contains an integer that mirrors the known state of the database in the endpoint from where the packet originates. This enables the endpoints to acknowledge each other's database state. Third, a packet contains a Boolean variable that simulates if the packet is corrupted during transfer. Normally, packet corruption would be noticed by comparing hash values generated at both endpoints, but in this PROMELA model a mere Boolean is a sufficient level of simulation.

Table 1 lists global variables, their initial values, and their purpose. The related initializations can be found from the code lines 11–18 in Appendix A. These variables control interaction, accessibility, and the shutdown of the PROMELA model.

Table 1: Global variables of PROMELA model.

Data type	Variable name	Initial value	Purpose
Byte	ER_usb_slot_taken	0	Guard variable to guarantee only one flash drive accessing Edge Relay
Byte	MNGR_usb_slot_taken	0	Guard variable to guarantee only one flash drive accessing Manager
Boolean	MNGR_alive	True	Variable to map Manager process existence. Needed in time finity.
Boolean	ER_alive	True	Variable to map Edge Relay process existence. Needed in time finity.
Integer	MNGR_lifetime	0	Variable to map how long Manager has existed. Needed in time finity.
Integer	ER_lifetime	0	Variable to map how long Edge Relay has existed. Needed in time finity.
Integer	lifetime_limit	100	Variable to limit Edge Relay and Manager existence. Needed in time finity.
Integer	FDs_alive	0	Variable that maps how many Flash Drives exists. Changes when drives are created or lost.

In PROMELA, all state machines are defined as processes when they are launched. The same state machine can be launched multiple times with different process identification number, i.e., *_pid*. Manager and Edge Relay state machines are modelled so that only one flash drive can be in interaction with each of them at any given time.

Variables *ER_usb_slot_taken* and *MNGR_usb_slot_taken* are used to mark which flash drive process occupies Manager or Edge Relay process. Marking is done with setting the variable value to respective *_pid.* When occupation ends, the value is reverted into its initial value 0.

Booleans *MNGR_alive* and *ER_alive* are marked *true* when Manager and Edge Relay state machine processes are launched. Variables are marked *false* when their respective process ends. Values of *MNGR_alive* and *ER_alive* are decided based on *MNGR_lifetime, ER_lifetime* and *lifetime_limit.* Each time when either, Manager or Edge Relay process, interacts with flash drive process, its respective *lifetime* is incremented. When *lifetime* goes over *lifetime_limit* the process in question shuts itself down and its *alive* -Boolean is marked *false*. These five variables configure the longevity of finite time with steps 2, 4 and 5. Finite lifetime is used to simulate finite operation time of mining equipment or its database.

FDs_alive is a number that counts running flash drive processes. The variable is incremented when processes are launched and decremented when shut down. Flash drive processes are shut down if flash drive is lost or the endpoints Manager and Edge Relay processes have reached the end of their lifespan. *FDs_alive* is a vital element to decide when finite time ends in the system where multiple flash drive processes exists.

In addition to the mentioned global variables, each launched process also has internal variables about their view on the database state on each endpoint. These variables are *MachineState* and *CloudState*. *MachineState* is increased when Manager process creates new *DATA* packet. *CloudState* is set to an equal value when the *DATA* packet reaches Edge Relay process uncorrupted. Following *ACK* packet enables Manager process to update its *CloudState.*

4.5.2 Manager state machine

Manager endpoint of the Spin verification generates *DATA* packets and consumes *ACK* packets. The actual code can be read from lines 214–259 in Appendix A. Table 2 presents the inner variables that are used to track the success of the synchronization. Figure 16 Illustrates the Manager PROMELA model drawn by iSpin.

Table 2: Inner variables of Manager state machine and their initial values.

Name	**Type**	**Initial Value**	**Purpose**
MachineState	integer	-1	Presents the amount of made changes in the simulated database.
CloudState	integer	-1	Presents the value of last arrived ACK packet.

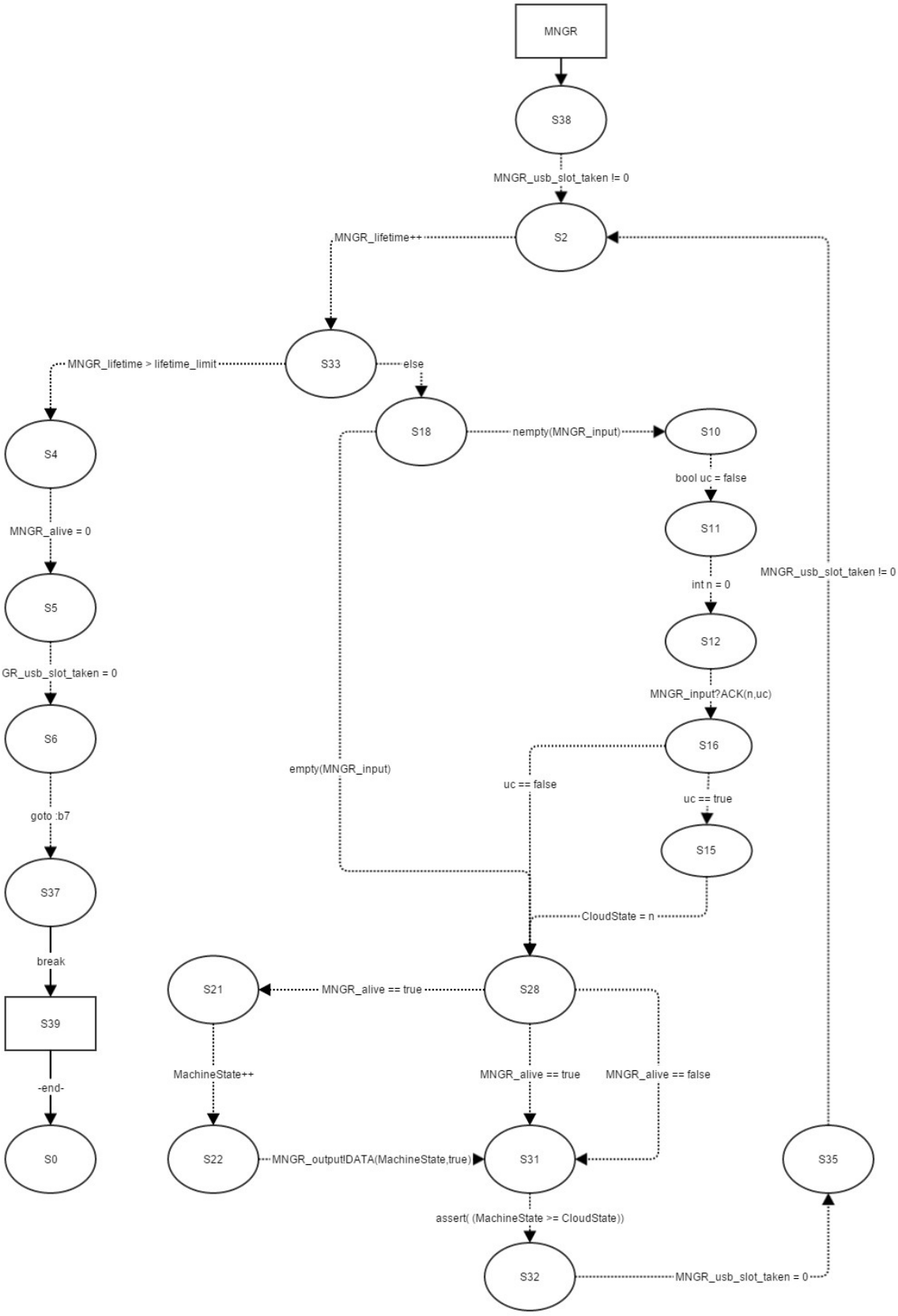

Figure 16: Manager as PROMELA defined state machine.

With using PROMELA's common functionality, the state machine starts its activity only when some flash drive occupies the Manager process, i.e., *MNGR_usb_slot_taken* is non-zero. First check is comparing *MNGR_lifetime* to *lifetime_limit*. If process is not in the end of its lifespan, input channel *MNGR_input* is processed instead in case of an *ACK* packet. Within input packet processing, the Boolean *uc* is used to decide if the packet is corrupted during propagation. Next, Manager goes through a loop which randomizes if new *DATA* packet into *MNGR_output* would be created. Probability for new packet to be created is 50%. Finally Manager releases the occupation by setting the *MNGR_usb_slot_taken* to zero. When process lifespan is over, the variable *MNGR_alive* is set to *false* and variable *MNGR_usb_slot_taken* is set to zero. After this, process is shut down.

4.5.3 Edge Relay state machine

Edge Relay endpoint of the Spin verification consumes *DATA* packets and generates *ACK* packets. Corresponding lines of code are 261–304, and can be found in Appendix A. Generally, the structure is very similar to Manager's state machine but it does not contain the random element in packet generation. Figure 17 illustrates the state machine logic. Following Table 3 presents the inner variables of Edge Relay state machine process.

Table 3: Inner variables of Edge Relay state machine and their initial values.

Name	Type	Initial Value	Purpose
MachineState	integer	-1	Presents the amount of simulated mining equipment database changes that are conveyed to the simulated Data Aggregator.
CloudState	integer	-1	Presents the amount of database changes in simulated Data Aggregator.

Edge Relay state machine activates when *MNGR_usb_slot_taken* is something other than zero. In the same way as in Manager state machine, this state machine also shuts down if its lifespan is over. Shutting down includes appropriate measures, i.e., setting the *ER_usb_slot_taken* to zero and setting the *ER_alive* to *false*. In functional branch where lifespan has not ended yet, the first action is to check the input channel *ER_input*. Only if *DATA* packet is found and it is uncorrupted, the process proceeds to update the *CloudState* variable. Otherwise, process gives up the control, setting the *ER_usb_slot_taken* to zero. After updating the *CloudState*, an *ACK* packet is created with the same *CloudState.* Afterwards, simulated USB slot is released by setting the *ER_usb_slot_taken* to zero.

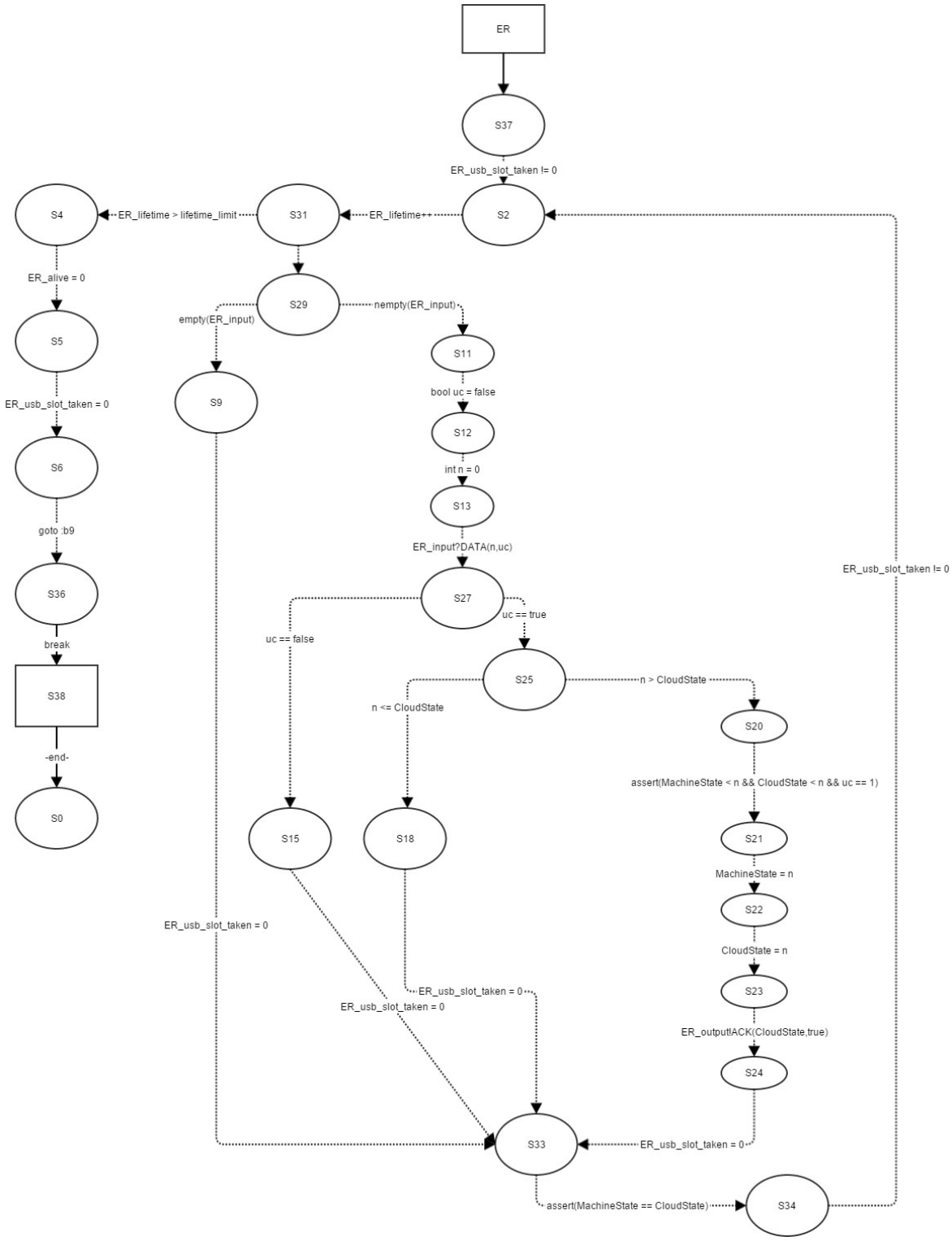

Figure 17: Edge relay as PROMELA defined state machine.

4.5.4 Flash drive state machine

This state machine simulates the challenging communication channel between Manager and Edge Relay. It is referred here as the flash drive state machine. Simulated situations include the following.

- Flash drive is transported to Manager endpoint, *at_MNGR*.
- Flash drive is transported to Edge Relay endpoint, *at_ER*.
- Flash drive is left somewhere else, *at_BETWEEN.*
- Flash drive is left somewhere else for too long and it is ultimately lost, *lost_in_time_and_space.*
- Flash drive data is corrupted so that the hash value will not match any more, Boolean *intact.*

Figure 18 illustrates how the state machine logic works. With each flash drive state machine process, there are inner variables and they are initialized as follows in Table 4.

Table 4: Inner variables of flash drive state machine and their initial values.

Name	**Type**	**Initial Value**	**Purpose**
FD_state	mtype	at_BETWEEN	Presents the simulated location of flash drive.
FD_msg	mtype	EMPTY	Presents the current transported packet type.
MachineState	integer	-1	Presents the Machine state of the sender
CloudState	integer	-1	Presents the CloudState state of the sender
lost_in_time_and_space	integer	0	Presents the amount of consecutive loops that the flash drive has been lost.

The main loop of the state machine activates when at least one endpoint is still alive and flash drive has not been lost too long i.e. five or more consecutive loops. First, a simulated place for the flash drive is randomly picked.

When *at_BETWEEN* is picked, the variable *lost_in_time_and_space* is incremented. Additionally, it is randomly decided if the data would be corrupted inside the simulated payload. Nothing happens if there is no payload packet. Probability of packet corruption is 25%. Probability for picking the *at_BETWEEN* in the start of the loop is 50% so the effective probability for packet corruption at the start is around 13%. Packet corruption is done by setting the variable *intact* to *false*. Probability for picking the *at_ER* or *at_MNGR* is 25%.

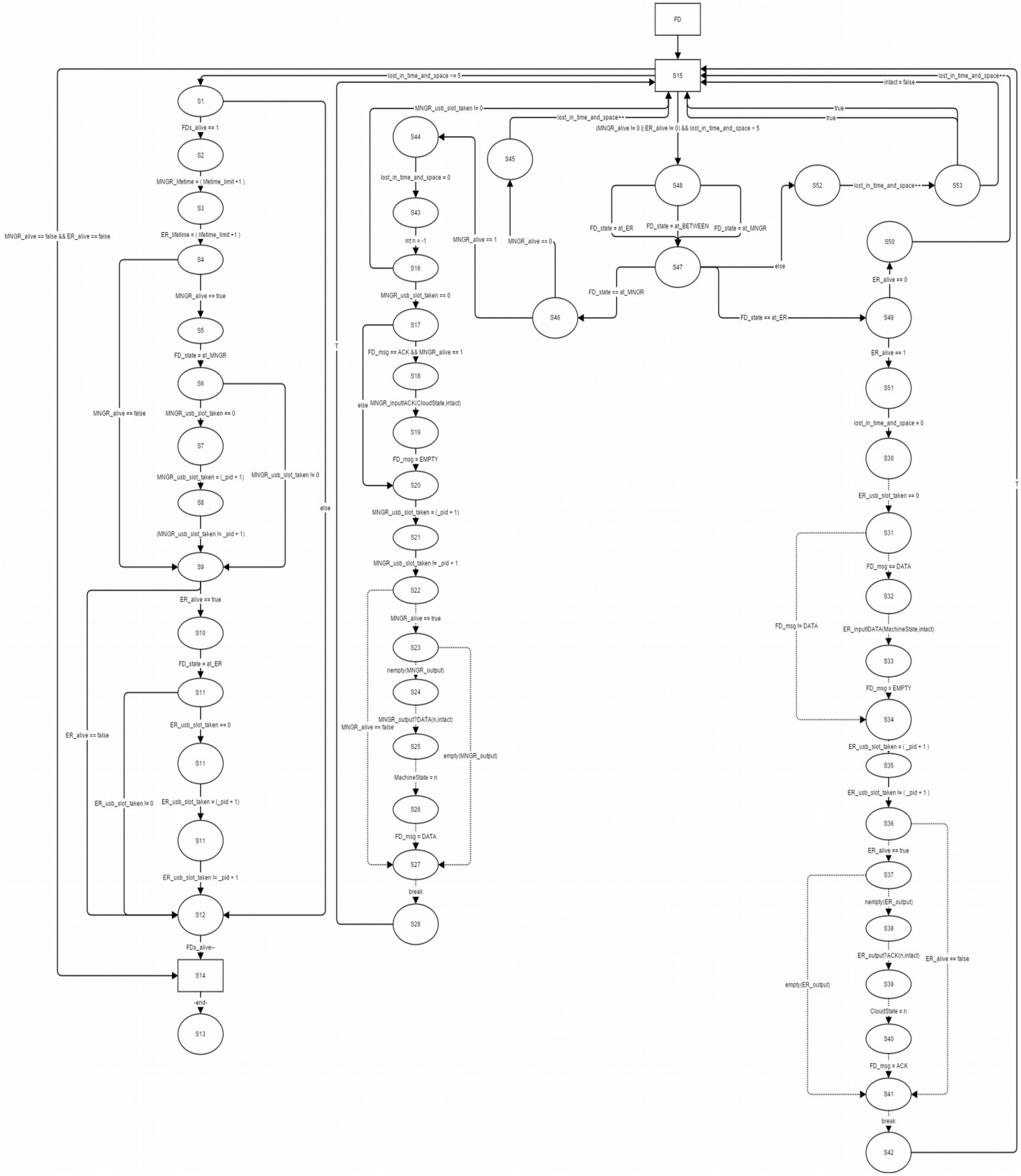

Figure 18: Flash drive state machine i.e. medium related events between Manager and Edge Relay.

If the *at_ER* is picked and if flash drive carries *DATA*, it is written into *ER_input.* After that, *ER_usb_slot_taken* is set to reflect process *_pid* and control is given elsewhere, i.e., Edge Relay process. When *ER_usb_slot_taken* is non-zero, Edge Relay state machine starts working (*ER_usb_slot_taken != 0* in Figure 17). In the same way the flash drive state machine has to wait until Edge Relay state machine returns *ER_usb_slot_taken* to zero (*ER_usb_slot_taken != (_pid + 1)* in Figure 18). Only after that is done, control is returned to the flash drive state machine process. That is, Edge Relay returns the control by writing the *ER_usb_slot_taken* back to zero. Once control is back, *ER_output* is read in case there is new *ACK* packet available. Afterwards, new iteration starts, and new location for flash drive is randomized. In the beginning of this loop, if E*R_alive is false*, the only action is to increment *lost_in_time_and_space*. This means that a flash drive state machine process is to be shut down if it does not does not have access to existing endpoint within certain amount of loops.

If *at_MNGR* is picked, the process would be much the same as with the *at_ER*. Difference is that first, a possible *ACK* packet is written into the *MNGR_input*. When control returns from the Manager process, the *MNGR_output* would be read in case there is a new *DATA* packet. In the beginning of this loop, if *MNGR_alive* is *false* the only action is to increment *lost_in_time_and_space.* End condition for the state machine is that *lost_in_time_and_space* reaches value five. If a flash drive process is last of its kind to be alive, it would shut down endpoints if not terminated before.

5 RESULTS AND ANALYSIS

DATAMiNe tries to solve a problem that is aligned with current trends, but simultaneously linked with restricted technical properties of mining environment. DATAMiNe concept consists of two agents and an interaction protocol implemented within them. The fundamental idea is to gather usage data from mining equipment into manufacturer's cloud Data Aggregator. Often this data must be gathered using flash drives.

This chapter combines core components and findings in this thesis. Main points are the design, the proof of concept, and the model verification of DATAMiNe.

5.1 General design of DATAMiNe

DATAMiNe synchronization packets contain usage data generated by mining equipment. Data inside those packets is kept concise and non-repetitive by choosing only the needed changes from mining equipment's database. Decisions are made by timestamping every line of the database. Data flows only upstream from equipment to the Data Aggregator, excluding the acknowledge messages. When using flash drives, connection information and other signalling between mining equipment and Data Aggregator are collected in separate file called Dialogue which is encrypted, and written in XML. Dialogue is transferred alongside the synchronization packets inside flash drives.

Data Aggregation database is an aggregation of all mining equipment databases. Data from different equipment instances is separated by an extra column in each table. That column contains equipment identification information. Transmission data that DATAMiNe generates is encrypted with a symmetrical method.

5.2 Proof of concept

Proof of concept of DATAMiNe contains two pieces of agent software, Manager and Edge Relay. Basic interaction protocol functionalities were established when doing the proof of concept implementation. Schemes like selection of new changes at mining equipment database, and connection signalling through Dialogue were included. The result was a demo where new changes could be fetched and written into a flash drive and afterwards that data could be written into another database according to the DATAMiNe interaction protocol. The proof of concept filled its purpose by showing the necessary steps to achieve a one-way synchronization scheme.

Not all the needed functionalities were present in the proof of concept. For example, integration with existing mining equipment software was not done. In the demonstration, Manager and Edge Relay were used through a test user interface, which helped to debug the interaction protocol. Also, information security features like encrypting and hash value comparison, complete support for all database tables and cloud support were not implemented.

The development of DATAMiNe benefited from the proof of concept implementation phase. The crucial advantage was refinement of used design decisions. Prospects that were deemed to be inconvenient, were substituted with alternatives. For example, the first vision of detecting new data on mining equipment database, was not to rely on timestamps. This was because it would also mean reliance to the internal clock of the mining equipment. However, it came evident that such solution would be overly complicated and the return of investment would be dubious. It is easier to keep the mining equipment clock running, than implement the necessary extra logic. Consequently, the timestamps were established. Also in the very begining, commercial solutions, such as MSSQL Change Data Capture, were inspected relating to the synchronization scheme. Very soon it became evident, that it was more cost-effective to make custom solution to use with DATAMiNe. Proof of concept implementation steered the DATAMiNe development towards cost-efficiency and practicality.

5.3 Verification results based on PROMELA model

The Spin tool was used to verify in a formal fashion that the DATAMiNe interaction protocol between Manager and Edge Relay does not produce deadlocks, or contain non-progress loops. Additionally, it was verified that Data Aggregator was updated only when new data arrived. Also, random cases of data corruption and flash drive misplacements were not able to break the system. As long as there would be flash drives moving between the endpoints, the synchronizable data would eventually reach Data Aggregator.

The main benefit of doing verification on the protocol was not necessarily the verification result itself, but the required level of detail which was laid by the nature of such formal presentation. In order to achieve formal verification, the PROMELA models were specified in high detail. That enforced a lot of decisions that would not have been made otherwise. One such detail is, how the protocol handles arrivals of obsolete packets.

PROMELA model must be implemented precisely to gain the benefits from the verification. The model had also many blind spots. It did not consider any information security issues or any actors outside of transmissions between Manager and Edge Relay.

5.4 Applicability

DATAMiNe is not a production-quality solution by any means. Still, the general design, proof of concept, and interaction protocol verification are usable when building a DATAMiNe solution into production environment.

Many features were not present in the proof of concept, or in the PROMELA model. In addition to information security and cloud implementation, sending data as SQL commands must be reconsidered. Plain SQL commands can produce security risk especially if symmetric encryption is compromised for any reason. Balakumar and Sakthidevi do not address this problem in their article [3] but a risk of injection is imminent.

5.5 Further development objectives

As shown in Figure 5, before production level implementation, synthesis from verification and a proof of concept must be created. The synthesis includes all the discoveries made in DATAMiNe development this far, arranged into some kind of lessons learned. This thesis' results are the starting point in that synthesis phase. Next, synthesis must be continued outside this document to initiate product level implementation of DATAMiNe. One topic that this thesis was unable to answer, was the selection of cloud service provider for Data Aggregator. Based on Section 2.5, the provider should probably be one of the big players. Provider should be one with history on the market and such that has plenty of resources to enforce their SLAs.

Before the production implementation, it could be useful to compare the architecture of DATAMiNe with the Industrial Internet Consortium's reference architecture [1] for best practices. This would potentially help tuning the system to be easier to maintain. The reference architecture was not considered during the thesis work, since the reference architecture document was not available during that time.

There are also visions of what additional features could be developed, after the current set of functionalities are finished. Currently DATAMiNe supports traffic to be synchronized only upstream, i.e., from mining equipment to the Data Aggregator. One possible way to develop DATAMiNe further, would be enabling downstream synchronization as well. This would enable new ways to rapidly deploy new configurations. However, the downstream path would raise some new questions. If more nodes in a synchronization network are able to make changes, the scheme gets more complicated. Availability and data consistency, i.e., optimistic versus pessimistic synchronization strategies should be reconsidered [8] [9].

DATAMiNe could be more versatile in the future. Filtering and selecting of relevant data can be implemented as a configuration. Data that would be transmitted, would not have to be database related either. For example, documentation files could be transferred during other transmissions. Also tombstones, like in Cohen's article [4], could become relevant if object deletion should be enabled in multiple nodes.

Further into the future, when DATAMiNe is taken into production, the next logical step would be the development of a new piece of software, which mines the data in Data Aggregator. Automatic algorithms could reveal correlation trends between mining machine use data patterns and maintenance needs. This program would be used directly by the mining machine manufacturer, and it could be used to improve the services the manufacturer offers.

6 CONCLUSIONS

The combination of the DATAMiNe design and verified interaction protocol can be used to create added value to the mining equipment manufacturer's business. In the near future, DATAMiNe can be integrated with the rest of the mining equipment software. The design has been prepared so far that further studying apart from real integration can be disadvantageous.

Designing an IoT data aggregation system that spreads over the perimeter of the Internet requires special attention. In DATAMiNe, the network is assisted by human workers who transport flash drives in and out from the mine. This is done to connect the mining equipment loosely to the data aggregation network.

The system consists of three pieces of agent software, Manager, Edge Relay and Data Aggregator. DATAMiNe's use case culminates around Manager and Edge Relay because those two make the data communication over the assisted mine network. The protocol used between these two endpoints has to take human error into consideration in the form of connection incoherency and information security. Data Aggregator is the component that gathers and maintains the data collection which is aggregated. It is planned to be hosted as a cloud service.

The design of DATAMiNe was prepared for production implementation by making a proof of concept implementation software, and by verifying the protocol that operates over the assisted mine data network. Both ways offered different but complementary discoveries that improved the original design.

The proof of concept implementation guided the design to be more cost-effective and the roles and boundaries between Manager and Edge Relay agents grew clearer. Verification of the protocol forced the designer to take critical details into consideration.

Compared to the ordinary software development, prototyping and verifying served DATAMiNe better than continuous integration. Rapid deployment is out of the question in production environment and strict waterfall process would require too much bureaucracy. The way DATAMiNe has been developed this far, is compromise between those two. As an anecdotal rule of thumb, it is worthwhile to write one version of the program and learn from it. Afterwards, a better version can be written using the lessons learned. The first version gives the composer a big picture of the system, the lack of which is the reason the first version is inadequate in the first place.

Data Aggregator was not included into the scope of proof of concept implementation or the protocol verification. The cloud service provider is yet to be decided but the provider should be one of the bigger players on the market that have at least few years of experience and solid monetary basis to compensate if quality of service does not meet the requirements of the SLA.

REFERENCES

[1] Industrial Internet Reference Architecture, 1, Industrial Internet Consortium, http://www.iiconsortium.org/, 2015, available: http://www.iiconsortium.org/IIRA.htm.

[2] P. Albright, Getting in sync with synchronization, Wireless Week, Vol. 8, No. 28, 2002, pp. 28.

[3] V. Balakumar, I. Sakthidevi, An efficient database synchronization algorithm for mobile devices based on secured message digest, 2012 International Conference on Computing, Electronics and Electrical Technologies, ICCEET 2012, 2012, pp.937-942, 2012, pp. 937-942.

[4] N.H. Cohen, A Java Framework for Mobile Data Synchronization, Cooperative Information Systems, 2000.

[5] M. Coleman, Data synchronization: What to look for, Telemarketing, Vol. 14, No. 3, 1995, pp. 120.

[6] D.E. Comer, Computer Networks and Internets: International Edition, Fifth Edition ed. Pearson Education, Upper Saddle River, New Jersey, 2009, 599 p.

[7] J. Cucurull, Surviving Attacks in Challenged Networks, IEEE Transactions on Dependable and Secure Computing, Vol. 9, No. 6, 2012, pp. 917.

[8] S.B. Davidson, Consistency in Partitioned Networks, ACM Computing Surveys, Vol. 17, No. 3, 1985, pp. 341-370.

[9] M.J. Fischer, Sacrificing serializability to attain high availability of data in an unreliable network, 1982, pp. 11.

[10] A. Freyberg, A. Hornigold, A. Willmott, Cloud provider valuations more an art than a science, Global Telecoms Business, 2011.

[11] I. Haikala, J. Märijärvi, 2.3. Laatu, laatujärjestelmä ja laadunvarmistus, in: Ohjelmistotuotanto, 11 ed., Gummerus Kirjapaino Oy, Jyväskylä, 2006, pp. 48-51.

[12] C. Hargrave O., J.C. Ralston, D. Hainsworth W., Optimising wireless LAN for longwall coal mine automation, Conference Record - IAS Annual Meeting (IEEE Industry Applications Society), 2005, Vol.1, pp.218-224, Vol. 1, 2005, pp. 218-IAS.

[13] V. Ho, No clear leader among cloud vendors, Business And Economics, United Kingdom, Singapore, 2014.

[14] G. Holzmann, Spin Model Checker, the: Primer and Reference Manual, First ed. Addison-Wesley Professional, 2003.

[15] G. Holzmann, Select - non-deterministic value selection. spinroot.com, web page. Available (referenced 11.05.2015): http://spinroot.com/spin/Man/select.html.

[16] M.P. Huget, Interaction protocol engineering, Communication In Multiagent Systems, Vol. 2650, 2003, pp. 179-193.

[17] L. JianJun, Z. XiangYun, Research for a Data Synchronization Model Based on Middleware and Rule Base, Information Science and Engineering, Dec.2009, pp.2998-3001, 2009, pp. 2998-3001.

[18] P. Kee-Hyun, Efficient Transmission Method for Mobile Data Synchronization Based on Data Characteristics, Proceedings of the International Conference on IT Convergence and Security 2011, 2012, .

[19] J. McKendrick, Cloud Computing Market May Become An Oligopoly of High-Volume Vendors, Cloud Computing Market May Become An Oligopoly of HighVolume Vendors - Forbes, Vol. 2015, No. Forbes, 2013, pp. 13/6/2015. available (referenced 13/6/2015): http://www.forbes.com/sites/joemckendrick/2013/07/11/cloud-computing-market-may-become-an-oligopoly-of-high-volume-vendors/.

[20] P. Ocenasek, On the Secure and Safe Data Synchronization, Human Aspects of Information Security, Privacy, and Trust, 2013, .

[21] Open Mobile Alliance, About Open Mobile Alliance, Open Mobile Alliance, web page. available (referenced 09.06.2015): http://openmobilealliance.org/about-oma/.

[22] B. Ostergaard, How to select the best hybrid cloud provider for your firm, Computer Weekly, 2014, pp. 17-19.

[23] W. Pauley, Cloud Provider Transparency: An Empirical Evaluation, IEEE Security & Privacy Magazine, Vol. 8, No. 6, 2010, pp. 32.

[24] C. Preimesberger, 10 Big Data Trends Apt to Influence Enterprises in 2015, eWeek, 2015, pp. 1-1.

[25] W. Royce, Managing the Development of Large Software Systems, Proceedings of IEEE WESCON 26, August, The Institute of Electrical and Electronics Engineers, Inc., WESCON 26, pp. 328.

[26] J. Spragins D., J.L. Hammond, Pawlikowski Krzysztof, Telecommunications: Protocols and Design, in: Addison-Wesley, New York, NY, USA, 1991, pp. 18-24.

[27] S. Tarkoma, Data Synchronization, Mobile Middleware, Chapter 8, p.225-239, 2009, pp. 225-239.

[28] C. Zhu, X. Deng, J. Zhu, L. Li, X. Zeng, H. Yu, S. Zhang, Performance analysis of wireless local area networks (WLAN) in a coal-mine tunnel environment, Mining Science and Technology, Vol. 20, No. 4, 2010, pp. 629-634.

APPENDIX A : PROMELA MODEL

```
001 /* ***PROMELA model presenting DATAMiNe communication.*** */
002
003 /* <<Presumptions>>
004   1) ER always finds internet connection
005   2) there's only one flashdrive and so, packet numbers are not simulated
006   3) There is only one available usb slot in ER and MNGR*/
007
008 /* -- Global Messagetypes -- */
009 mtype = { DATA,ACK,EMPTY,at_ER,at_MNGR, at_BETWEEN }
010 /* -- Global bytes to limit only one flashdrive to be connected on ER and
MNGR at any time  -- */
011 byte ER_usb_slot_taken = 0;
012 byte MNGR_usb_slot_taken = 0;
013 bool MNGR_alive = 1;
014 bool ER_alive = 1;
015 show int MNGR_lifetime = 0;
016 show int ER_lifetime = 0;
017 int lifetime_limit = 100;
018 int FDs_alive = 0;
019
020 /* -- Communication channels for ER and MNGR input/output -- */
021 /*mtype marks the type of the message, int marks the progression of
changes in DB, bool marks if data has remained intact*/
022 chan ER_input = [1] of { mtype, int, bool}
023 chan ER_output = [1] of { mtype, int, bool }
024 chan MNGR_input = [1] of { mtype, int, bool }
025 chan MNGR_output = [1] of { mtype, int, bool }
026
027 /* <--> Initiation process <--> */
028 init
029 {
030   atomic
031   {
032          run FD();
033          FDs_alive++;
034          run FD();
035          FDs_alive++;
036          run FD();
037          FDs_alive++;
038          run FD();
039          FDs_alive++;
040          run MNGR();
041          run ER()
042   }
043 }
044 /* -||- Process for Flashdrive statemachine -||- */
045 proctype FD()
046 {
047   /* -- States for flashdrive -- */
048   show mtype FD_state = at_BETWEEN;
049   show mtype FD_msg = EMPTY;
050   show bool intact = true;
051   show int MachineState = -1 ;
052   show int CloudState = -1 ;
053   int lost_in_time_and_space = 0;
054   /* begin finite loop that depends on lost_in_time_and_space and ER/MNGR
lifetimes */
055 end_FD:  do
056   :: (MNGR_alive == false && ER_alive == false) ->
057          break;
058   /*if flash drive has been lost extended period of time*/
059   :: ( lost_in_time_and_space >= 5) ->
060          if
```

```
061         :: (FDs_alive == 1) ->
062               MNGR_lifetime = ( lifetime_limit +1 );
063               ER_lifetime = ( lifetime_limit +1 );
064               if
065               :: (MNGR_alive == true) ->
066                     FD_state = at_MNGR;
067                     if
068                     :: (MNGR_usb_slot_taken == 0) ->
069                           MNGR_usb_slot_taken = (_pid + 1);
070                           do
071                           :: (MNGR_usb_slot_taken != _pid + 1) ->
072                                 break;
073                           od
074                     :: ( MNGR_usb_slot_taken != 0 ) ->
075                     fi;
076               :: (MNGR_alive == false) ->
077               fi
078               if
079               :: (ER_alive == true) ->
080                     FD_state = at_ER;
081                     if
082                     :: (ER_usb_slot_taken == 0) ->
083                           ER_usb_slot_taken = (_pid + 1);
084                           do
085                           :: (ER_usb_slot_taken != _pid + 1)      ->
086                                 break;
087                           od
088                     :: ( ER_usb_slot_taken != 0 ) ->
089                     fi
090               :: (ER_alive == false) ->
091               fi
092         :: else ->
093         fi
094         FDs_alive--;
095         break;
096   :: ( (MNGR_alive != 0 || ER_alive != 0) && lost_in_time_and_space < 5 )
->
097         /*choose FD_state randomly*/
098         do
099         :: FD_state = at_BETWEEN ->
100               break;
101         :: FD_state = at_ER ->
102               break;
103         :: FD_state = at_MNGR ->
104               break;
105         od
106         /* *** Act according to FD_state *** */
107         if
108         /*If stick is in Manager slot*/
109         /*Read all MNGR_output and if it is not empty change FD_msg*/
110         :: ( FD_state == at_MNGR) ->
111               if
112               :: ( MNGR_alive == 1 ) ->
113                     lost_in_time_and_space = 0; /*reset lost counter*/
114                     int n = -1
115                     if
116                     :: (MNGR_usb_slot_taken == 0) ->
117                           /* check that usb slot is not occupied*/
118
119                           if
120                           :: ( FD_msg == ACK && MNGR_alive == 1 ) ->
121                                 MNGR_input!ACK(CloudState,intact);
122                                 FD_msg = EMPTY;
123                                 /* <<>> */
124                           :: else ->
125
126                           fi;
127                           MNGR_usb_slot_taken = (_pid + 1); /* occupy
usb slot and give turn to MNGR */
128                           do /*wait for the release*/
129                           :: (MNGR_usb_slot_taken != _pid + 1)    ->
130                                 atomic
131                                 {
132                                 if
```

```
 133                                     :: (MNGR_alive == false) ->
 134                                     :: (MNGR_alive == true) ->
 135                                           if
 136                                           :: ( nempty(MNGR_output) ) ->
 137                                                 MNGR_output?
DATA(n,intact);
 138                                                 MachineState = n;
 139                                                 FD_msg = DATA;
 140                                           :: ( empty(MNGR_output))->
 141                                           fi
 142                                     fi
 143                                     break
 144                                     }
 145                               od
 146                         :: ( MNGR_usb_slot_taken != 0 ) ->
 147
 148                         fi
 149                   :: (MNGR_alive == 0 )->
 150                         lost_in_time_and_space++; /*increment lost
counter*/
 151                   fi
 152            /*If stick is in Edge Relay slot*/
 153            /*Write into ER_input if FD_msg is DATA. If so, make ACK
message*/
 154            :: ( FD_state == at_ER) ->
 155                   if
 156                   :: ( ER_alive == 1 ) ->
 157                         lost_in_time_and_space = 0;/*reset lost counter*/
 158                         if
 159                         :: ( ER_usb_slot_taken == 0 ) -> /* check that usb
slot is not occupied*/
 160                               atomic
 161                               {
 162                                     if
 163                                     :: ( FD_msg == DATA ) ->
 164                                           ER_input!
DATA(MachineState,intact);
 165                                           FD_msg = EMPTY;
 166                                     :: ( FD_msg != DATA ) ->
 167
 168                                     fi
 169                                     ER_usb_slot_taken = ( _pid + 1 ); /*
occupy usb slot and give turn to MNGR */
 170                               }
 171                               do /*wait for the release*/
 172                               :: ( ER_usb_slot_taken != ( _pid + 1 ) ) ->
 173                                     atomic
 174                                     {
 175                                           if
 176                                           ::(ER_alive == false)->
 177                                           ::(ER_alive == true)->
 178                                                 if
 179                                                 :: ( nempty(ER_output) )
->
 180                                                       ER_output?
ACK(n,intact)
 181                                                       CloudState = n
 182                                                       FD_msg = ACK
 183                                                 :: ( empty(ER_output) )
->
 184
 185                                                 fi
 186                                           fi
 187                                           break
 188                                     }
 189                               od
 190                         :: ( ER_usb_slot_taken != 0 ) ->
 191
 192                         fi
 193                   :: ( ER_alive == 0 ) ->
 194                         lost_in_time_and_space++; /*increment lost
counter*/
 195                   fi
 196            /*If stick is in In between*/
```

```
197         /*Do nothing*/
198         :: else ->
199                lost_in_time_and_space++;
200                do
201                :: 1 ->
202                       break;
203                :: 1 ->
204                       break;
205                :: intact = false ->
206                       /*1/4 propability that packet corrupts if
BETWEEN,that is 1/8 propability (~13%) all together*/
207                       break;
208
209                od
210         fi;
211  od;
212 }
213 /* -||- Process for Manager statemachine -||- */
214 proctype MNGR()
215 {
216  int MachineState = -1;
217  int CloudState = -1;
218  /* begin loop */
219  atomic
220  {
221         do
222         :: (MNGR_usb_slot_taken != 0) ->
223                MNGR_lifetime++
224                if
225                :: ( MNGR_lifetime > lifetime_limit ) ->
226                       MNGR_alive = 0
227                       MNGR_usb_slot_taken = 0
228                       break
229                :: else ->
230                       if
231                       :: (empty(MNGR_input)) ->
232                       :: (nempty(MNGR_input)) ->
233                              bool uc = false
234                              int n = 0
235                              MNGR_input?ACK(n,uc)
236                              if
237                              :: (uc == false) ->
238                              :: (uc == true) ->
239                                     CloudState = n
240
241                              fi
242                       fi
243                       do
244                       :: (MNGR_alive == true) ->
245                              MachineState++
246                              MNGR_output!DATA(MachineState,true)
247                              break
248                       :: (MNGR_alive == true) ->
249                              break
250                       :: (MNGR_alive == false) ->
251                              break
252                       od
253                       assert( (MachineState >= CloudState))

254                       MNGR_usb_slot_taken = 0
255                fi
256
257         od
258  }
259 }
260 /* -||- Process for Edge Relay statemachine -||- */
261 proctype ER()
262 {
263  show int MachineState = -1;
264  show int CloudState = -1;
265  /* begin loop */
266  atomic
267  {
268         do
```

```
269         :: (ER_usb_slot_taken != 0) ->
270                 ER_lifetime++
271                 if
272                 :: ( ER_lifetime > lifetime_limit ) ->
273                         ER_alive = 0
274                         ER_usb_slot_taken = 0
275                         break
276                 :: else ->
277                         if
278                         :: (empty(ER_input)) ->
279                                 ER_usb_slot_taken = 0
280                         :: (nempty(ER_input)) ->
281                                 bool uc = false
282                                 int n = 0
283                                 ER_input?DATA(n,uc)
284                                 if
285                                 :: (uc == false) ->
286                                         ER_usb_slot_taken = 0
287                                 :: (uc == true) ->
288                                         if
289                                         :: ( n <= CloudState)->
290                                                 ER_usb_slot_taken = 0
291                                         :: (n > CloudState)->
292                                                 assert(MachineState < n &&
CloudState < n && uc == 1)
293                                                 MachineState = n
294                                                 CloudState = n
295                                                 ER_output!ACK(CloudState,true)
296                                                 ER_usb_slot_taken = 0
297                                         fi
298                                 fi
299                         fi
300                 fi
301                 assert(MachineState == CloudState)
302         od
303   }
304 }
```

APPENDIX B : SPIN COMMAND PROMPT OUTPUT

```
001 spin -a  DATAMiNe_4_corruptable_finite_without_real_random.pml
002 gcc -DMEMLIM=1024 -O2 -DXUSAFE -DSAFETY -DNOCLAIM -w -o pan pan.c
003 ./pan -m10000  -c1
004 Pid: 2490
005 Depth=    7170 States=    1e+06 Transitions= 1.38e+06 Memory=   270.398
      t=     0.77 R=   1e+06
006 Depth=    7171 States=    2e+06 Transitions= 2.77e+06 Memory=   476.160
      t=     1.56 R=   1e+06
007 Depth=    7171 States=    3e+06 Transitions= 4.15e+06 Memory=   682.019
      t=     2.37 R=   1e+06
008 Depth=    7171 States=    4e+06 Transitions= 5.54e+06 Memory=   887.879
      t=      3.2 R=   1e+06
009 pan: reached -DMEMLIM bound
010   1.07365e+09 bytes used
011   102400 bytes more needed
012   1.07374e+09 bytes limit
013 hint: to reduce memory, recompile with
014    -DCOLLAPSE # good, fast compression, or
015    -DMA=228   # better/slower compression, or
016    -DHC # hash-compaction, approximation
017    -DBITSTATE # supertrace, approximation
018
019 (Spin Version 6.4.3 -- 16 December 2014)
020 Warning: Search not completed
021   + Partial Order Reduction
022
023 Full statespace search for:
024   never claim               - (not selected)
025   assertion violations      +
026   cycle checks        - (disabled by -DSAFETY)
027   invalid end states  +
028
029 State-vector 228 byte, depth reached 7171, errors: 0
030    4661219 states, stored
031    1788487 states, matched
032    6449706 transitions (= stored+matched)
033     303986 atomic steps
034 hash conflicts:    155749 (resolved)
035
036 Stats on memory usage (in Megabytes):
037   1066.868      equivalent memory usage for states (stored*(State-vector +
overhead))
038    959.827      actual memory usage for states (compression: 89.97%)
039                 state-vector as stored = 204 byte + 12 byte overhead
040     64.000      memory used for hash table (-w24)
041      0.343      memory used for DFS stack (-m10000)
042   1023.914      total actual memory usage
043
044
045
046 pan: elapsed time 3.76 seconds
047 No errors found -- did you verify all claims?
048 spin -a  DATAMiNe_4_corruptable_finite_without_real_random.pml
049 gcc -DMEMLIM=1024 -O2 -DXUSAFE -DNP -DNOCLAIM -w -o pan pan.c
050 ./pan -m10000  -l -c1
051 Pid: 2524
052 error: max search depth too small
053 Depth=    9999 States=    1e+06 Transitions= 1.88e+06 Memory=   169.910
      t=     0.85 R=   1e+06
054 Depth=    9999 States=    2e+06 Transitions= 3.76e+06 Memory=   274.793
```

```
      t=     1.71 R=   1e+06
 055 Depth=    9999 States=    3e+06 Transitions= 5.65e+06 Memory=   379.675
      t=     2.58 R=   1e+06
 056 Depth=    9999 States=    4e+06 Transitions= 7.53e+06 Memory=   484.461
      t=     3.45 R=   1e+06
 057 Depth=    9999 States=    5e+06 Transitions= 9.41e+06 Memory=   589.441
      t=     4.33 R=   1e+06
 058 Depth=    9999 States=    6e+06 Transitions= 1.13e+07 Memory=   694.324
      t=     5.23 R=   1e+06
 059 Depth=    9999 States=    7e+06 Transitions= 1.32e+07 Memory=   799.207
      t=     6.12 R=   1e+06
 060 Depth=    9999 States=    8e+06 Transitions= 1.51e+07 Memory=   903.992
      t=      7.1 R=   1e+06
 061 Depth=    9999 States=    9e+06 Transitions= 1.69e+07 Memory=  1008.875
      t=     8.06 R=   1e+06
 062 pan: reached -DMEMLIM bound
 063   1.07365e+09 bytes used
 064   102400 bytes more needed
 065   1.07374e+09 bytes limit
 066 hint: to reduce memory, recompile with
 067    -DCOLLAPSE # good, fast compression, or
 068    -DMA=232   # better/slower compression, or
 069    -DHC # hash-compaction, approximation
 070    -DBITSTATE # supertrace, approximation
 071
 072 (Spin Version 6.4.3 -- 16 December 2014)
 073 Warning: Search not completed
 074   + Partial Order Reduction
 075
 076 Full statespace search for:
 077   never claim               + (:np_:)
 078   assertion violations      + (if within scope of claim)
 079   non-progress cycles       + (fairness disabled)
 080   invalid end states  - (disabled by never claim)
 081
 082 State-vector 232 byte, depth reached 9999, errors: 0
 083    4574154 states, stored (9.14404e+06 visited)
 084    8070683 states, matched
 085   17214724 transitions (= visited+matched)
 086     611111 atomic steps
 087 hash conflicts:    799747 (resolved)
 088
 089 Stats on memory usage (in Megabytes):
 090   1081.839      equivalent memory usage for states (stored*(State-vector +
overhead))
 091    960.602      actual memory usage for states (compression: 88.79%)
 092                 state-vector as stored = 204 byte + 16 byte overhead
 093     64.000      memory used for hash table (-w24)
 094      0.343      memory used for DFS stack (-m10000)
 095      1.032      memory lost to fragmentation
 096   1023.914      total actual memory usage
 097
 098
 099
 100 pan: elapsed time 8.19 seconds
 101 No errors found -- did you verify all claims?
 102
 103
```