University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: **Computer Science**	Spring semester, 2014 Open access
Writer: **SURYANTO ANG**	…………SURYANTO ANG………… (Writer's signature)
Faculty supervisor: **TOMASZ WIKTOR WLODARCZYK** **CHUNMING RONG**	
Thesis title: **Framework of Evidence Collection with Temporal Logic and First-Order Logic for Providing Accountability in Cloud Service**	
Credits (ECTS): **30**	
Key words: **Cloud Service, Accountability, Evidence, Temporal Logic, First-Order Logic, Policy, A-PPL**	Pages: ……98……… + enclosure: **program on CD** Stavanger, **June 2014**

Framework of Evidence Collection with Temporal Logic and First-Order Logic for Providing Accountability in Cloud Service

Suryanto Ang

Faculty of Science and Technology

Department of Electrical and Computer Engineering

University of Stavanger

June 2014

Abstract

With the introduction of cloud computing, many types of services have been introduced within the umbrella of this technology. With these services, some of the computations are brought into network, cloud of service machines. Although the technology gives lots of benefits and flexibility to its users, there are some areas that need to be taken to concern. The lack of mechanism to verify the policies are in place in the real system is one of the main reasons for difficulty of cloud computing adoption. There is a need of way to control the processes within the service chain and make sure that they are complied with service level agreement.

A framework for collecting evidence based on source of information about system's running is proposed. The process is based on the obligations or policies defined for services in the system. This framework of evidence collection can be used as basis for providing accountability in cloud.

Accountability policies are, in the context of this framework, expressed in A-PPL which is an accountability policy representation framework. A use case is selected to demonstrate how the approach for evidence collection works. It is about health care service in the cloud. Test environment to represent healthcare service in cloud is set up. The purpose of this is to have data as source of evidence to be processed using proposed method on selected use case and related defined policies or obligations. The environment is set up using VMs (Virtual Machine) in Linux.

Two approaches on processing source of evidence and policy are shown and compared. The first approach processes them as MFOTL using MonPoly. The second approach processes them as Prolog (FOL) using Pyke.

Testing on those two approaches using the implementations done on this thesis shows that representing accountability policies in MFOTL gives more expressiveness than representing them in pure Prolog (FOL). However processing of MFOTL used in MonPoly gives no more flexibility in terms of practical usage and improvement than using Prolog with Pyke.

Keywords: Cloud Service, Accountability, Evidence, Temporal Logic, First-Order Logic, Policy, A-PPL

Acknowledgements

I would like to thank my supervisors which during my work in the thesis have given me valuable inputs, advices, and feedbacks. These really help me to not only make this work completed but also solve the problem in the right way.

My thanks also go to all my lecturers in University of Stavanger who have given me solid knowledge which is useful to support the work I am completing in the thesis.

I would also extend my deepest gratitude to my family, my parents, my brothers and sisters who always give me endless supports at whatever forms while completing the work.

Last but not least, I would also like to thank my friends for their accompaniments during days of completing my work in this thesis.

Preface

This thesis is submitted as partial fulfilment of the requirements to complete the Master of Science (M.Sc.) degree in Computer Science at the Department of Electrical and Computer Engineering at the University of Stavanger, Norway.

The work has been completed under supervision of Dr. Tomasz Wiktor Wlodarczyk. It covers work from February to June 2014. This thesis has been made solely by the authors with reference to researches of others. All researches discussed in the thesis have been properly referred in reference section.

The work is about providing method or way to be able to collect evidence on violation to obligations cloud service based on information that are collected from activities of every component in the scenario. The output of the process is evidence as a proof of something has violated the rules. This then can be basis for auditing process by a trusted pointed cloud auditor. The method is evidence-based i.e. the proposed framework has goal to collect evidence based on source of information about the system's running with respect to obligations or policies defined for services in the system. The framework of evidence collection can be used as basis for providing accountability in cloud.

Contents

List of Figures

List of Tables

1. Introduction

Providing accountability in cloud service is important as one of the ways to tackle obstacles for cloud computing adoption. Accountability solves problem by providing account of an entity's actions in the cloud with respect to obligations that have been set up for the cloud service. This gives every entity involved in the cloud service a way to measure if all entities are following all obligations related to them. Evidence is then necessary to show the obligations are followed i.e. supporting the accountability to be achieved. This section provides motivation behind solving the problem. Contributions of work in this thesis are listed. In addition to that, several works related to digital evidence and providing accountability in cloud with respect to obligations are discussed in short. At the last section, organization of this thesis is given.

1.1. Motivation

With the introduction of cloud computing, many types of services have been introduced within the umbrella of this technology. With these services, some the computations are brought into network, cloud of service machines. One of the famous examples of cloud services is where users can store their data in the network as if they store the data in their local machines. In fact, this stored data can be accessed wherever they are. This technology has somehow changed the way people use the Internet and their computers.

Although the technology gives lots of benefits and flexibility to its users, there are some areas that need to be taken to concern. In the previous example, how users are sure that their data are being processed according the agreements that have been set at the first place. They have no insight and control about how the processes are going on behind the scene. It is something called ensuring accountability of cloud service. It means that how the users are sure that the services or processes are accountable. The lack of mechanism to verify the policies are in place in the real system is one of the main reasons for difficulty of cloud computing adoption. In Dropbox example, where users can store their data using Dropbox service in the cloud, accountability is important. In this Service Delivery Chain type of scenario, there is a need of way to control the processes within the service chain and make sure that they are complied with service level agreement.

The aforementioned intention is the goal of this Thesis. We want to develop a method or way to be able to detect if there is violation in this type of service, Service in Cloud, based on information that are collected from activities of every component in the scenario. The output of this process is evidence as a proof of something has violated the rules. This then can be basis for auditing process by a trusted pointed cloud auditor. The method is evidence-based i.e. the proposed framework has goal to collect evidence based on source of information about the system's running with respect to obligations or policies defined for services in the system. The framework of evidence collection can be used as basis for providing accountability in cloud. This is because that evidence can be used as a media for making all entities in the system accountable for their actions especially on those that violating obligations.

1.2. Contributions

This thesis proposes framework for evidence collection in purpose of providing accountability in cloud service. The framework focuses on collecting and verifying evidence. Following list are general items produced from this work:

- Source of evidence identification and collection

  Source of evidence gives information about system activities for the purpose of collecting evidence i.e. on violation to policy. Source of evidence needs to be identified. Discussion on how the sources of evidence are identified and collection is presented.

- Evidence collection method

  As part of the framework for evidence collection, collection process collects evidence on violation to the obligations with continuous phase. Method on how the evidence is collected from system information in relation to obligations is presented. The implementation shows how the proposed method can be implemented.

- Evidence verification method

  Verification method verifies if potential evidence from the collection process can be used as evidence as proof of violation to policies. Method on how the potential evidence is verified is presented. Along with the proposed method, implementation shows how it can be implemented.

- Study of policy representation

  All entities in the cloud system have to comply with policies. In the effort of providing accountability by collecting evidence i.e. on violations to policies, the framework is executed based on the policies. To do so, policies need to be represented in machine understandable format and be processable for evidence collection process. The work in this thesis includes study on how to represent the policies. Methods and implementations of the framework then take this information into account.

- Test case implementation

  Use case on cloud service is selected and proposed methods and related implementations are executed on this use case to show how the proposed method works. In order to do so, a test system simulating the use case is also built up i.e. to simulate the use case on healthcare service in cloud. Based on this system, source of evidence is collected to be processed in the evidence collection step.

- Comparison of two approaches used in collection process

  Two approaches on processing source of evidence and policy are shown and compared. The first approach processes them as MFOTL using MonPoly. The second approach processes them as Prolog (FOL) using Pyke.

1.3. Related Works

Accountability is a way for an organization to provide account for their actions in relations to obligations or policies their systems must comply with [1]. In realizing this condition, some actions must be taken at monitoring what the systems are doing and checking if the activities comply with the policies or obligations that have been set up. As a result, evidences as proof of something has happened are constructed from the monitoring and checking activities. As explained in [1] that accountability evidences are collections of data that provide verifiable account about fulfilment of obligations with respect to observable system, the goal of monitoring are to get evidences if something happened in the system are to be claimed happened. This section discusses some of the works that are trying to solve problems relating to monitoring for the purposes of providing account of actions happened in the system with respect to obligations or policies that have been set up.

- Policy Monitoring using MFOTL

[2] provides approach on monitoring system policies. The policies are expressed using expressive fragment of temporal logic. A case study is used to show the effectiveness of the specification language on compliance monitoring together with the monitoring algorithm that is developed based on this specification language. The algorithm is based on monitoring to verify system properties by using algorithm to check whether a system trace satisfies a temporal property. Temporal logic used for expressing policies is called Metric First-Order Temporal Logic (MFOTL). MFOTL is an extension of metric temporal logic [3]. Monitoring algorithm proposed is evaluated by monitoring policies on synthetic data streams. MFOTL formulas expressing policies are described over signature that registers all the relations in the observable system. The monitoring takes place on temporal structure which expressing all events happened in the system with time information. To detect violations, monitorable formulas are written in negation form. Then, the monitoring algorithm iteratively processes the temporal structure, evaluating the formula at each time point. Example case of monitoring is to read log files and report policy violations.

A simple example provided in the work is about monitoring publish/approve report policy on system log. The policy requires that before a report is published, it must be approved. System log containing events happened is depicted in Figure 1.1.

```
2010-03-04     archive_report (Alice, #104)
     .                 .
     .                 .
2010-03-09     approve_report (Alice, #248)
     .                 .
     .                 .
2010-03-13     approve_report (Alice, #234)
2010-03-13     publish_report (Bob, #248)
     .                 .
     .                 .
```

Figure 1.1. Log for Publish/Approve Example

The temporal overview of events in the system log is shown as below in Figure 1.2.

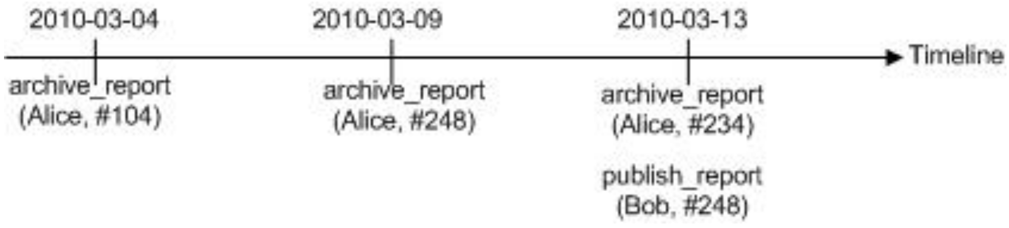
2010-03-04
2010-03-09
2010-03-13
Timeline
archive_report (Alice, #104)
archive_report (Alice, #248)
archive_report (Alice, #234)
publish_report (Bob, #248)

Figure 1.2. Temporal Overview for Publish/Approve Example

The policy is expressed in MFOTL as

$$\Box \forall e. \forall r. publish\_report(e,r) \rightarrow \blacklozenge_{[0,11)} \exists m. approve\_report(m,r)$$

The result of running the monitoring algorithm using defined policy on the system logs result in no policy violation detection where the event *publish_report(Bob, #248)* is preceded by event *approve_report(Alice, #248)* which is by meaning report #248 has been approved when it is published.

The work also presents analysis on space requirement for their monitoring algorithm. Since the algorithm iteratively processes the temporal structure in temporal database, the upper bounds are given in terms of the processed prefix. The largest portion of memory usage is the space needed to store relations of the extended structures [4].

- Forensic Standard

[5] works on providing ISO 27037 which is the first of a developing family of international standards that try to create common baseline of the practice of digital forensic. Digital forensic has close relationship with cloud computing domain where it becomes one of the tools to tackle challenges of cloud computing related to privacy and security. Digital forensic is used to find any unappropriate actions through investigations in the multi-tenant, highly virtualized environment that cloud service exposes. Standard in ISO 27037 is intended to facilitate usability of evidences obtained in one jurisdiction operating in another jurisdiction. It addresses the steps of forensic as identifying, obtaining, and preserving potential digital evidences.

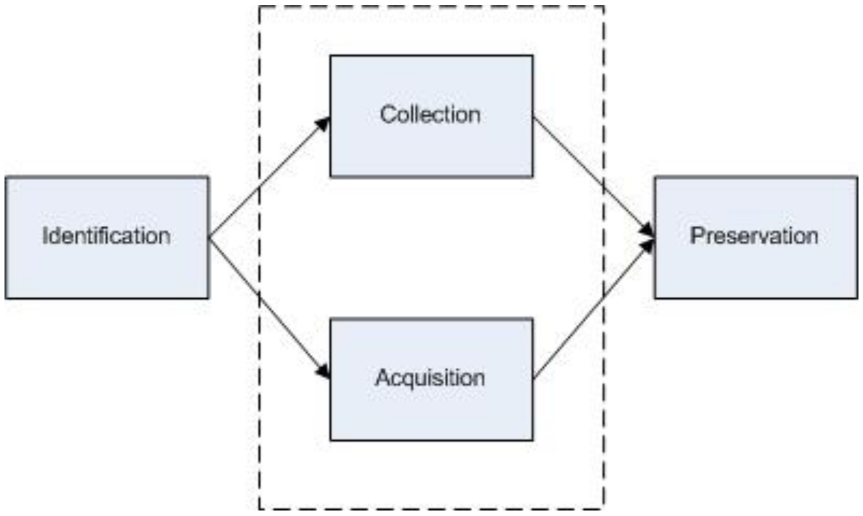

Figure 1.3. Steps of Digital Forensic

Figure 1.3. shows the steps of processing evidence as part of digital forensic proposed in [5]. The process starts with identifying any data that could be potential evidences. Formally identification is the process involving searching for, recognizing and documenting of digital evidence [6]. Obtaining step following identification may be either collection or acquisition. Collection is taking items containing potential evidence and removing them for further processing and analysis. While acquisition is taking copy of items so as to minimize business impact because of ongoing investigation. Once the potential evidences have been collected or acquired, they must be preserved. Storage requires strict access controls to potential evidences from any undesired modifications.

The work also defines general guideline on how the digital evidence handling should be in cloud setting in terms of identification, obtaining, and preservation. It compares the standard ISO 27037 and how it is implemented in cloud setting. It also identifies which items that could potentially be the evidence in cloud setting.

Process following preservation is analysis of potential evidences. [5] defines analysis of potential evidences as serial of several steps which are depicted in Figure 1.4.

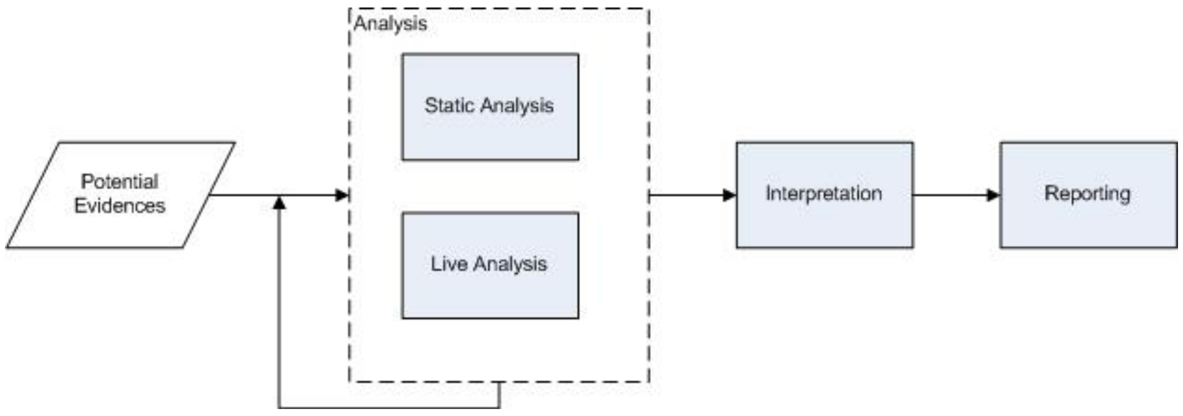

Figure 1.4. Analysis Step of Digital Forensic

It defines analysis as identify and evaluate potential evidences whether they are valid evidences or not. Analysis may be static (by inspection only) or live (analysis on site). Interpretation following analysis tries to define the meaning of analysed evidences. The result is presented in the reporting step.

- Logical Method for Policy Enforcement over Evolving Audit Logs

Work discussed in [7] proposes iterative algorithm for enforcing policies which are represented in first-order logic. The algorithm checks over evolving logs. It means that, in each iteration, the algorithm tries to enforce as much policies as possible over current log and outputs residual policies for next iteration when logs are extended with additional information.

Policies are represented using what is called PrivacyLFP [8], an expressive first-order temporal logic. It is claimed that the language is more expressive than propositional temporal logic [9]. The iterative enforcement algorithm works on incomplete logs which are represented as three-valued partial structures that map each atomic formula of policies to either true, false, or unknown.

In the logic syntax, there are propositional connectives T (true), $\perp$ (false), $\wedge$ (conjunction), $\vee$ (disjunction), $\neg$ (negation), and first-order quantifier $\forall$ and $\exists$ with restriction formula. The syntax also include standard connective of Linear Temporal Logic (LTL) that provides quantification over the sequence of states. It also assumes each state has timestamp associated with it.

Application example is presented in the work. One of them is policy about disclosure of health information from one entity to another in that it is only allowed if the receiving entity is patient’s doctor and the purpose is for treatment, or the patient has given consent about. The policy is represented in the logic as

$$\alpha_{pol1} = \forall p1, p2, m, u, q, t.(send(p1, p2, m) \wedge purp(m, u) \wedge \\ tagged(m, q, t) \wedge attr\_in(t, phi)) \\ \supset (inrole(p2, doc(q)) \wedge purp\_in(u, treatment))$$

$$\vee \Diamond consents(q, sendaction(p1, p2, (q, t)))$$

The iterative enforcement algorithm matches the policy representation with incomplete logs which are defined in partial structures.

- Open Architecture for Digital Evidence Integration

[10] works around digital evidence bags which become important part on storage and sharing of digital evidence between organizations. It proposes an architecture for digital evidence bags which is developed on top of Turner's digital evidence bags concept [11]. This architecture overcomes some shortcomings from the previous concept. It treats bags as immutable objects, and facilitates the building of digital evidence corpus by composition and referencing between them.

Digital evidence bags contain several components which are:

- Evidence Metadata Records
  This record contains information about description of the evidence, the location and time of the evidence acquisition.

- Provenance Records
  This record contains chain of information related to the evidence which describes whole story of events constructing the evidence.

- Identification Records
  This is to uniquely identify the bag. It may contain other case related information such as case number, item number, collecting organization, suspect and victim.

- Integrity Device
  Integrity Device is built in form of seal that protect the evidence inside the bag so as to provide integrity for the evidence itself.

- Evidence Container
  It is the inside of the bag.

The architecture for digital evidence bag proposed in the work is called Sealed Digital Evidence Bags as shown in Figure 1.5. Similar to Turner's digital evidence bags, the architecture has tag file, metadata file, and the digital evidence bags. Tag file contains integrity information. Metadata defines digital evidence bags associated with it. The identification for digital evidence bags uses RDF's URI approach.

The work also developed a prototype online acquisition tool for creating digital evidence bag containing images of the Internet Explorer cache and history index files.

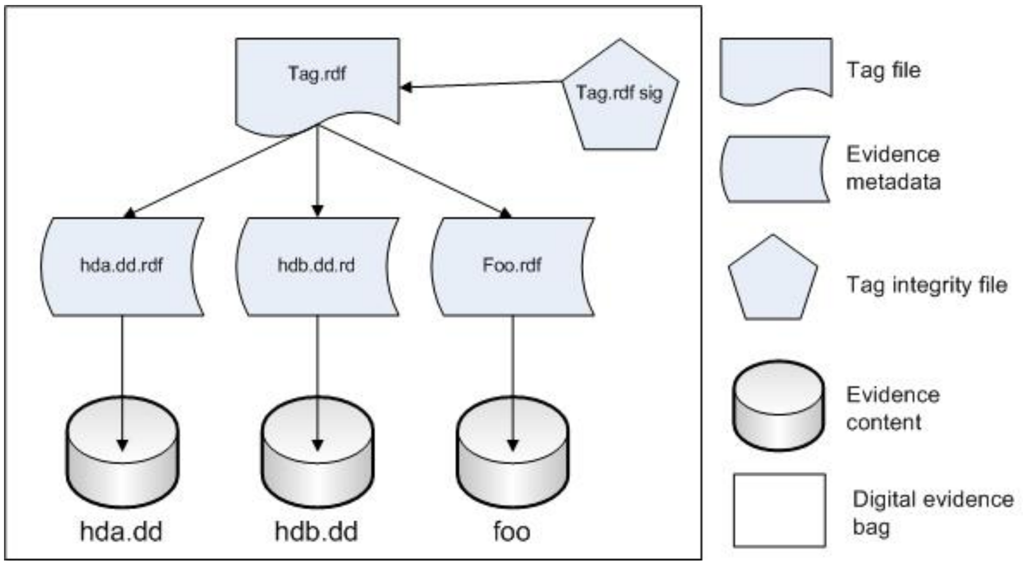

Figure 1.5. Digital Evidence Bag

- Evidence of Log Integrity in Policy-based Security Monitoring

[12] tries to solve a problem when logs containing information about system activities of an organization have potential to be modified by malicious entities to hide any malicious activities that are not compliance with the policies defined. It proposes cloud-based framework to ensure log integrity based on small amount of evidence data. A simple Cloud Security Monitoring (CSM) API is made available for organizations operating in cloud to retrieve additional information about their systems. This information is used to verify system compliance against policies.

The approach used is to identify minimal sets of events needed to construct proofs of policy compliance using only information gathered through CSM API.

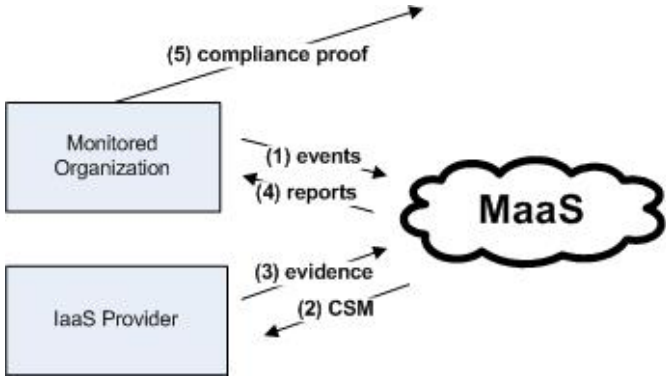

Figure 1.6. CSM API in Cloud

Figure 1.6. shows the introduction of CSM API that enables the monitoring system to retrieve additional information for verification of system compliance with policies to the organization using cloud service. Policies are represented as rules and policy violations are specified as indicating sequences of events that are not supposed to occur in the system. Events are characterized by type, set of parameters, and two timestamps which are start and end times.

The approach discussed in the paper is tested on a case study on Payment Card Industry Data Security Standard (PCI-DSS) policies which is intended for cloud service providers handling credit card data of credit card companies. Source of evidences for each policy are identified such as network traffic, firewall information, running programs, documentations, etc. Data for the sources are collected through monitoring systems such as cloud configuration information, network monitoring, VMI (Virtual Machine Introspection), etc.

Experiments on 4 monitored policies show that the approach is able to monitor 100% for 2 of them and 37.5% and 33.3% for the other two.

- Log Design for Accountability

[13] addresses the problem on designing logs of system activities so that analysis on policies or rules compliance is supported. This problem started from the fact that Personally Identifiable Information (PII) is becoming more often be shared by Data Subjects in exchange for services. As this type of data is sensitive, it is then become more important legislation on how this data should be collected, distributed, and accessed. In current approach, Data Controllers are allowed to manipulate data and are trusted to follow rules. However, they are still accountable for their actions through analysis on information on how the activities are performed.

The work in the paper is based on building formal definition of PrimeLife Privacy Policy Language (PPL) which is first presented in [14] and is partly build on XACML. PPL is used to express access and usage control rules. The work formalized and implemented log compliance analyser against rules that are specified in PPL. Figure 1.7. shows the components of the work.

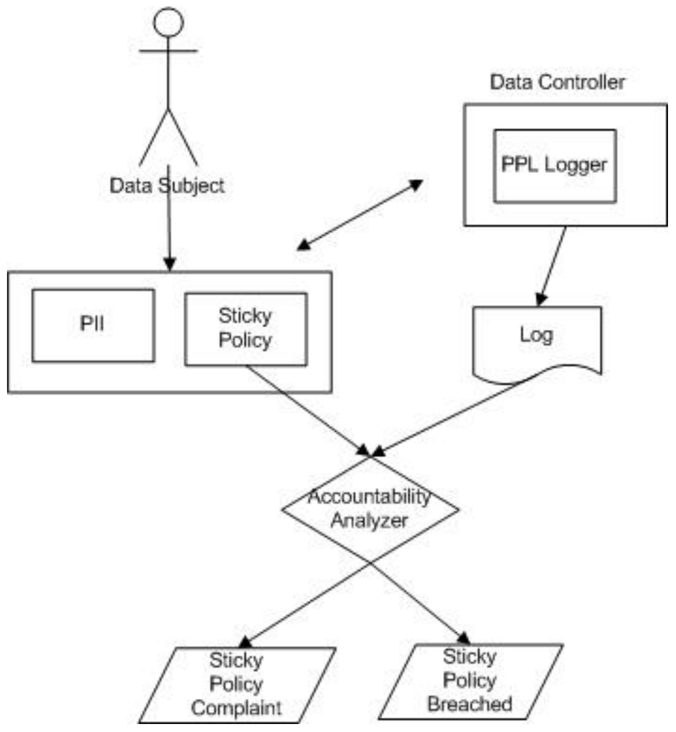

Figure 1.7. PPL Accountability Analysis Framework

Data subjects define rules in terms of sticky policy. Data controller must log all the events occurred in the system. The log information is used by analyser to determine if the policy defined by data subject is followed or not. Obligations as rules are defined in terms of trigger and actions. Several triggers have been defined in PPL such as when PII is deleted, is updated, is accessed, is forwarded to other entities, etc. All available triggers and actions definition are given in the paper. The definition in PPL also includes authorizations which declare whether PII can be transmitted to other entities and for which purposes it may be used.

Example scenario is introduced in the work to illustrate issues that might arise when analysing log. The scenario is on data handling events for private bank account. Several obligations are defined relating to the requirement for bank to send related notification to customer. Issues that might arise can be insufficient event information which arises from missing parameter in log entries for given events which result in undecidability of obligation compliance. Other issues can be incomplete support for third party interaction which cause events for complying with obligations are not generated or can be no support for manual verification because there is no comprehensive information such as actual content of an event.

At last, the paper presents guidelines for accountable log design based on the issues that might arise. Some of them are that log architecture should reflect full policy language semantics, that there should be links between formal specifications and policies requiring human verification.

- Data Handling Requirements-aware Cloud Computing

Work presented in [15] identifies challenges for enabling data handling requirement awareness in cloud service. It is due to the fact that in order for cloud service to be utilized properly, it should have mechanisms for users or companies to define requirements on how their data should be treated and for the providers to be exercised on how they process the data based on requirements set up. Some examples of the requirements are restricting on how long and where a specific piece of data might be stored. Cloud provider must meet the requirements and are monitored on doing so. The work then presents high-level solution for data handling aware cloud computing. The idea proposed is to enrich data with data handling annotation using PrimeLife Privacy Policy Language before it is uploaded to cloud.

Two main challenges for cloud data handling are location of storage and duration of storage. Location of storage challenge comes from the situation where users usually have requirement that their data is stored in the jurisdiction of their location or where EU has different data handling policy than US does. When this situation is applied to cloud, then the challenge appears. On the other hand, there is usually data storage duration requirement. This condition poses challenge on providing way for making sure that cloud providers meet this requirement.

Figure 1.8. shows data handling requirement aware cloud proposed in the work. User creates data handling obligations in data annotation before handing over his data to cloud service. In receiving this annotation, cloud provider matches the obligation with the policies it has. When matches, it signs the data annotation and sends back to user. Since then, it is responsible for following the data handling obligations. In the cloud stack architecture, cloud provider makes the same agreement with another provider in purpose of following the obligations. Broker is utilized for determining the most appropriate provider in terms of QoS, SLA, pricing and support for data handling requirements as well.

For formalize data handling obligations, PrimeLife Privacy Policy Language (PPL) [16] is used. Obligations are defined as set of triggers and actions. When trigger occurs, defined action must be executed.

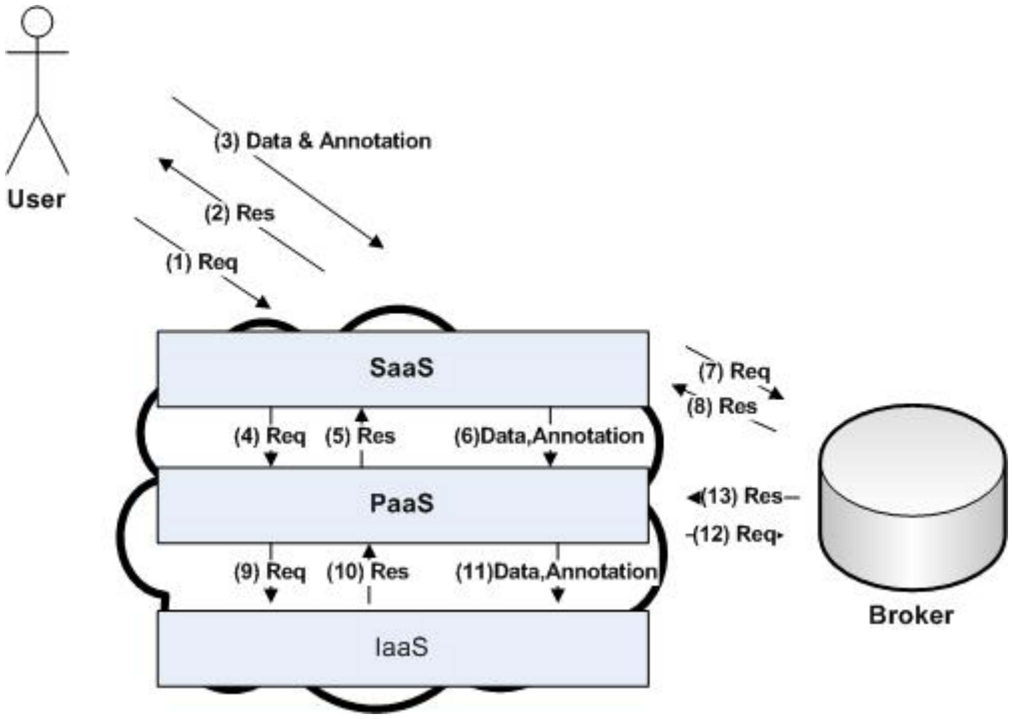

Figure 1.8. Data Handling Requirements-aware Cloud Stack

- Obtaining and Admitting Electronic Evidence - Using Log Record Analysis to Show Internet and Computer Activity in Criminal Cases

Paper [17] presents a log record analysis for revealing criminal activities that happened in a system. A hypothetical scenario is presented which is an attacker exploits system vulnerability to gain unauthorized access. The work started with explaining possible log information associated with each type of activities that might occur in the system. Then types of logging devices are presented which produce logs such as firewall logs, web server access logs, FTP server logs, Proxy server logs, etc. Typical information that can be extracted from the logs is IP address, timestamp, userid, request, HTTP information, etc.

Three steps for obtaining log record proposed in the work are identification, preservation, and collection which correspond to identify types of records, preserve records, and use legal process to collect records respectively.

After dealing with obtaining log records, the next step is to conducting log analysis. Log analysis is divided into 5 substeps which are:

- Data Collection
  This step involves assembling log records from several sources to be used for analysis. By combining logs from several sources, analyst can confirm and corroborate activities in the system

- Data Normalization
  This step involves parsing, filtering, and revealing additional metadata that can help collection process. As the data may come from different sources with different format, normalization is trying to produce the same format with key information for all logs coming from different sources.
- Analysis
  This step includes review for log entries in relation to investigation. Normalization results in normalized fields for the log entries which can give information for analysis process. Normalization on time also produces time line for events which can lead for the investigation.

- Correlation
  This step involves comparison and confirmation of common records from different logs. This activity may lead to new information by combining extracted information from different logs.

- Report
  This step includes summarization of the information extracted from data set.

Several sample cases on using log records for analysis are shown for example revealing specific activity in email account, revealing posting and deleting content activity on the Internet, etc.

1.4. Organization of the Thesis

This thesis is organized in following way:

- Section 1 explains about problems in current cloud service that this thesis is trying to solve, expected results from the work in this thesis, and some works that have been done related to policy monitoring and enforcement in cloud service in providing accountable cloud system

- Section 2 explains background information about items proposed and discussed in the next section – proposed method and implementation which includes accountability cloud and policy monitoring using logic concept

- Section 3 explains proposed methods for solving problem mentioned in section 1 with selected use case. This section also explains about implementation of the proposed method in detail. Explanations also include data collection for applying proposed method.

- Section 4 presents information about results from applying the method proposed in section 3 for the selected use case together with background information leading to the results

- Section 5 concludes the work in the thesis which includes problem, method to solve the problem, and method and implementation result

2. Background

This section describes some background information about terms, techniques, technologies, tools, etc. that are related to this work. Accountability in cloud service concept is explained together with evidence collection and processing concept associated with it. Next, policy or obligation monitoring system is explained. This explanation includes temporal logic which is used in the existing monitoring system. At last, information related to data collection for performing analysis is presented.

2.1. Cloud Service

A cloud service is a service made available to users from cloud provider's premise as opposed to being provided from company's own premise [18]. A company providing service may put their application or service in other provider's premises because of efficiency reasons. If, for example, there is another provider which provides data storage service. the company that data storage services to store its users or clients' data.

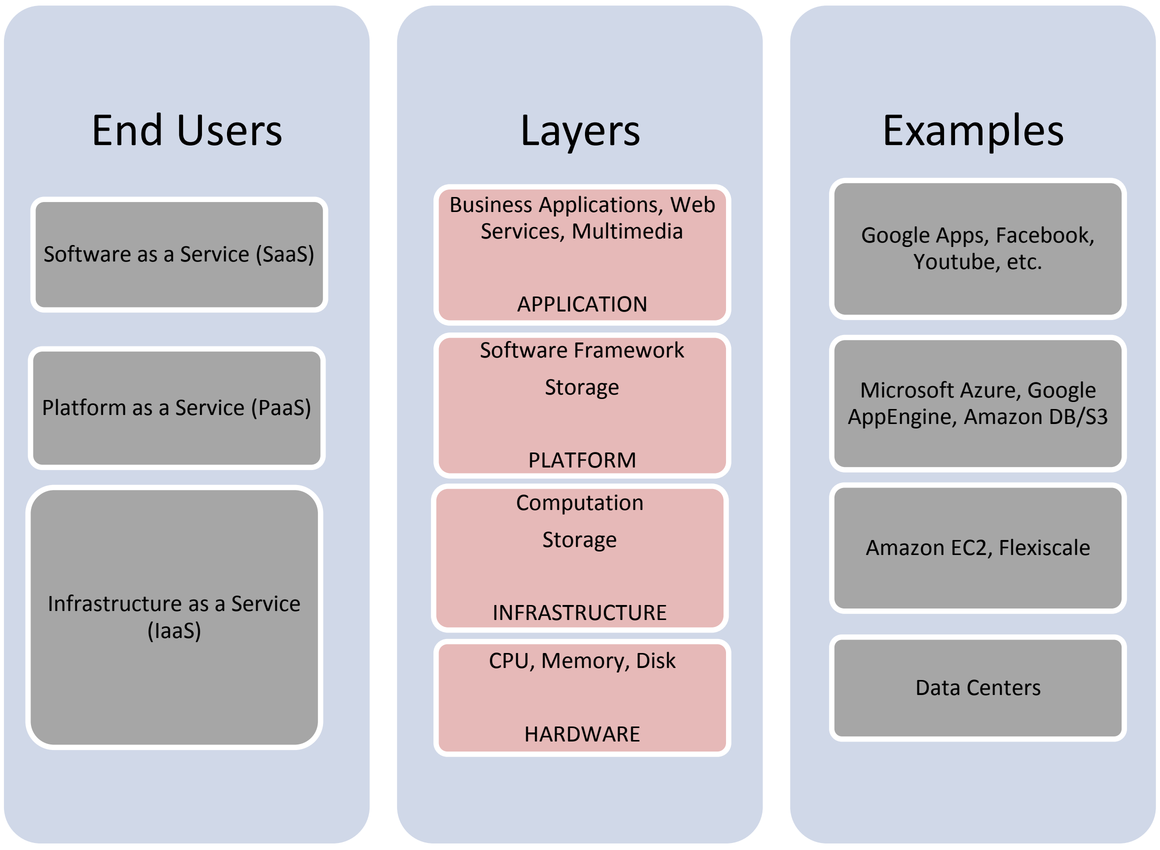
End Users
Software as a Service (SaaS)
Platform as a Service (PaaS)
Infrastructure as a Service (IaaS)
Layers
Business Applications, Web Services, Multimedia
APPLICATION
Software Framework
Storage
PLATFORM
Computation
Storage
INFRASTRUCTURE
CPU, Memory, Disk
HARDWARE
Examples
Google Apps, Facebook, Youtube, etc.
Microsoft Azure, Google AppEngine, Amazon DB/S3
Amazon EC2, Flexiscale
Data Centers

Figure 2.1. A Cloud Service

Figure 2.1. (adapted from [18]) shows model of cloud service. Services provided to users may be built up from several providers. Users access service from a point of contact and the services are enabled by combination of providers that build up the service for example provider that responsible for hardware or infrastructure (IaaS), for platform (PaaS), or for application (SaaS) [19].

2.2. Accountability in Cloud

Accountability is becoming main concept in the cloud service paradigm that helps increase trust in cloud computing [20]. It is related to corporate data governance. It is mainly about how the data in the cloud is governed. The level of governance must meet or comply with the agreement that have been set up between communicating parties. Accountability ensures the party which is measured is responsible for the activities it has done. In definition, accountability is the obligation to act as a responsible steward of the personal information of others, to take responsibility for the protection and appropriate use of that information and to be accountable for any misuse of that information [21]. Lack of way to trust the cloud provider is one of the reasons that the adoption of cloud service is inhibited.

Accountability for cloud arises as the consequence of the rising of cloud service. In this type of service, users are giving their data to cloud service providers. This handover causes the customer no way of controlling how their data are stored and processed [21]. They have at first place defined an agreement with the service provider. But later, they don’t have way and tool to control if the agreement is respected. Therefore, it is necessary that customers are given way to control how the cloud service provider treats their data according to the agreement. And we say that the cloud service provider must be accountable for its activities so that customers have way to measure on how much the service provider respects the agreement in respect to processing their data.

There is a need to provide accountability in the cloud by providing mechanisms and tools to measure cloud service provider about its activities regarding customers’ data which must comply with the agreement. There are two types of ways of providing accountability in the cloud [21]:

- Prospective (and proactive) accountability using preventive controls
  Preventive controls include risk analysis tools, trust assessment.
- Retrospective (and reactive) accountability using detective controls
  Detective controls include auditing, tracking, reporting and monitoring.

2.3. Service Delivery Chain

Service delivery chain is the condition where service is provided to customer through a service chain which involves several agents [18]. Some of the service agents are hidden from customer. Customer does not have direct interaction or communication with those agents. They only have direct interaction and communication with the rest service agents. Customer's data may be travelled along the service chain. Therefore, the customer wants to ensure that all the agents in the chain respects all the service agreements that have been set up between customer and the direct service agents.

The situation may happen for example when a cloud service provider offers a service to customer. However, this service provider does not have full facilities to support this service. Therefore, it needs some more supports from other service agents that offer different type of service needed for providing the service to customer. Typically in cloud service, there are several types of service models. They are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), and Network as a Service (NaaS). In this example, the cloud service provider may need other service agents that provide one or more service models that it does not have.

Typical service delivery chain is shown in following Figure 2.2.

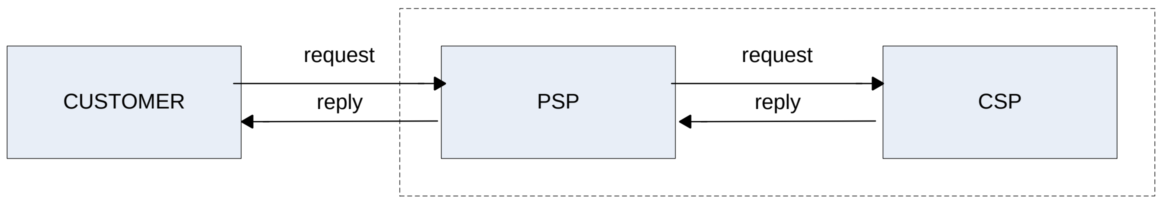

Figure 2.2. Service Delivery Chain in Cloud

In the figure, Primary Service Provider (PSP) is the one going to give service to customer. In order to give service to customer, PSP needs supports from other service providers. They are called Cloud Service Provider (CSP). Customer in this case only interacts with PSP. It does not have direct interaction with CSP. Customer data may travel along the service chain. And it wants to ensure that both PSP and CSP follow the service agreements that it has set up with PSP.

The simple and obvious real example of service delivery chain in cloud service is Dropbox service. Dropbox is a file hosting service that offers cloud storage, file synchronization, and client software [22]. Dropbox uses Amazon's S3 storage system to store the files [23]. In this case, customers using Dropbox service only interacts with Dropbox and signs agreement with Dropbox. However, their data are actually stored in another entity which is Amazon.

As type of cloud service is diverse, we will only work on this type of scenario. The goal is to have a way to control over the chain about customer data processing and treatment.

In relation to accountability in cloud context, PSP is accountable for respecting customer's preferences in terms of following obligations and accountable for what it does to customer data [1]. CSP is accountable for data stewardship of customer personal information to PSP and regulators.

2.4. Evidence Collection / Processing

Evidence can be one of the tools to provide accountability in the cloud. Evidences are collected from the activities of all components in the cloud services. In relation to accountable cloud, evidences are collected based on the policies or obligations defined that must be followed by the cloud providers. Schema for evidence collection is derived from the definition in the policies and obligations. This results in evidences that are highly valuable for further investigation process for ensuring accountability in the cloud.

Evidences then become sources where all the monitoring and auditing processes to provide accountability in the cloud are based one. It gives a way to measure if all components in the cloud service follow rules that govern the system. Processing the evidence means collecting information from components' activities and giving alarms when there are activities that are not comply with the rules.

Evidence processing is necessary [18]:

- For cloud service provider, to make sure that it complies with all rules and agreement that have been set up.
- For cloud customer, to monitor provider compliance. This gives a way for customers to control over their data processing by provider if the processing complies with the agreement they have with the provider. This can overcome the problem for adoption of cloud service where previously one of the factors is there is no way for customers to monitor how their data being processed by provider.
- For cloud auditor, to audit all activities of all components within cloud system and figure out if there are any violations made by each component.

2.5. Policy / Obligation in Cloud Context

A policy is set of rules related to a particular purpose [24]. Rules can be expressed as obligation, authorization, permission or prohibition. In accountability context, policies are set of rules to allow users to define preferences or requirements on how their data are to be treated in the system. Process of evaluating cloud provider in treating users' data is based on these policies. This way cloud provider is accountable for its action in relation to processing users' data.

Policy is needed in the cloud service as a way to govern how the data is processed throughout the cloud. Security and privacy issues usually arise when sensitive data is moving between different cloud service providers within the cloud. In this case, there should be way to govern how this thing is going on. And defining policies or obligations is the way that users, data controller, or data processors can take to define how they govern the service in cloud. Later on, these policies or obligations act as basis for auditing whether all entities in the cloud commits to the policies defined or not. This way, it can be ensured that all actions in the cloud are accountable.

Before, users hand in their data into cloud, they define obligations about how their data should be handled. These obligations are to be guaranteed by cloud service provider. When cloud service provider agrees with the obligations, it sends back notification to users as confirmation that it will follow the obligations. Now the provider is responsible for following the data handling obligations. In cloud service chain, this administration also takes place between several cloud providers in the chain and the primary service provider.

In order to specify obligations or policies, there must be a representation that can be used to express those obligations that are processable in purpose of auditing and making sure that all the obligations are followed. One of the representation is discussed in next section 2.6. Policy Representation (PPL, AAL, A-PPL).

2.6. Policy Representation (PPL, AAL, A-PPL)

In order to provide accountable cloud, there should be way to monitor activities occurred in the system in relation to the rules or obligations that are set up as policies to be followed by each component in the system. For monitoring purposes, representation of the policies is required. A policy language allows concrete policies to be represented. The language enables representation of rules for governance of users data processed in the cloud service. This representation is in machine-readable format so that automatic monitoring and enforcement of policies is possible. Automatic policy monitoring makes sure that every action in the system that violates the policies is reported with information about the actor. Evidence proves that actor is responsible for such action.

A number of policy representation languages have been introduced in recent years. [24] reviewed existing policy languages and studied their suitability for expressing policies in cloud accountability context. It firstly defined requirements for policy language to be used in accountability context. The study showed that none of the existing policy languages is suitable to express accountability policies. It then proposed new policy language called Accountability PrimeLife Policy Language (A-PPL). A-PPL is developed based on existing PrimeLife Policy Language [16] (PPL) which covers data handling issues and can be extended to address accountability requirement for policy language.

Figure 2.3. (taken from [24]) shows proposed A-PPL framework which translate human readable policy into machine understandable representation (A-PPL) through intermediate state of AAL. The translation to machine understandable format involves usage of temporal logic to produces policy sentence in with temporal logic properties. Policy calculus is used to describe the semantics. With temporal properties and semantics, the policy is translated to machine understandable

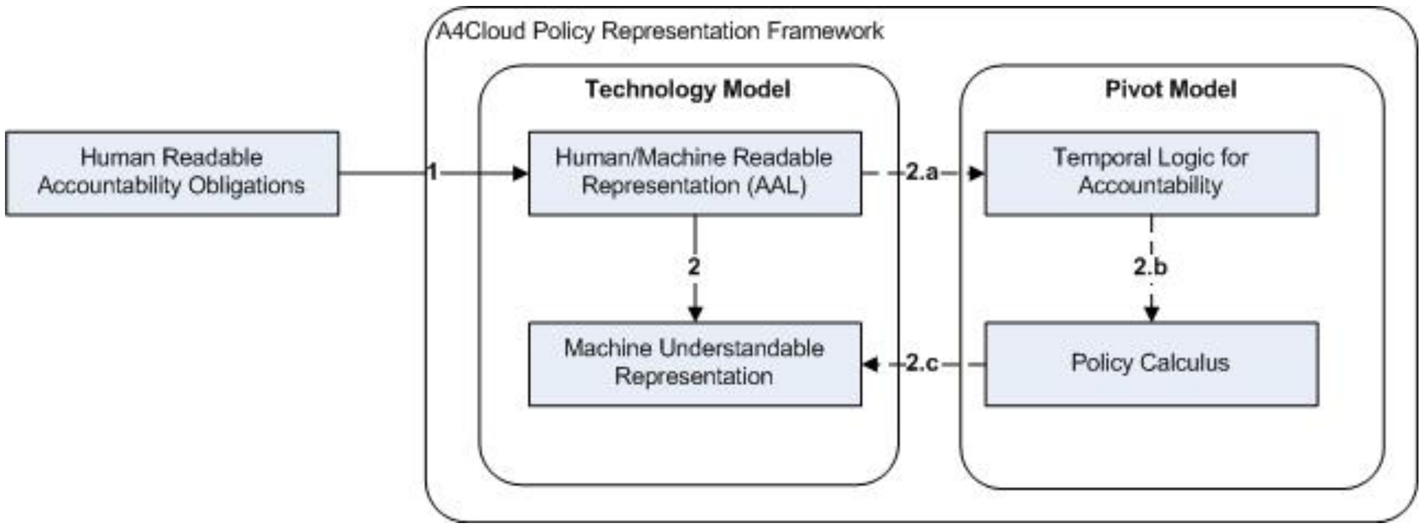
A4Cloud Policy Representation Framework
Technology Model
Pivot Model
Human Readable Accountability Obligations
1
Human/Machine Readable Representation (AAL)
2.a
Temporal Logic for Accountability
2
2.b
Machine Understandable Representation
2.c
Policy Calculus

Figure 2.3. Machine Understandable Translation of AAL

PPL is XML-based policy language which combines access control and data handling policies. A PPL policy has general structure as shown in Figure 2.4 (taken from [24]). Three important section of a PPL policy are access control, authorizations, and obligations.

- Access control specifies credentials need to be presented by requestor to be granted access to the system.
- Authorizations specifies actions that data controller is allowed to perform in respect to the defined purposes of data usage.
- Obligations define how data controller treats data subject personal data.

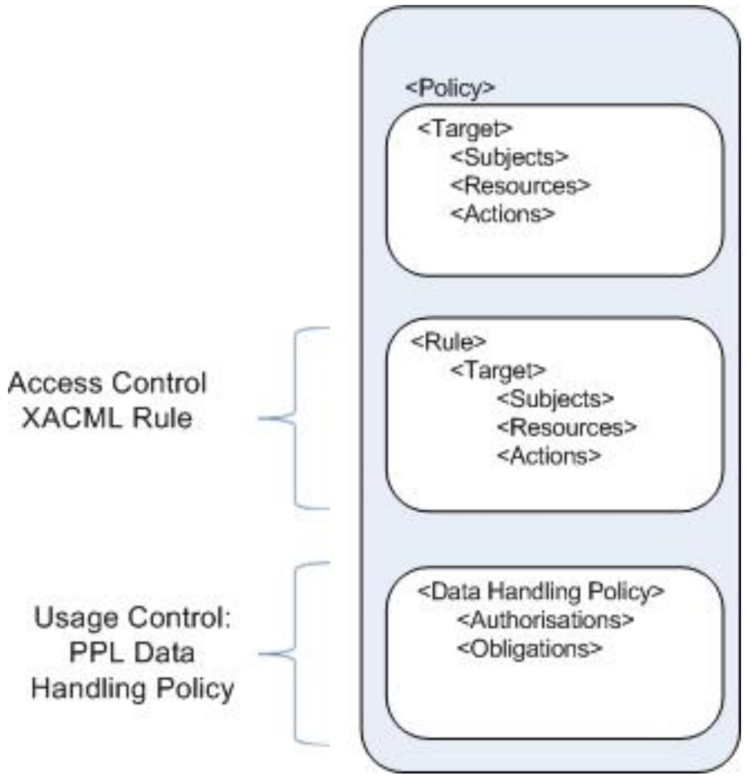

Figure 2.4. Structure of PPL Policy

A-PPL introduces extensions to PPL to be able to represent accountability policies. The extensions include:

- Roles
  Roles contain information about entity to which the defined policy is applied. This information is included in <Subject> element
- Actions
  PPL defines obligation as Trigger-Action. When trigger occurs, associated action needs to be executed. A-PPL has listed actions that are required for expressing accountability policies.
- Triggers
  Triggers are events that trigger an action. A-PPL has listed actions that are required for expressing accountability policies.

Figures 2.5. (taken from [24]) shows example of format of policy representation in A-PPL.

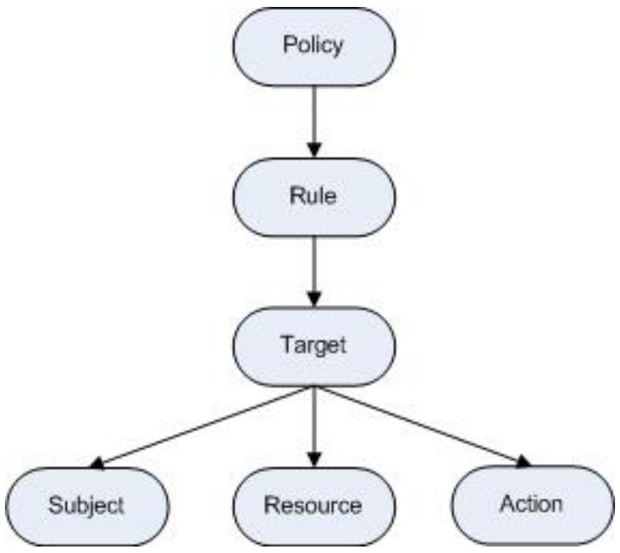

Figure 2.5. Example Policy Representation Structure in A-PPL

2.7. Policy Monitoring

In accountable cloud, every entity is accountable for its action. Evidence is necessary to support the account of an entity in the cloud system. Evidence shows if obligations that are set up for the system are followed. To produce evidence, such system must have mechanism for verifying if obligations are fulfilled. This is achieved by monitoring system activities or events with regards to the obligations.

Policies are represented in machine readable and understandable representation. This representation is used by monitoring engine to examine system events logs. These logs are called source of evidences. If events in the logs are detected to not comply with obligations defined, then violations are detected. This constructs evidence which describes that some events are not complying with some obligations.

Several works have been put on policy monitoring. [25] and [2] proposed policy monitoring framework. Policies are represented in temporal logic. The logic used is able to specify order of events and time information such as past, present, and future. This properties are important as the policies definition will be used to examine source of evidences which are typically contains information about sequence of events occur in the system with time information. Policy monitoring determines whether sequence of events occur in the system satisfies the policies. If not, violations are reported and the information is used as evidence that violations to policies are suspected to occur with information about relevant actors involved.

Figure 2.6. shows how policy monitoring is generally working. The inputs are policy definition and system events logs. Policy definition is expressed using policy representation language. System events logs contain information about system activities. This is the source of information to reveal if there is action that is not complied with the policies. The policy monitoring engine firstly interprets policy definition. It continues by examining logs with the interpretation information and determines if each event in the log confirm with the policies or not and reports the result.

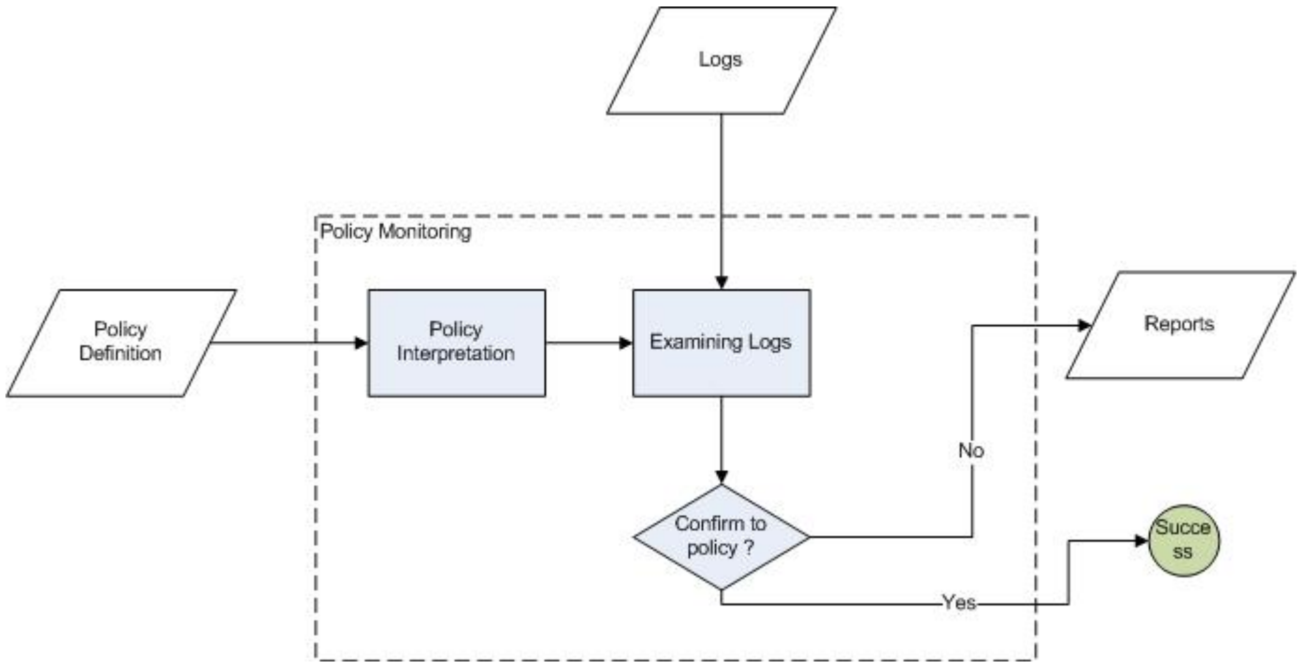

Figure 2.6. Policy Monitoring

[25] shows work on monitoring policies which are represented using MFOTL. Monitoring algorithm is used to monitor whether system behaviour conforms the policies. System events are represented as sequences in temporal structures that suited the definition structure of the policies. The algorithm check if event occurs, then it determines whether the sequence satisfies the policies expressed. If not, then violation is reported.

System events are represented as sequence in temporal structure which is defined over a signature. Signature S = (C,R,a) where C is set of constant symbols, R is set of relations, and a is mapping function. Timed temporal structure D over signature S is defined as events D = (Do, D1, …) with timestamp T = (To, T1, …).

Each policy expressed in a MFOTL formula is monitored. To detect violations, the monitor works with negated formula and outputs for each time point the satisfying assignments of the negated formula. Monitor works sequentially by processing timed temporal structure (D,T) and determines for each time point those elements in (D,T) that violate the formula.

2.8. Temporal Logic (& MFOTL)

Temporal logic [26] is an extension from classical propositional logic where propositions are evaluated to true or false. Temporal logic itself is focussing on propositions whose values depend on time. Temporal propositions contain reference to time conditions. In temporal logic, there are two temporal quantifiers which are "always" and "eventually". Combination of temporal quantifiers can be used to express more complex time conditions. Definition of truth of a formula in temporal logic is given as follow [26]

$M,t \Rightarrow q$	if	$\pi(t))(q) = 1$
$M,t \Rightarrow \neg\varphi$	if	not $M,t \Rightarrow \varphi$
$M,t \Rightarrow \varphi \wedge \psi$	if	$M,t \Rightarrow \varphi$ and $M,t \Rightarrow \psi$
$M,t \Rightarrow G\varphi$	if	$M,s \Rightarrow \varphi$ for all s with $t < s$
$M,t \Rightarrow H\varphi$	if	$M,s \Rightarrow \varphi$ for all s with $t > s$

M is the model in which the formula is to be evaluated. $M,t \Rightarrow \varphi$ saying that formula φ holds at time point *t*. *G* and *H* are temporal operators (in this case "always" and "eventually"). In addition to *G* and *H*, temporal operators are extended to include *S* ("Since") and *U* ("Until"). Definition for these operators is as follow:

$M,t \Rightarrow U\varphi\psi$	if	$M,s \Rightarrow \varphi$ for some *s* such that $t < s$ and $M,u \Rightarrow \psi$ for all *u* with $t < u < s$
$M,t \Rightarrow S\varphi\psi$	if	$M,s \Rightarrow \varphi$ for some *s* such that $s < t$ and $M,u \Rightarrow \psi$ for all *u* with $s < u < t$

Another temporal operator is *X* ("next time"). Formula $X\varphi$ holds at time point *t* if φ holds at the next moment in time.

In addition to point-based temporal logic, there is also interval-based temporal logic. In this type of temporal logic, formula is evaluated at pairs of points representing beginning and end point of the interval. Temporal operators for interval-based temporal logic are *D* ("During") and *o* ("conjunction"). The definition of truth is given as follow:

$M,[s,t] \Rightarrow \langle D \rangle \varphi$ if $M,[u,v] \Rightarrow \varphi$ for some *t,u* with $s \le u \le t$

$M,[s,t] \Rightarrow \varphi \circ \psi$ if $M,[s,u] \Rightarrow \varphi$ and $M,[u,t] \Rightarrow \psi$ for some u with $s \le u \le t$

Metric first-order temporal logic (MFOTL) extends temporal logic [27]. This logic is interval-based. First-order fragment is for formalizing relations on system and metric temporal operators are for specifying properties depending on times associated with past, present, and future. Syntax for MFOTL is given as follow:

$$\begin{aligned} \phi ::= & t_1 \approx t_2 \mid t_1 \prec t_2 \mid r(t_1,\ldots,t_{a(r)}) \mid \\ & (\neg\phi) \mid (\phi \wedge \phi) \mid (\exists x.\phi) \mid \\ & (\bullet_I \phi) \mid (\circ_I \phi) \mid (\phi S_I \phi) \mid (\phi U_I \phi) \end{aligned}$$

A formula ϕ if of the form $(\bullet_I \phi) \mid (\circ_I \phi) \mid (\phi S_I \phi) \mid (\phi U_I \phi)$ where $\bullet_I, \circ_I, S_I, U_I$ are temporal operator "Previous", "Next", "Since", "Until" respectively and *I* is time interval and is bounded. These basic grammars are then extended to include more temporal operator such as:

- "Sometime in past" : $\blacklozenge_I \phi :=$ `true` $S_I \phi$
- "Always in past" : $\blacksquare_I \phi := \neg \blacklozenge_I \neg \phi$
- "Sometime in future" : $\lozenge_I \phi :=$ `true` $U_I \phi$
- "Always in future" : $\square_I \phi := \neg \lozenge_I \neg \phi$

Semantics of MFOTL are defined with respect to timed temporal structures. These structures typically represent sequence of events.

Illustration the usage of MFOTL in expressing policy is given as follow. Consider there is policy about publishing business reports within a company. The requirement is that before a report is published, it must have been approved. To express this policy in MFOTL, two relations are registered in the signature which are PUBLISH and APPROVE. The policy then is expressed in MFOTL as:

$\square\ \forall f.publish(f) \rightarrow \blacklozenge\ approve(f)$

Adding more information to the example such as to define policy like "whenever a report is published, it must be published by an accountant and the report must be approved by her manager within at most 10 time units prior to publication" gives us MFOTL formula as:

$\square\ \forall a.\ \forall f.publish(a,f) \rightarrow acc(a) \wedge \blacklozenge[0,11)\ \exists m.mgr(m,a) \wedge approve(m,f)$

Additional relations "being accountant" and "being manager" is defined as:

- *acc(a)* is defined as $\neg acc_F(a) S acc_s(a)$ where *s* marks the time when a becomes an accountant and *F* is the finishing time
- *mgr(m,a)* is defined as $\neg mgr_F(m,a) S mgr_s(m,a)$ stating that m is manager of a.

2.9. Source of Evidences

Evidence allows assurance of accountability in cloud services. Verification of compliance to obligations or policies by monitoring and auditing system activities, as part of providing accountable service, results in evidences that show activities in the system that are not complied with the obligations. Evidence collection involves capturing, integrating and processing of system information with respect to policies.

Evidences might be derived from many sources. As input to evidence collection is information about system activities that can be used to reveal information if there is no compliance to obligations or policies. Such input information to evidence collection is called source of evidences.

In cloud service environment, [28] mentioned sources of evidence by logging. Logging can include business relevant log or operational log. Logging is performed in several levels such as system level, data level, service level, business level, etc. Typical information required in the log is data creation, data access, data flow, data type, data deletion, data handling, and data notification. This information can be used for analysing whether the policies or obligations have been followed.

Dividing cloud system into SaaS, PaaS, and IaaS, [5] defines potential source of evidences in each level.

Potential source of evidences in SaaS environment includes

- web server logs
- application server logs
- database logs
- guest operating system logs

- host access logs
- virtualization platform logs
- network logs

Potential source of evidences in PaaS environment includes

- web server logs
- application server logs
- guest operating system logs
- host access logs
- virtualization platform logs
- network logs
- management portal logs

Potential source of evidences in IaaS environment includes

- cloud or network provider perimeter network logs
- logs from DNS servers
- virtual machine monitor logs
- host operating system logs
- API logs
- management portal logs
- packet captures

2.10. MonPoly Monitoring Tool

MonPoly is a tool to monitor log files for policy compliance [29]. Policy defines obligation that need to be followed by every component in the system. Log files as one of the sources for information about activities occur in the system is used as a basis for monitoring system activities for the policy compliance. Events information in log file is time-stamped and is ordered based on the timestamps. MonPoly is developed based on the concept introduced in [27]. Policies are expressed in MFOTL (Metric First-Order Temporal Logic) formula. The monitoring is implementing algorithm proposed in [27] by taking information about policy in form of MFOTL formula and information about system activities in form of log file. It then reports any violations to policies based on information in log file.

MonPoly [30] takes command-line input signature file, policy file, and log file. Signature file registers all possible events in the log file. Log file contains information about events occur in the system that are going to be checked against policies defined whether they comply or not. Policy file contains policies expressed as MFOTL formula. MonPoly runs the monitoring algorithm and output violations to policies.

Following example how MonPoly is used to monitor a policy in log file. Supposed that the policy to be monitored is “financial report must be approved at most a week before it is published”. MFOTL formulation of this policy is

$\Box\ \forall\, r.\mathit{publish}(r) \rightarrow \blacklozenge_{\leq 7\text{days}}\ \mathit{approve}(r)$

In MonPoly, in policy file, this policy is written as

```
publish(?r) IMPLIES ONCE[0,7d] approve(?r)
```

All predicates used in the policy definition such as publish and approve are registered in the signature file. MonPoly output reports that are not compliance to the policy in free variable *?r*. It tries to match every instance of the variable in the log file with negation of the formula. If such instance is found, then it is said that the instance is not following the policy. If the log file contains information as following

```
@1301252862 approve (1)
@1301675201 approve (2)
            publish (3)
@1302197200 approve (4)
            publish (2) (1)
```

MonPoly processes the log incrementally and output for each time point all policy violations as following

```
@1301675201 (time-point 1): (3)
@1302197200 (time-point 2): (1)
```

The output tells that publishing report 60 and 52 each violates the policy. Looking at the log, report 60 was never approved and so was report 52.

MonPoly is written in OCaml programming language [29]. The implementation is functional and module-based. There are module for MFOTL formula, relations and, temporal structures, parsing formula and log file, and monitoring algorithm.

Currently, MonPoly only supports monitoring one policy at a time. To monitor several policies at once, conjunction of the policies is suggested. However, using this way, specific pointing to policy to which events in log file violate is not possible.

2.11. Pyke

Pyke [31] introduces form of Logic Programming, which is inspired by Prolog, to Python language. It provides knowledge-based inference engine written in Prolog. Pyke is invoked from Python. The usage of Pyke for logic programming in Python environment is done by programming logic statement and rules and executing them in the inference engine. The reason why Pyke is developed on top of Python is that Python is a good general purpose programming language that allows programming in a compact way. Pyke can be used for complicated decision making applications, diagnosis systems, control module, etc.

Pyke integrates Logic Programming into Python by providing knowledge engine that supports forward-chaining and backward-chaining inferencing. It introduces concepts of fact base, rule base and question base. Fact base represents all information about the system i.e. information that is true about the system or false about the system, and they are all called facts with respect to the syste. Rule base represents rules that govern the system. Rules in Pyke can be forward chaining and backward chaining rule. Through its inference engine, Pyke uses these rules to deduce new information about the system. Sometimes, questions may be asked to Pyke to prove something. And these questions form question base.

A fact in Pyke is a statement with several arguments. For example a sentence:

```
"A data controller must log all accesses to personal data."
```

is expressed in Pyke statements as:

```
access(ds), log(dc,ds)
```

These statement follows syntax proposed by Pyke which is:

```
statement ::= IDENTIFIER '.' IDENTIFIER '(' {argument,} ')'
```

In the example, family is the name of knowledge base. Son_of is the name of knowledge base. Three values “Bruce”, “Thomas”, and “Norma” are statement arguments.

A rule in Pyke is defined as:

```
if
    A
    B
    C
then
    D
    E
```

The rule shown above has meaning “if A, B, and C are true, then D and E are also true”.

There are two types of inferencing in Pyke:

- Forward chaining
  In this type, forward-chaining rules are processed. New facts are asserted based on the defined rules. Pyke finds rules whose if clause matches facts on the fact base. Each time of a match, it fires the rule which will add the facts in the then clause of that

corresponding rule to fact base. Newly added facts will fire other rules by matching their if clause. The process continues until there is no more match.

Following figure shows example of forward chaining rule:

```
obligation
    foreach
        source.access($ds)
    assert
        source.log($dc, $ds, ())
```

In the example, every time there is fact son_of in the fact base, Pyke inference engine will assert new fact father_son into fact base.

- Backward chaining
  Backward chaining rules are processed when Pyke is asked a question i.e. to prove something. In backward chaining, Pyke find rules by matching their then clause with the question. In every match, it tries to prove if all statements in the if clause holds. If all statements can be proven, then the rule succeeds. If not, then Pyke tries to find another rule whose then clause matches the question, and so on.

  Following figure shows example of backward chaining rule:

```
obligation
    use log($dc, $ds, ())
    when
        source.access($ds)
```

In the example, asking question “is accessed logged?” is done by matching log with statement in the fact base, and try to prove if access for the associated relationship holds in the fact base.

Pyke has three kinds of source files:

- Knowledge Fact Base (KFB) files for fact bases
  These files have .kfb suffix. All facts are put in this file

- Knowledge Rule Base (KRB) files for rule bases
  These files have .krb suffix. All rules are put in this file

- Knowledge Question Base (KQB) files for question bases
  These files have .kqb suffix.

All the files are put in a directory structure. Pyke knowledge engine receives the directory information as argument and compile the source files within it.

3. Proposed Method & Implementation

In this section, methods for providing accountability in cloud service by evidence collection and processing through policy or obligation monitoring is presented. The explanation is about framework for evidence collection. One of the important sections in the framework is monitoring policy by which evidence can be collected about events that are not complied with the policy. A use case is selected to show how the proposed method works. Policy representation is used for the use case and all processes throughout the framework with respect to the use case. Sources of evidences are also discussed mentioning about how to collect sources of evidences for evidence collection framework. In addition to the method used, implementation of such method is also discussed in this section using some tools and techniques that are discussed in background section.

3.1. Working Steps in the Thesis

This section describes overall general steps of works for competing the goal of this thesis. Each step represents general question to be addressed in the work. The steps are depicted in following Figure 3.1.

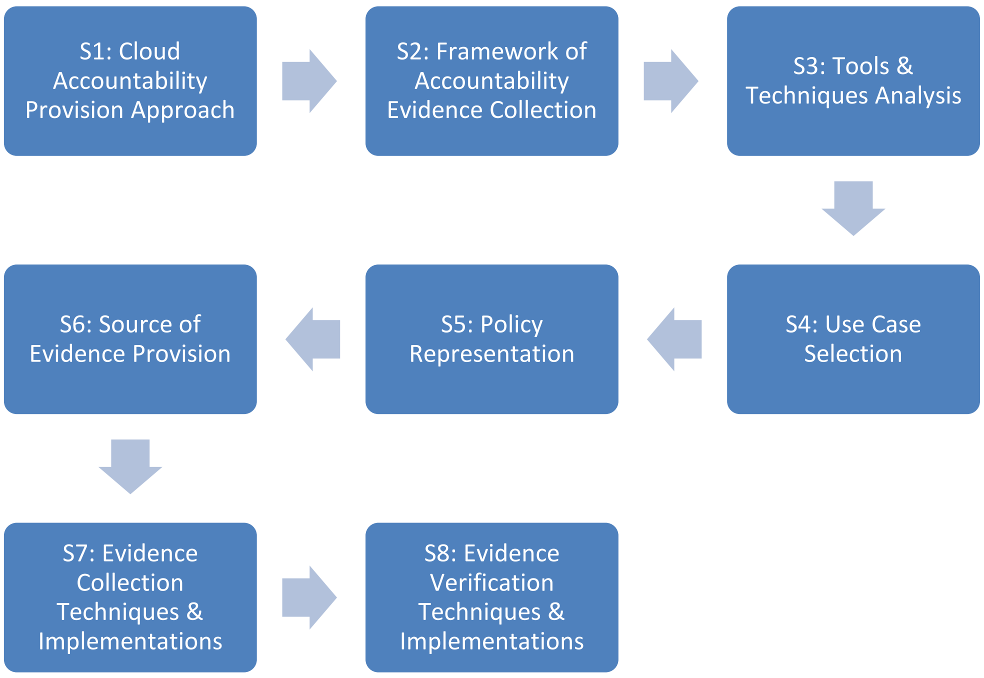
S1: Cloud Accountability Provision Approach
S2: Framework of Accountability Evidence Collection
S3: Tools & Techniques Analysis
S4: Use Case Selection
S5: Policy Representation
S6: Source of Evidence Provision
S7: Evidence Collection Techniques & Implementations
S8: Evidence Verification Techniques & Implementations

Figure 3.1. Working Steps

First step (S1) in the workflow is defining approach for providing accountability in cloud service. This includes study of existing researches on how accountability in cloud service is achieved. The question arises from the fact that putting service in the cloud where several providers are taking different roles in providing the service gives challenges on how to ensure that each provider complies with obligations that users or customers have set up with respect to processing their personal data. There must be way to measure providers' activities in processing the data and provide account on what it has done on the data. This question is tried to be answered in the first step. The result of study on this step will be the basis for next step on designing framework for providing accountability in cloud service.

Second step (S2) in the workflow is to define framework for providing accountability in cloud service. As the accountability is provided in the presence of evidence of policy compliance, the framework defined is on evidence collection with respect to the policy. Framework is defined as sequence of steps required to acquire evidence that can be used as basis for providing accountability. This framework is concrete basis for evidence collection. Each step is to be described in more detail accompanied with implementation to prove that it works.

Having defined general steps that need to be completed in order to acquire evidence, the next step (S3) is to study on existing tools and techniques on how to realize each step in the framework. This is to review best and most suitable approach for achieving the goal for each step and also for implementing the approach and seeing the result. Techniques studied are around processing source of evidence, policy monitoring, policy representation, digital evidence collection and processing, etc. Tools associated with the techniques are also studied.

With the framework for evidence collection for achieving accountability in cloud service already defined, the basis for answering the general question on this thesis is there. Next step is to dig more detail into each component in the framework and implement prototype of it. However, before jump into that step, it is also beneficial to look into a use case (S4) where this approach is typically applied to. All steps following are then developed with the use case solving aim in mind.

As the accountability in cloud service is achieved by acquiring evidence on policy compliance, representing policy is one of the main actions (S5). Customers or users define obligations to cloud providers regarding processing their personal data. These obligations are to be represented in formal format so that they are processable in purpose of detecting violations to them. This step studies on how to express the policy. Taking examples of policies from the use case selected and represents them in the selected policy representation is one of the exercise also in this step. This representation will be used in the next step which it to extract evidence in regards to policy compliance.

The next step (S6) is to study and review how to get source of evidence. Source of evidence represents system activities with which the policies are monitored. Some implementations may be needed in this step to prepare the source of evidence which later will be used in the evidence collection process.

One of the main parts in the work is collecting evidence from cloud system activities in purpose of compliance checking to policy defined by users or customers. In this step (S7), techniques used for collecting evidence are studied. Implementation of the techniques is then performed and also using the use case to show how the technique works.

Next step (S8) following collecting evidence is to solve problems on evidence verification. This step is to study how verification on the collected evidence should be done. Techniques and tools around evidence verification are studied. Based on that, implementation on the techniques is performed to show case that it can solve the problem using selected use case.

3.2. Use Case and Policies

In this thesis, in addition to proposing approaches and methods for providing accountability in general cloud service scenario, a use case of cloud service is also selected. This selection has the intention to get real example of cloud service and application of accountability provision as well. With this use case, all techniques proposed within the framework of accountability evidence collection will be tested with the use case. This gives insight on how the framework works in real example.

The use case is taken from [24]. It is about health care service in the cloud. This example is selected because it demonstrates cloud service concept where there are users that uses services provided by one or more service providers in cloud which involves processing of user personal data. With this setup, providers must follow rules that users have given at the first place before using the service. This introduces challenges as what have been discussed about cloud service which is to ensure that the cloud providers follow the policies or obligations and are accountable for what they are executing regarding users personal data.

In health care service in the cloud, user data generated by medical sensors are flowing to the cloud for processing. Medical sensors are embedded in elderly people to capture information about their life activities. The processing is regarding diagnosis of patients by the collection and processing of data from wearable sensors. The medical data is then possibly exchanged between patients, their relatives, hospital, and cloud providers. Patients in this case are data subjects who own personal data processed in the cloud. Hospital is data controller which holds responsibility for health care services that process patient’s data. It determines purpose and means of processing with respect to patient’s personal data. Several cloud providers may be involved for data collection and processing. In this case, chain of service delivery is created with one of the cloud service providers be the primary service provider which is the entity that provide the total service to users. The service is available for accessing in the cloud by patients, their relatives, and hospital. The health care service in cloud explained is depicted in following Figure 3.2 (adapted from [24]).

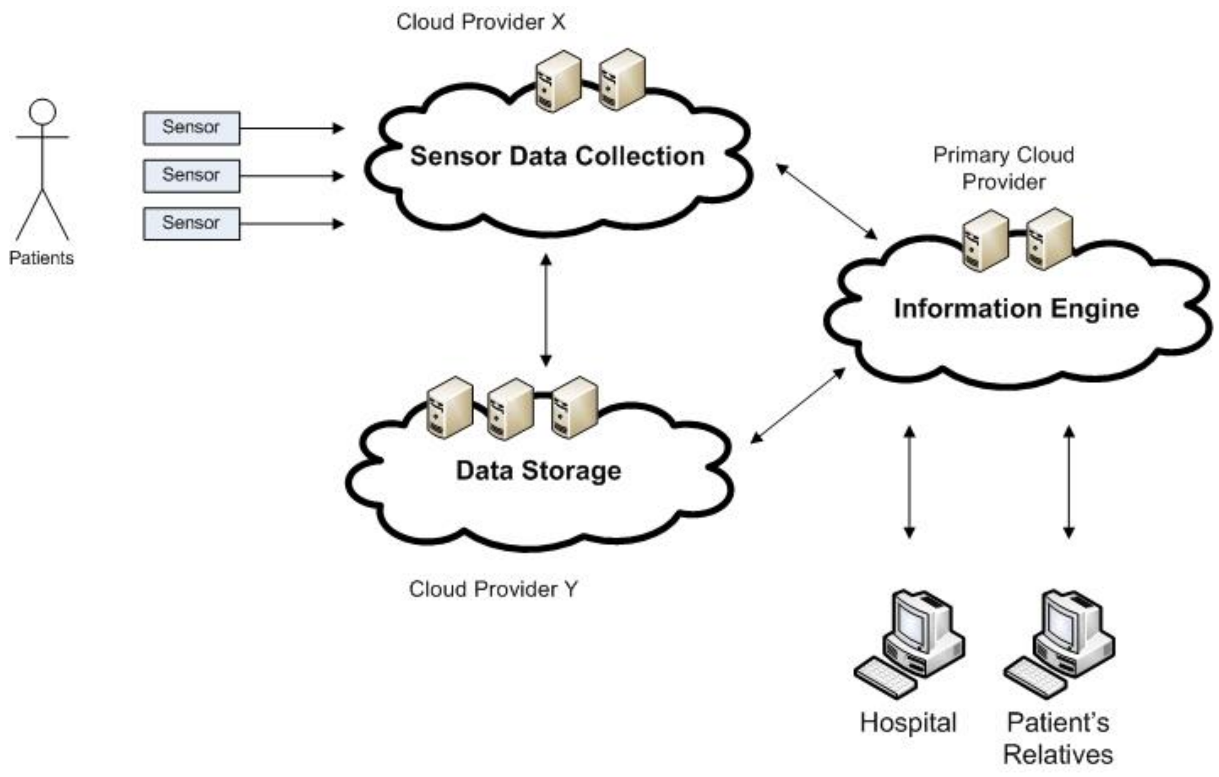

Figure 3.2. Use Case: Healthcare Service in Cloud

With the flow of sensitive personal data along the cloud service chain, there is a set of obligations that cloud service providers and other entities in the system need to follow related to generation, processing, and flow. For this thesis, set of obligations for the use case is selected especially that supports evidence collection. Later on this work, these obligations are to be used in the proposed framework of accountability evidence collection to show that the application on real case. The obligations that need to be followed in the healthcare cloud service are [24]:

1. O1: As a data controller, the hospital needs to provide a policy on what data is collected and for what purposes

2. O2: As a data controller, the hospital must ask the data subjects (patients) explicit consent for collecting and processing personal data

3. O3: As joint data controllers, the relatives must ask the data subjects (patients) explicit consent for collecting and processing personal data

4. O4: As a data controller, the hospital must, upon request, provide evidence to the data subjects on their personal data processing activities

5. O5: As a data processor, the primary service provider must log all access to personal data

6. O6: As a data processor, the primary service provider must, upon request, provide evidence to the data controller (hospital) on its personal data processing activities

7. O7: As a data processor, the primary service provider must, upon request, provide evidence to the data controller (hospital) on the correct and timely deletion of personal data

The work in [24] studied about policy representation framework in cloud environment. It studied several existing policy languages and reviewed their suitability for accountability representation. It later found that none of the existing languages could satisfy accountability requirements that policies should express. Therefore, it proposes new policy representation framework specifically developed to express policy in cloud environment. It is called A-PPL (Accountable-PPL) which is developed based on PPL (PrimeLife Policy Language). A-PPL expresses obligations for cloud environment in machine readable representation. For that reason, all obligations set up in the use case will be represented using this language.

3.3. Accountability Evidence Collection

In the effort for providing accountability in cloud service, policy compliance checking is performed on system activities and produces evidences that can act as proofs that an entity had performed events and whether they are complied with the policies or obligations or not. This section describes how the evidences are collected with respect to policies or obligations defined. The processes are defined in framework for accountability evidence collection.

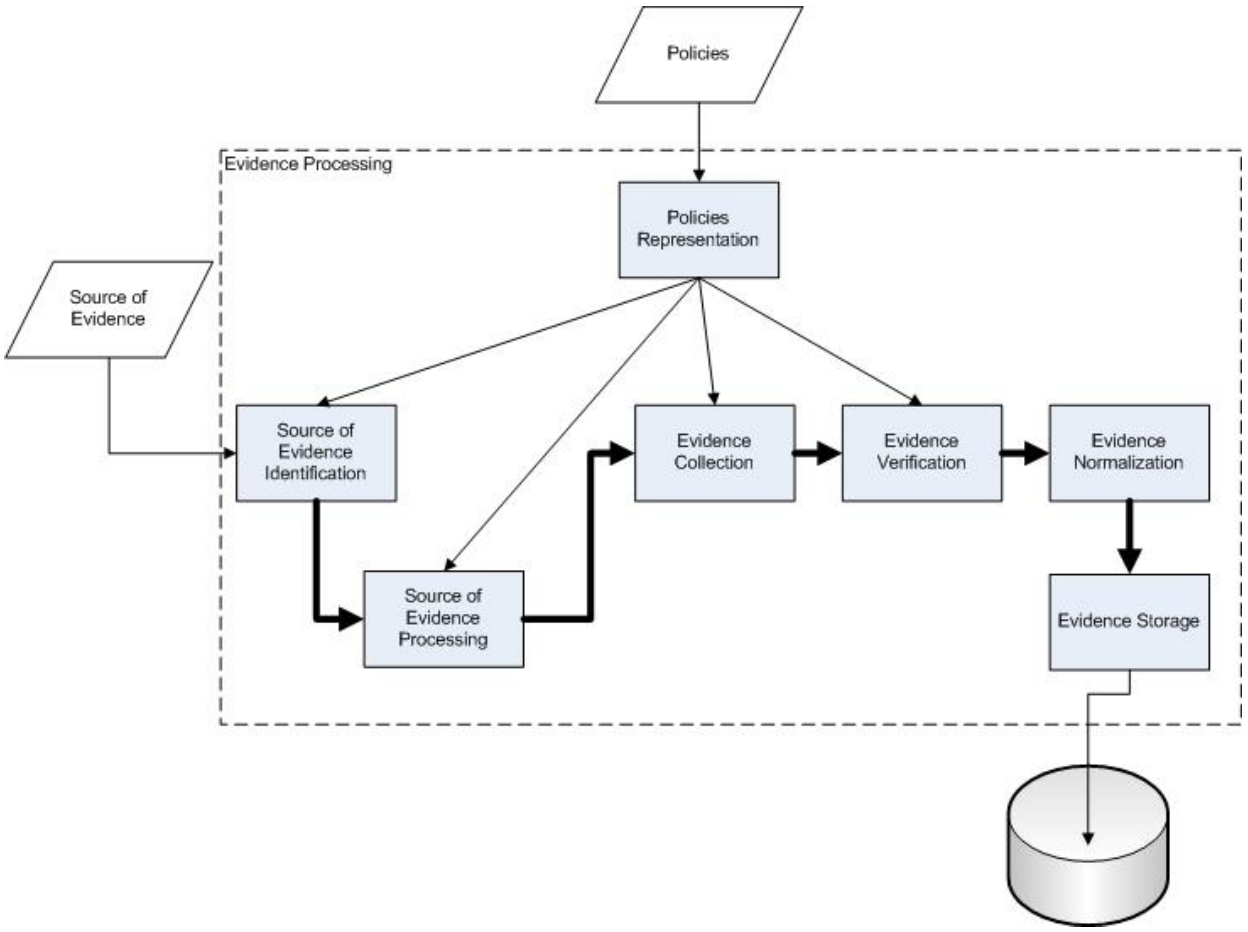

Figure 3.3. Proposed Framework of Evidence Collection

Figure 3.3. shows the framework for accountability evidence collection.

The framework receives defined policies and source of evidence as inputs. Policies are defined in human readable language. This definition is then translated into A-PPL machine readable format by A-PPL engine as proposed in [24]. As will be discussed in section 3.9. Evidence Collection with MonPoly and 3.14. Evidence Collection with Pyke, policy representation is needed to take the policy expressed in A-PPL format and translate them into format that collection process can handle. In another part, source of evidence may be logs that capture cloud system activities. This is the basis for evidence collection in terms of policy compliance.

Identification of source of evidence includes effort on picking suitable logs for evidence processing i.e. logs that contain information related to policy compliance checking in purpose of producing evidence on any violating events occur in the system with regards to the policies or obligations that have been set up. Identification process is influenced by the policies that govern the whole cloud system.

As source of evidence has been identified with input information from policies, the next step is to prepare the source of evidence, in this case is typically logs (as explained in section 3.5. Source of Evidence). Output from this preparation is ready for evidence collection process. Processing source of evidence includes activities on collecting, combining, formatting, and normalizing log data for next collection process with input information about policies. These steps are needed because source of evidence may come from several sources with different format. Before it can be processed, these information need to be uniformed in terms of format so that comparing and correlating are possible.

The output from source of evidence processing is ready for evidence collection process. With information about source of evidence and policy representation, evidence collection is performed to detect any violations to policies done by some activities or events recorded in the source. The result of this collection process is the potential candidate for evidence which serves as proof of what have been done by entity pointed out in the evidence with respect to accountability provision in the cloud service and that the entity is accountable for its action.

Output from collection process which is candidate of evidence needs to be verified. It is necessary to examine whether to accept the evidences, to reject them, or to revoke them. Verification seeks information to test validity of the collected evidence. It involves clarifying the evidence with existing information and possibly searching for additional information to strengthen the evidence validity.

Evidence normalization is performed to take the collected evidence into format that is processable in next further step. The format should contain enough information so that other entities will be able to analyse further in relation to the event and the system itself.

At the last stage, after evidence is collected, storing or preservation of evidence is performed. [5] mentioned that preservation of evidence is one of the process in digital forensic where evidence has been collected. Integrity of the evidence must be maintained and be safeguarded before it is actually used in legal proceeding. Study on how to store the evidence including implementation of it is out of the scope for this thesis. It will be instead stored directly. The introduction of this step is just for giving complete chain of process in the framework of accountability evidence collection.

3.4. Policy Representation in A-PPL

In section 3.2. Use case and policies, several policies are introduced for selected use case i.e. healthcare service in cloud. These policies are to be followed by all entities in the service chain i.e. data subject (patient), data controller (hospital), and data processor (cloud service provider). Evidence is collected from monitoring those policies on system activities which are logged in some system logs. These accountability policies are expressed in A-PPL which is the accountability policy representation framework proposed in [24].

A-PPL is using XML format in representing accountability policy. It is translated to machine understandable format from AAL by involving semantics which is based on LTL. Therefore, processing the policy in A-PPL format will involve temporal logic processing. As explained in section 2.6. Policy Representation (PPL, AAL, A-PPL), A-PPL contains several important sections such as role, trigger, and action. Representation of policy for the selected use case will focus on these sections.

Following list shows A-PPL representation of each policy defined in section 3.2. Use case and policies (adapted from [24])

1. As a data controller, the hospital needs to provide a policy on what data is collected and for what purposes

   This policy can be fulfilled by creating processing info element before hospital collects patient data, that contain information about what data will be collected and for what purpose.

```
<Obligation>
      <Identification>O1</Identification>
      <TriggerSet>
            <TriggerOnProcessingInfo>
                  <MaxDelay>
                        <Duration>5M</Duration>
                  </MaxDelay>
            </TriggerOnProcessingInfo>
      </TriggerSet>
      <ActionSet>
            <ActionCollect>
                  <Subject>
                     <AttributeValue DataType="String">Hospital</AttributeValue>
                  </Subject>
            </ActionCollect>
      </ActionSet>
</Obligation>
```

2. As a data controller, the hospital must ask the data subjects (patients) explicit consent for collecting and processing personal data

```
<Obligation>
     <Identification>O2</Identification>
     <TriggerSet>
           <TriggerOnPreferencesUpdate>
                 <MaxDelay>
                       <Duration>5M</Duration>
                 </MaxDelay>
                 <Subject>
                       <AttributeValue DataType="String">Hospital</AttributeValue>
                 </Subject>
           </TriggerOnPreferencesUpdate>
     </TriggerSet>
     <ActionSet>
           <ActionCollect>
                 <Subject>
                       <AttributeValue DataType="String">Hospital</AttributeValue>
                 </Subject>
           </ActionCollect>
           <ActionProcess>
                 <Subject>
                       <AttributeValue DataType="String">Hospital</AttributeValue>
                 </Subject>
           </ActionProcess>
     </ActionSet>
</Obligation>
```

`TriggerOnPreferenceUpdate` can be used to inform hospital that data subject has given consent for processing his personal data. Collection and processing personal data by hospital then can be executed based on this consent.

3. As joint data controllers, the relatives must ask the data subjects (patients) explicit consent for collecting and processing personal data

```
<Obligation>
    <Identification>O3</Identification>
    <TriggerSet>
         <TriggerOnPreferencesUpdate>
              <MaxDelay>
                   <Duration>5M</Duration>
              </MaxDelay>
              <Subject>
                    <AttributeValue DataType="String">Hospital</AttributeValue>
              </Subject>
          </TriggerOnPreferencesUpdate>
     </TriggerSet>
     <ActionSet>
          <ActionCollect>
              <Subject>
                 <AttributeValue DataType="String">Hospital</AttributeValue>
```

```
               </Subject>
           </ActionCollect>
           <ActionProcess>
               <Subject>
                  <AttributeValue DataType="String">Hospital</AttributeValue>
               </Subject>
          </ActionProcess>
      </ActionSet>
</Obligation>
```

4. As a data controller, the hospital must, upon request, provide evidence to the data subjects on their personal data processing activities

```
   <Obligation>
            <Identification>04</Identification>
            <TriggerSet>
                  <TriggerPersonalDataAccessedForPurpose>
                        <MaxDelay>
                              <Duration>5M</Duration>
                        </MaxDelay>
                  </TriggerPersonalDataAccessedForPurpose>
                  <TriggerOnPersonalDataDeleted>
                        <MaxDelay>
                              <Duration>5M</Duration>
                        </MaxDelay>
                  </TriggerOnPersonalDataDeleted>
                  <TriggerOnPersonalDataSent>
                        <MaxDelay>
                              <Duration>5M</Duration>
                        </MaxDelay>
                  </TriggerOnPersonalDataSent>
            </TriggerSet>
            <ActionSet>
                  <ActionLog>
                        <Action>
                              <AttributeValue DataType="String">Access, Send,
                                    Delete</AttributeValue>
                        </Action>
                        <Subject>
                              <AttributeValue
                                    DataType="String">Hospital</AttributeValue>
                        </Subject>
                  </ActionLog>
            </ActionSet>
      </Obligation>
```

Hospital needs to keep track of all actions performed in the personal data, so that audit is possible for checking conformance of data usage. ActionLog is used to track personal data processing activities in cloud. Therefore, whenever, for example data is accessed, data controller logs it in the system within some defined time.

```
      <Obligation>
                  <Identification>04</Identification>
                  <TriggerSet>
                        <TriggerOnEvidenceRequestReceived>
```

```
                              <MaxDelay>
                                    <Duration>5M</Duration>
                              </MaxDelay>
                              <Subject>
                                    <AttributeValue
                                     DataType="String">Hospital</AttributeValue>
                              </Subject>
                        </TriggerOnEvidenceRequestReceived>
                  </TriggerSet>
                  <ActionSet>
                        <ActionEvidenceCollection>
                              <Subject>
                                    <AttributeValue
                                     DataType="String">Hospital</AttributeValue>
                              </Subject>
                        </ActionEvidenceCollection>
                  </ActionSet>
      </Obligation>
```

Logging enables hospital to keep track of all processing activities on personal data. When there is a request to provide evidence on data processing activities, hospital can collect this log and provide the information to the requestor.

5. As a data processor, the primary service provider must log all access to personal data

```
   <Obligation>
            <Identification>05</Identification>
            <TriggerSet>
                  <TriggerPersonalDataAccessedForPurpose>
                        <MaxDelay>
                              <Duration>5M</Duration>
                        </MaxDelay>
                  </TriggerPersonalDataAccessedForPurpose>
            </TriggerSet>
            <ActionSet>
                  <ActionLog>
                        <Action>
                              <AttributeValue
                                   DataType="String">Access</AttributeValue>
                        </Action>
                        <Subject>
                              <AttributeValue DataType="String">Data
                                   Processor</AttributeValue>
                        </Subject>
                  </ActionLog>
            </ActionSet>
      </Obligation>
```

ActionLog provides way for data processor to log all actions on data subject’s personal data. It is triggered when there is access to the personal data.

6. As a data processor, the primary service provider must, upon request, provide evidence to the data controller (hospital) on its personal data processing activities

```
    <Obligation>
             <Identification>06</Identification>
             <TriggerSet>
                   <TriggerPersonalDataAccessedForPurpose>
                         <MaxDelay>
                               <Duration>5M</Duration>
                         </MaxDelay>
                   </TriggerPersonalDataAccessedForPurpose>
                   <TriggerOnPersonalDataDeleted>
                         <MaxDelay>
                               <Duration>5M</Duration>
                         </MaxDelay>
                   </TriggerOnPersonalDataDeleted>
                   <TriggerOnPersonalDataSent>
                         <MaxDelay>
                               <Duration>5M</Duration>
                         </MaxDelay>
                   </TriggerOnPersonalDataSent>
             </TriggerSet>
             <ActionSet>
                   <ActionLog>
                         <Action>
                               <AttributeValue DataType="String">Access, Send,
                                    Delete</AttributeValue>
                         </Action>
                         <Subject>
                               <AttributeValue DataType="String">Data
                                    Processor</AttributeValue>
                         </Subject>
                   </ActionLog>
             </ActionSet>
       </Obligation>
```

The same as data controller, data processor can use ActionLog to keep track on all processing activities of personal data. When, there is a request on providing evidence, data processor can collect information from this log.

```
       <Obligation>
                   <Identification>06</Identification>
                   <TriggerSet>
                         <TriggerOnEvidenceRequestReceived>
                               <MaxDelay>
                                     <Duration>5M</Duration>
                               </MaxDelay>
                               <Subject>
                                     <AttributeValue DataType="String">Data
                                         Processor</AttributeValue>
                               </Subject>
                         </TriggerOnEvidenceRequestReceived>
                   </TriggerSet>
                   <ActionSet>
                         <ActionEvidenceCollection>
                               <Subject>
                                     <AttributeValue DataType="String">Data
                                          Processor</AttributeValue>
                               </Subject>
                         </ActionEvidenceCollection>
                   </ActionSet>
       </Obligation>
```

7. As a data processor, the primary service provider must, upon request, provide evidence to the data controller (hospital) on the correct and timely deletion of personal data

```
    <Obligation>
            <Identification>07</Identification>
            <TriggerSet>
                  <TriggerOnPersonalDataDeleted>
                        <MaxDelay>
                              <Duration>5M</Duration>
                        </MaxDelay>
                  </TriggerOnPersonalDataDeleted>
            </TriggerSet>
            <ActionSet>
                  <ActionNotify>
                  </ActionNotify>
                  <ActionLog>
                        <Action>
                              <AttributeValue
                                   DataType="String">Delete</AttributeValue>
                        </Action>
                        <Subject>
                              <AttributeValue DataType="String">Data
                                   Processor</AttributeValue>
                        </Subject>
                  </ActionLog>
            </ActionSet>
       </Obligation>
```

When there is action on deletion of data subject's personal data, data processor logs it. This information can be used when there is request to provide evidence on correct deletion of personal data. In addition to logging, data processor also notifies data controller when there is such action.

3.5. Source of Evidences

For implementing proposed method in providing accountability in cloud service i.e. processes defined in framework of accountability evidence collection, selected use case is used. Explanation of the use case is in section 3.2. Use Case and Policies. In relation to that, source of evidence for this use case need to be identified.

For providing data to be used in the implementation and testing of all processes defined in the framework of evidence collection, a test environment needs to be created. More detail explanation about the test environment is contained in section 3.6. Test System Settings. This step of creating test environment is needed because there is no available data to be used as source of evidence in relation to the use case selected i.e. healthcare service in cloud. From this test environment, source of evidence is collected.

Based on explanation given in background about source of evidence, most of the sources are system logs. In the test environment, which is set up using VMs (Virtual Machine) on Linux system, the sources of evidence are Linux system logs. These logs are the logs that capture internal system events and also external system events. Internal system events related to events that occur within a VM host. While external system events are events that occur between a VM host and its outside environment for example with another VM hosts.

Since the test environment for the health care service in cloud use case is set up using VMs environment in Linux system, some of Linux system logs are used as source of evidence. The logs as source of evidence are identified based on the information about policies or obligations defined for the use case and also based on events that occur in the VMs system which depicts activities within an entity or between entities in the healthcare service settings. For example, in healthcare service in cloud case, there must be event that is triggered when data subject registers to the cloud service or accesses his data in the cloud. Another example is when data controller (i.e. hospital) is using the cloud service to process data subject personal data or when data controller accesses data subject personal data. All possible typical events or activities in such setting (i.e. healthcare service) are identified for the purpose of identifying which Linux system logs that contain relevant information based on the representing events in the VMs environment.

As will be explained in section 3.7. Event Generation, to represent the use case activities, several events in VMs are generated, each of them mimics the activities in the use case. There are generally 3 types of events which are internal system events, communication systems, and file sharing events. Based on that, Linux system logs [32] that are identified as source of evidence for the use case are

- /var/log/messages: contains information about global system messages, including messages that are logged during system startup. Several other things that are logged in this log are mail, cron, daemon, kern, auth, etc. This log is used to capture information about events related to internal event within each entity in the healthcare service case.

- /var/log/auth.log: contains information about system authorization, including user logins and authentication mechanism that were used. This log is used to capture information about events like sending data between entities in the healthcare service case for example when data subject (i.e. patient) personal data is sent to cloud.

- /var/log/secure: contains information related to authentication and authorization privileges. This log is used by ssh to log all of its messages. It can be used to capture information about events like sending data between entities in the healthcare service case.

- /var/log/mail.log: contains log information from mail server that running on the system. It records all activities about sending mail. This log is used to capture information about events like communication activities between entities in the healthcare service case for example when data controller (i.e. hospital) asks for data subject (i.e. patient) explicit consent for collecting and processing his data.

- /var/log/httpd: contains web server access logs. This is used to capture information related to events like data subjects accessing cloud service.

Figure 3.4. shows relationships between policies, system logs, VMs environment, and source of evidence. It shows how the source of evidence is identified based on the test environment that is set up for the purpose of representing healthcare service in cloud use case. VMs test environment contains several system logs that capture events occur in the system. Some of the logs are identified as source of evidence based on policies defined for the use case and events that occur in the system which represent typical events occur in the use case.

3.6. Test System Settings

Test environment to represent healthcare service in cloud is set up. The purpose of this is to have data as source of evidence to be processed using proposed method on selected use case and related defined policies or obligations. The environment is set up using VMs (Virtual Machine) in Linux. The source of evidence will be collected from system logs that are generated by all events occurred in the environment replica. There will be simplification on test environment compared to original healthcare service in cloud. This is just to get the essence of important elements going on in the system and use that information to feed in as data for proposed method for providing accountability in cloud service.

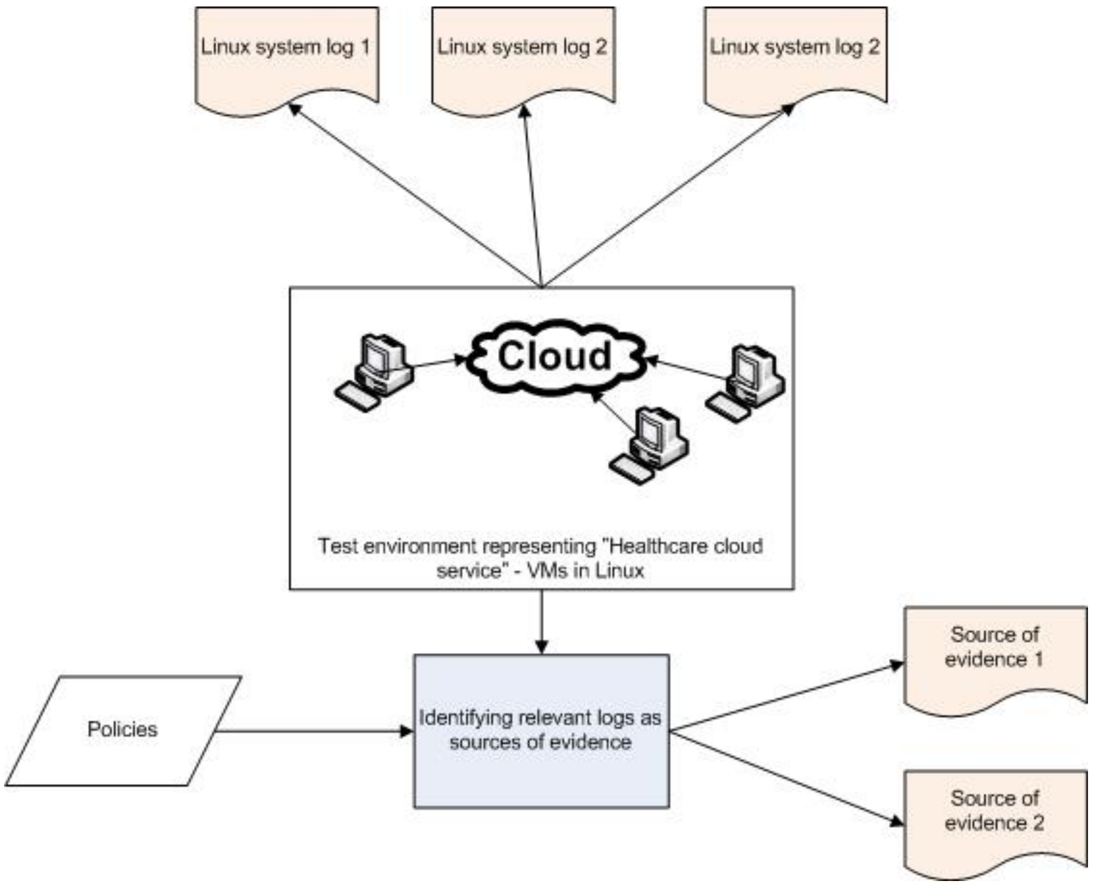
Linux system log 1
Linux system log 2
Linux system log 2
Cloud
Test environment representing "Healthcare cloud service" - VMs in Linux
Policies
Identifying relevant logs as sources of evidence
Source of evidence 1
Source of evidence 2

Figure 3.4. Source of Evidence for Test System

There are four VMs created, each of them is acting as Data Subject (patient), Data Controller (hospital), Data Processing (Cloud Service Provider), and Other Entity (relative of the patient or other parties) respectively. Communication between VMs is considered as between actors in the healthcare service in cloud use case.

Activities between actors that are related to the use case are of interest. Typical activities are processing, data transfer with regards to collection, and communication. For running the system automatically, an automatic event generator is developed that takes as input the information about selected use case and related policies. Events related to the use case will be generated as if they occur in the real setting i.e. healthcare service in cloud. More detail about event generation is explained in section 3.7. Event Generation. Events that are generated in the processing activities are for example processing personal data, etc. Data transfer activities are for example uploading data to cloud, forwarding data, etc. Communication activities are for example asking consent, conveying processing information, accessing data, etc. All the activities are generated as events in the VMs environment. Information about all events occurred will be used as source of evidence for proposed processing evidence framework.

Figure 3.5. shows the VMs environment created for the purpose of collecting source of evidence for healthcare service in cloud use case.

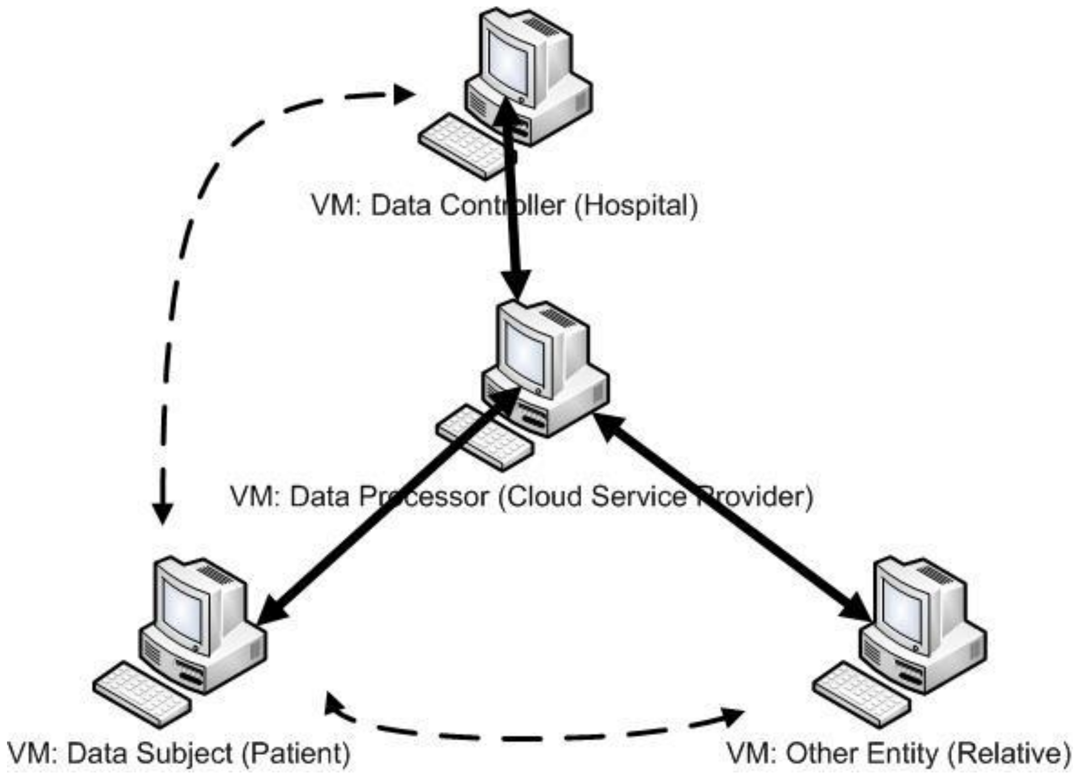
VM: Data Controller (Hospital)
VM: Data Processor (Cloud Service Provider)
VM: Data Subject (Patient)
VM: Other Entity (Relative)

Figure 3.5. Test System Environment

There are communication lines between VM: Data Controller, VM: Data Subject, VM: Relative with VM: Data Processor which show the access lines between patient, hospital, relative with data processor in the healthcare service in cloud. Similarly, there is communication line between hospital and patient and also between patient and relative.

All the events are logged in the system and the logs are collected as source of evidences. Identification of Linux logs that are relevant for processing with regards to selected use case and its defined policies is explained in section 3.5. Source of Evidence. Information about events is taken from system event log, network communication log, data transfer log, and access log of associated VM.

3.7. Event Generation

In healthcare service in cloud use case, patient personal data is collected by hospital by using infrastructure of cloud service provider. Before doing so, hospital needs to ask explicit consent from patient. In addition to that, hospital must be able to, upon request, provide information about processing activities of patient personal data. In order to provide that, any actions on the personal data need to be logged. Cloud service provider, on the other hand, also must be able to, upon request, provide information about processing activities on patient personal data. Hospital must also inform patient on the purpose of processing his personal data. In addition to that, patient's relative can upload patient's data and access them based on patient's consent.

Based on the description of activities within the use case i.e. healthcare service, some events are identified and are generated in the test environment i.e. VMs simulating the cloud service. Events that are to be generated within the test environment are:

- Upload data
  This event simulates event when patient personal data from wearable sensors are uploaded into cloud.

- Access data
  This event simulates event when patient personal data stored in the cloud is accessed by an entity in scenario.

- Delete data
  This event simulates event when patient personal data is deleted from the cloud.

- Process data
  This event simulates processing activity regarding patient personal data.

- Ask consent
  This event simulates asking consent event directed to patient as data subject.

- Give consent
  This event simulates event when patient as data subject gives explicit consent to data controller for processing their personal data.

- Give processing info
  This event simulates event when data controller conveys information to data subject about processing activities of their personal data in cloud.

- Request info
  This event simulates event when patient as data subject request for information about processing activities of their personal data.

- Give info
  This event simulates event when data controller gives processing personal data information requested by data subject.

- Log action
  This events simulates logging event occur in the cloud system.

An event generator is developed to automatically simulate the activities mentioned in healthcare service use case. It is also in order to have better scalability in generating events for providing data for processing. Figure 3.6. shows architecture of the event generator. The generator is built based on information that the system, in which events are to be generated, is VMs system in Linux. Therefore, all events generated are using commands related to VM commands in Linux. Main generation file takes as input the information about event collection and policies. Event collection is a set of events that have been identified for the healthcare service use case and to be chosen for creation in the system. Event collection is configurable so that the generator can be used to generate event on other use case with different set of events. Policies info is all information related to policies that is needed in order to generate the event. The information includes which policies to follow and which are not to. When the indicator indicates to follow a policy, then necessary events are to be generated in the system. This is to give control to user on how the generator generates events that later the data can be used for evidence processing in relation to the policy.

Looking at the definition of a policy, it generally has two parts which are trigger and action. Generation of events takes this information into account. Trigger events are considered predecessor which occur before another event. Action events are, on the other hand, considered accessor which occur after another event i.e. event that trigger it to occur. Therefore, there will be 2 types of generation i.e. generating predecessor and generating accessor. This generation is controlled by main generation based on policies info and event collection. Each generating predecessor and generating accessor will call associated generator scripts in target VM by giving them necessary input parameter. In each VM, there are 3 types of generator script i.e. generating internal process, generating sending process, and generating communication process. These generator scripts are called by generating predecessor and accessor based on policies info and event collection info. Main generator will have information about available VMs by running a script to collect info about VMs. This way, the event generator is even configurable in terms of VMs to simulate a scenario. The information is used to call associated script in each VM.

Policy indicator gives control on how the events are generated with respect to policies defined for the use case whether the policies are to be followed or not. With policy indicator set to true (i.e. value = 1), all related events for that policy are generated. However, when policy indicator is set to false (i.e. value = 0), not all related events for that policy are generated, but pseudo-randomly generated.

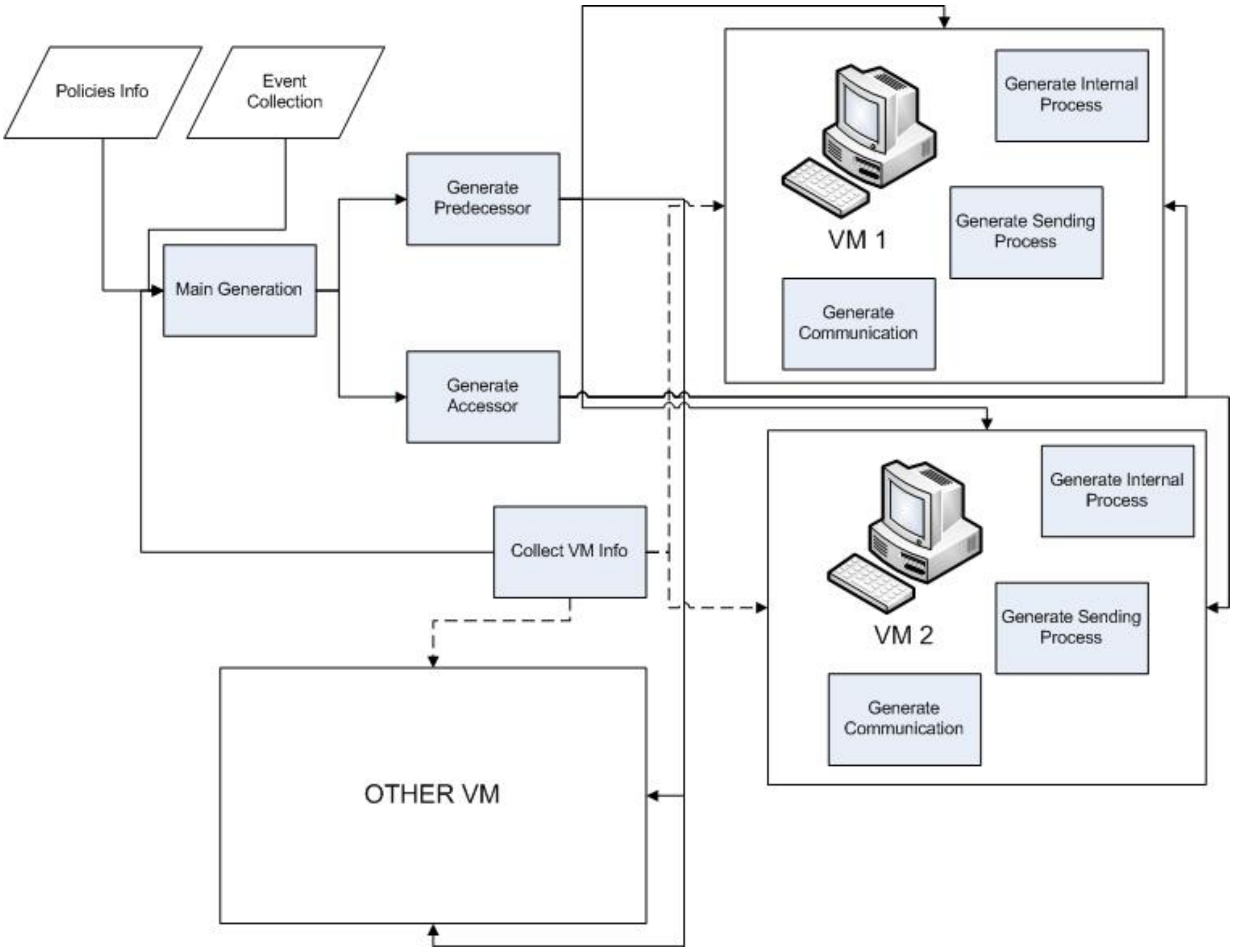

Figure 3.6. Event Generation Architecture

Following Figure 3.7. shows pseudocode for the event generator.

```
Setup EventCollection
Setup PolicyInfo
Collect VMs Info

counter <- expected_iteration

While number_of_iteration < counter
   Randomly select an event X from EventCollection

   Generate event X on target VM
```

```
        Generate all required events for event X on target VMs
        Read policies related to X

        For each policy Y for X
              If policyIndicator = 1
                    Generate predecessor events for policy Y and event X
                    Generate all required events for each predecessor event

                    Generate accessor events for policy Y and event X
                    Generate all required events for each accessor event

              ElseIf policyIndicator = 0
                    Pseudo-randomly create predecessor or accessor event for
                    policy Y and event X
              EndIf
        EndFor
EndWhile
```

Figure 3.7. Event Generation Pseudocode

Generally, each iteration in the event generation contains processes as shown in following Figure 3.8.

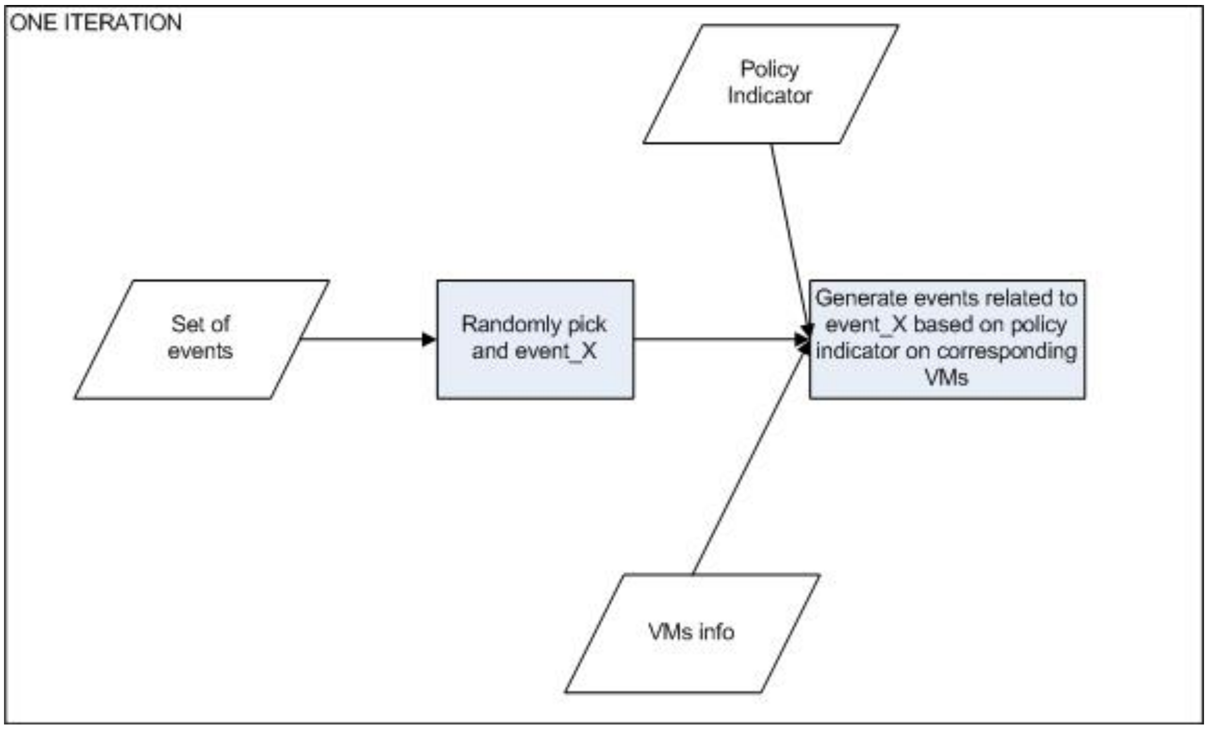
ONE ITERATION
Policy Indicator
Set of events
Randomly pick and event_X
Generate events related to event_X based on policy indicator on corresponding VMs
VMs info

Figure 3.8. Event Generation Iteration

Event generation is an iterative process. Inputs are set of events, VMs info, and policy indicator. Set of events contains all events that are going to be generated on the environment prototype. As the events are generated in related VMs, information about VMs are needed to execute those events. Policy indicator tells which policies are to be followed and which are not. This way, data for negative testing (where one/some policies are not followed and must be detected by the framework and generate the evidences) can be generated. All the inputs, set of events, VMs info, and policy indicator are configurable. In one iteration, an Event_X is randomly picked from the set of events and is generated in VM. Then, events related to Event_X are generated based on the information on the policy indicator on corresponding VMs.

3.8. Data Pre-processing

Logs are one of the sources of evidences used in the selected use case in this Thesis. These logs are collected from system logs of VM used to simulate healthcare service in cloud use case. Pre-processing prepares the logs to be used in evidence collection tool. Figure 3.9. shows the overall pre-processing of logs as source of evidence starting from collecting single log in each VM to processable source of evidences.

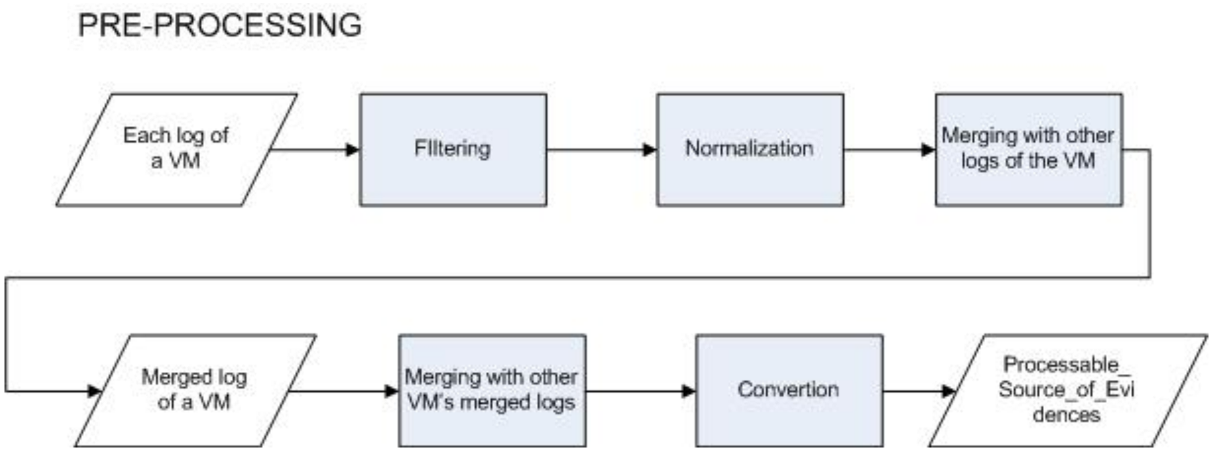
PRE-PROCESSING
Each log of a VM
FIltering
Normalization
Merging with other logs of the VM
Merged log of a VM
Merging with other VM's merged logs
Convertion
Processable_ Source_of_Evi dences

Figure 3.9. Source of Evidence Processing

Each VM represents each entity in the healthcare service in cloud i.e. data subject (patient), data controller (hospital), and data processor (cloud service provider). Each event of the entity related to policy compliance checking is collected from log. Several logs are involved in this case as explained in section 3.5. Source of Evidences. The pre-processing starts from single log from each VM. Not all information in the log is collected, but only related log information to the use case and policy compliance checking is collected. Therefore, filtering is needed in place for this purpose. As log entry of different logs may have different format, normalization is needed to have universal format within the framework of evidence collection. Normalization is performed on each filtered log on each related VM. Normalization essentially prepares the log entry with following information:

- Who
  Information about user or identifiable entity that is associated with the event being tracked or monitored

- What

  Action within event that is being monitored. This gives information on what actually happened and logged in the entry.

- When

  Information about time of when the event occurred.

- Why

  Additional information about event that may be beneficial for processing.

With this guideline, normalization produces universal format for all logs so that they are comparable and processable. The source of evidence for accountability evidence collection process should contain integrated information about activities happened in the cloud service. As each VM acts as an entity in the cloud service, it means that merging logs as source of evidence from VMs is needed. The last step in the pre-processing is to prepare the log in MonPoly standard format of log. This step is completed in convertion substep.

Figure 3.10. shows more detail on the filtering process of logs collected from each VM.

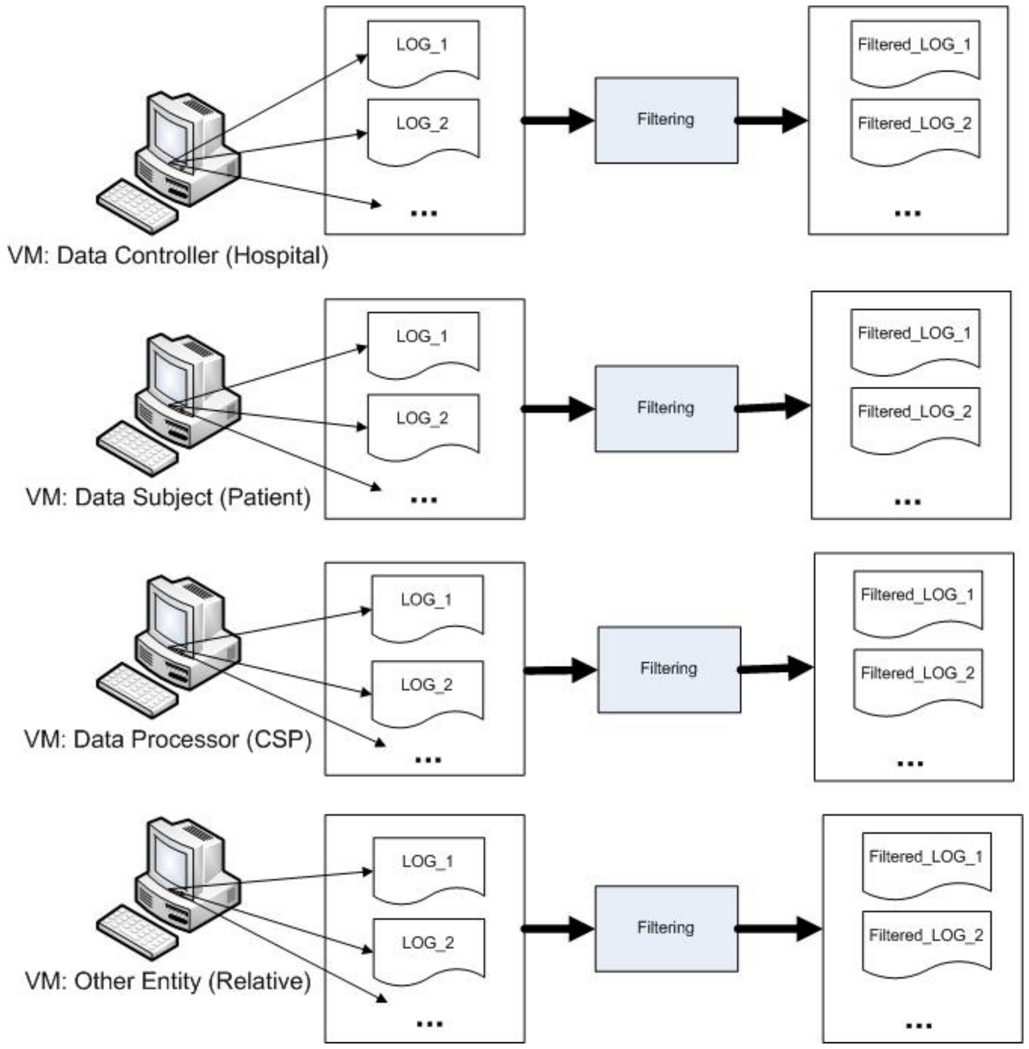
LOG_1
LOG_2
...
Filtering
Filtered_LOG_1
Filtered_LOG_2
VM: Data Controller (Hospital)
VM: Data Subject (Patient)
VM: Data Processor (CSP)
VM: Other Entity (Relative)

Figure 3.10. Filtering on Source of Evidence Processing

Figure 3.11. shows the normalization and merging processes performed in each VM. Normalization is performed on each log and then the results are combined together to produce a combined log for each VM.

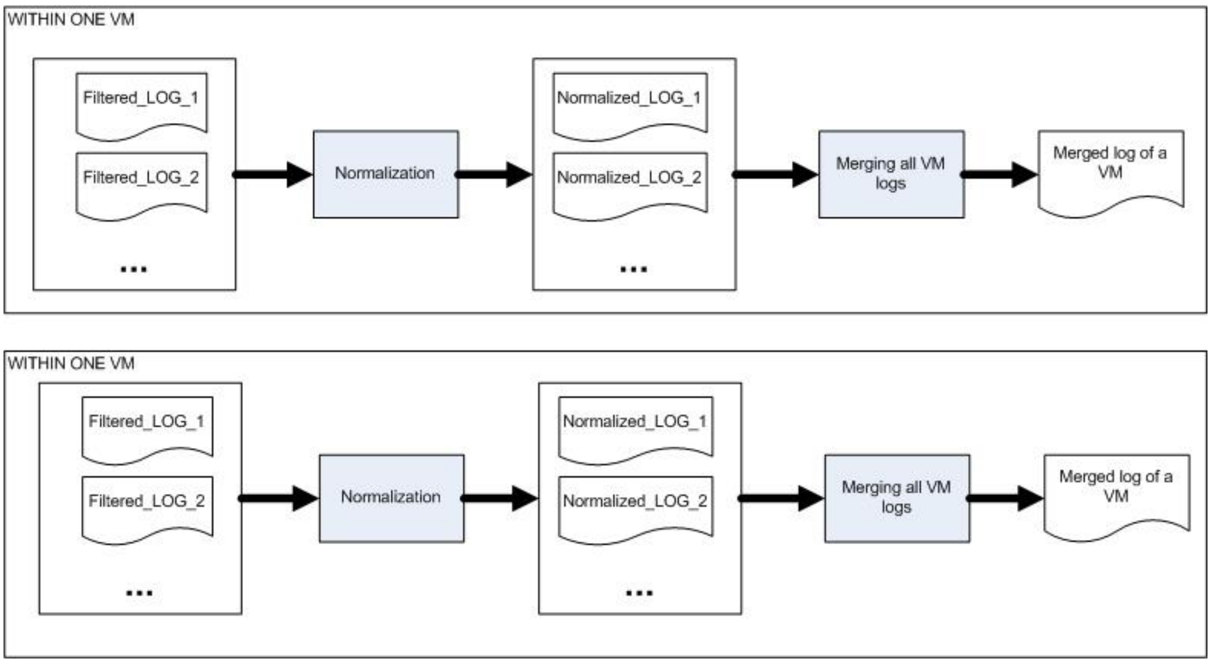

Figure 3.11. Source of Evidence Normalization

Figure 3.12. shows where the convertion process is in the overall processing steps. This convertion produces source of evidence that is processable in evidence collection tool.

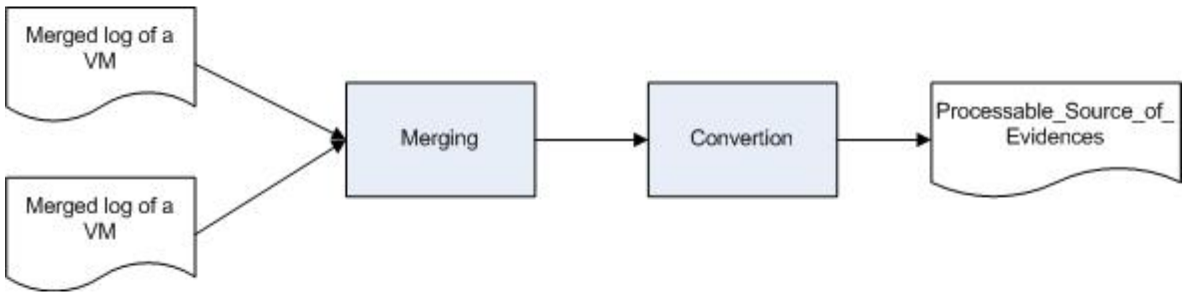

Figure 3.12. Source of Evidence Collection

3.9. Evidence Collection with MonPoly

Evidence collection process is the process following source of evidence identification and processing in the framework of accountability evidence collection. This process is to collect evidence from source of evidence with respect to policy compliance. With the source of evidence containing information about system events, collection process analyses if all the events complied with policies defined for the use case and collect evidences on any violations to policies.

Policies are defined in human understandable format which later by A-PPL engine are translated into machine understandable format A-PPL. Policy generally has two parts which are trigger and action. If trigger event occurs, then all related action events must occur. If not, then violation to particular policy is reported. The evaluation of whether system events are complied with policies to collect evidence is done by monitoring each policy on system event logs as source of evidences.

More specific, policy is defined with temporal logic language using temporal operators. This definition enables monitoring to be done through temporal logic reasoning. The policy formula is considered as rule and all system events in logs are considered as facts. As suggested in [25], Metric First-Order Temporal Logic (MFOTL) is used to formulate the policy as rule. Generally, events represented as timed-sequence of first-order structure are monitored, when an event occurs, monitoring determines whether the sequence satisfies with the policy defined. If not, violation to policy is reported. The monitor works by evaluating negated format of the policy formula. All events satisfying the negation form of the formula means that the events are violating the policy.

Figure 3.13. shows how the collection process is implemented. The process uses MonPoly monitoring tool which is explained in section 2.10. MonPoly Monitoring Tool. This tool expects log and policy formula as inputs and will output any violations to the policy based on information in the log. As input to the collection process are source of evidence in form of system logs and policy formula. The first step in collection process is to prepare the source of evidence as log according to the format given for MonPoly monitoring tool. In addition, formula which is firstly given in A-PPL format is converted to MFOTL formula expected by MonPoly monitoring tool. As the preparation is completed, MonPoly monitoring tool is called by giving the policy formula and log as inputs. Output from MonPoly tool is evidence that policy is violated by some events in the log. This evidence is collected as evidence from the overall collection process.

Pseudocode of monitoring policy on system log is given in following Figure 3.14., based on the definition given in [4]. Generally the monitor processes the system logs as timed-sequence sequentially. At each iteration, the monitor builds relation of the current event and check if the built relation matches the policy definition. If not, then it reports violation.

Simple example on how the evidence collection process works is shown following. Suppose that there is a policy stating that as data processor, primary cloud service provider must log all access to data subject personal data. Following statement shows the simple formulation of the policy in MFOTL

```
accessDS(?r) IMPLIES EVENTUALLY[0,5M] DPLogAccess
```

Suppose that an access event to personal data is occured. Source of evidence contains this information. However, following the access event, there is no log event performed by cloud service provider. Following figure shows example of source of evidence containing only information about access event without associated log event.

```
@1397835131 accessDS (person_66)
@1397835132 forwardDS (person_66)
@1397835133 DPLog (person_66)
@1397835133 DPLog (person_66)
@1397835135 RelativeCollect (person_15)
@1397835135 DCLog (person_66)
@1397835135 DCLog (person_66)
@1397835135 deleteDS (person_66)
@1397835136 DCEvidReqReceived (person_66)
@1397835138 DPLog (person_66)
@1397835138 DCEvidCollect (person_66)
@1397835139 DPLogDelete (person_66)
@1397835288 askConsentDS (person_78)
@1397835291 DCProcess (person_78)
@1397835295 askConsentDS (person_78)
@1397835295 DCCollect (person_78)
@1397835412 deleteDS (person_59)
@1397835413 Notify (person_59)
@1397835413 DPLogDelete (person_59)
@1397835413 Notify (person_59)
@1397835414 DPLog (person_59)
@1397835415 DPLogDelete (person_59)
@1397835416 DCLog (person_59)
```

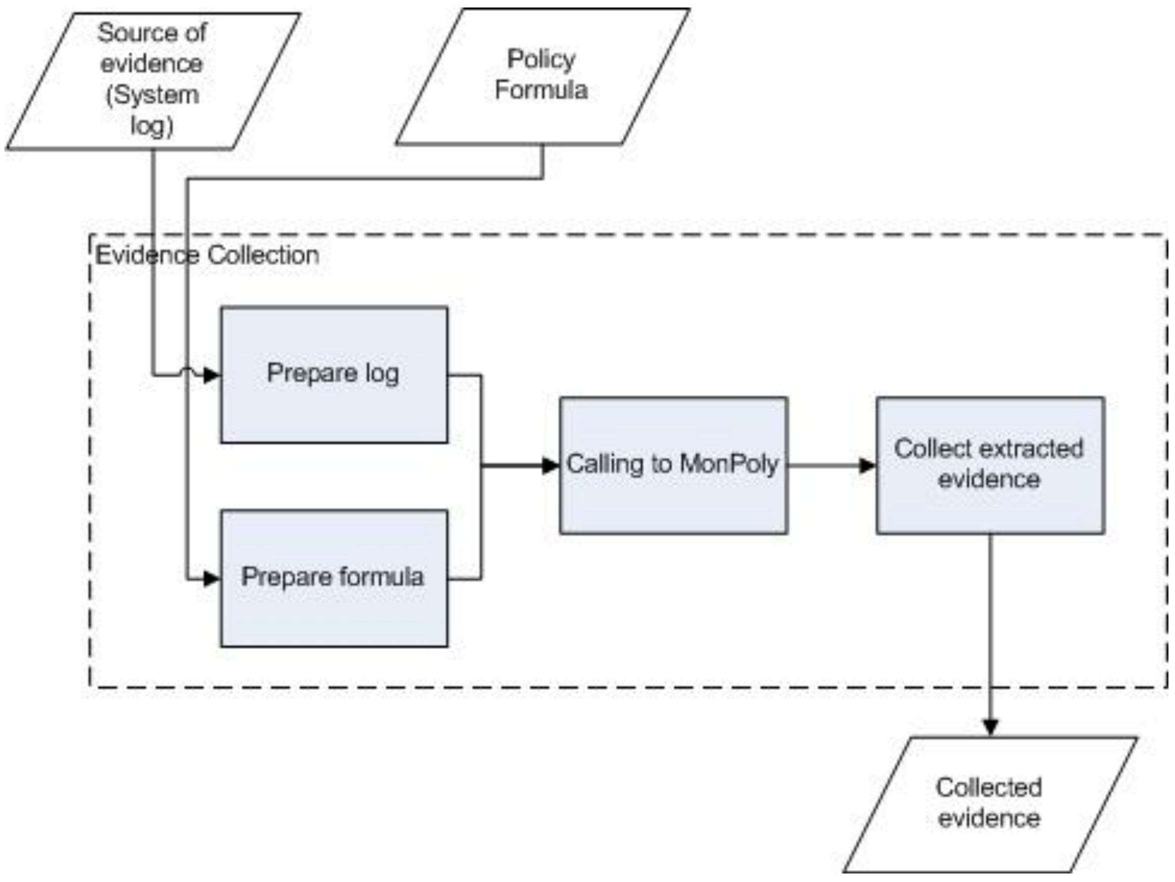

Figure 3.13. Evidence Collection Process with MonPoly

```
Input policy formula Y

Sequence of events as timed-sequence

For each element X in timed-sequence
      Build relation of X

      If relation of X is complete
            Evaluate if relation of X matches policy Y
            Report violation
      EndIf
EndFor
```

Figure 3.14. MonPoly Policy Monitoring Pseudocode

Evidence collection in this case outputs violation because there is no compliance to the policy defined. The collected evidence is tied to the policy identification to correlate which evidence of violation is for which policy. This enables verification of evidence as next step or event tracing of the evidence information in relation to log information and policy.

3.10. LTL and MFOTL

MonPoly tool provides monitoring on log as source of evidence based on policy formula expressed in MFOTL. While, the accountability policy representation framework proposed A-PPL language to expressed accountability policy which is based on Linear Time Temporal Logic (LTL). This section describes study performed in this thesis to compare MFOTL and LTL. This is to have information on determining if MFOTL is suitable on expressing policy in the selected use case so that MonPoly monitoring tool can be used to collect evidence on policy violation. If that the case, policy for the selected use case i.e. healthcare service in cloud, which is expressed in A-PPL will then be translated to MFOTL to be processes within MonPoly monitoring tool. Policy in A-PPL language format already has contained accountability properties within it and brought temporal semantics with it as explained in [24]. Therefore, in this thesis, definition of policy will start on A-PPL and continue on MFOTL based on the comparison study performed in this section.

Comparison of LTL and MFOTL is given in following Table 3.1.

	Comparison Items	Linear-time Temporal Logic (LTL)	Metric First-order Temporal Logic (MFTOL)
1	Extension over	Propositional Logic – logic that concerns with propositions and their relationships, proposition is a possible condition of the world that is in attention [33]	Metric Temporal Logic [2] – MTL is extension of classical temporal logic with real-time timing constraint [34] and quantitative temporal operators [35]
2	Type of logic	Propositional [36]	Quantitative temporal logic [37] – logic that equipped with capability to express event in time unit e.g. “X will happen within one unit of time” [38]

3	Time model	Discrete-time [34]	Real-time [34]
4	Application area	Concurrent and reactive systems [39]	Real time systems
5	Time references	Discrete time [2]	With timing constraints (in interval) [34]
6	Operator	Temporal operators – until, since, eventually, always [40]	Quantitative metric temporal operators [3]
7	Expressiveness	Expresses linear sequence of states [40]	Express metric constraint [37], can be used to specify broad set of complex temporal constraints with reasonable computation complexity [41]
8	Order	Sequence of states where each point in time has a unique successor [42]	Event and state predicates with complex temporal relationships [2]
9	Temporal modalities	Next, Until	Quantitative temporal operators [2] with time interval
10	Interpreted over	Structure (S,→, L), S is set of states, →is transition relation, L is labelling function [40]	Arbitrary structure [40], Signature (C,R,t), C is set of constants, R is set of predicates, t is functions from R → N [2]
11	Satisfiability and model checking	Decidable [40] – satisfiability is decidable means there is possible way to find an interpretation of model that makes the formula true. Model checking checks model of world given the specification in logic formula.	Undecidable [2], decidable over finite time
12	Axiomatization	Sound and complete [40]	Sound [2]
13	Complexity of satisfiability	NP-Complete	High computational complexity [40]
14	Model checking tool	SPIN	Not available free
15	Drawback	Inadequate for real-time system [34], cannot express time interval [34]	Double translation from LTL → MFOTL → temporal logic programming language
16	Complexity of model checking	Model checking problem for LTL can be solved in time $2^{O(\|\varphi\|)}O(\|S\|)$ [43] where φ is the formula and S is the structure	Model checking problem for MTL over finite time has non-primitive recursive complexity [44]

Table 3.1. LTL and MFOTL Comparison

From the comparison result shown on the table, it can be seen that MFOTL with its properties can be used to express accountability policy used in the selected use case. MFOTL is applicable for real time system where expressiveness in time unit with timing interval is needed. The quantitative metric temporal operators enable expressing of event and state predicates with complex temporal relationship. While it enables expression of temporal relationship, MFOTL has reasonable computation complexity. With the use of MFOTL in expressing policy formula within evidence collection process, a translation from A-PPL to

MFOTL is needed. It is because the definition of policy for the selected use case is starting from A-PPL as suggested in [24].

3.11. Temporal Logic Processing Tools

As temporal logic is used for expressing semantic of policy in the selected use case, temporal logic reasoning is needed for processing the policy in the purpose of collecting evidence on policy violation based on the log information. Several existing logic language that supports temporal logic reasoning are studied. PROLOG [45] is programming language specified for logic processing, usually used for artificial intelligence. It is based on running query over relations that represent facts and rules. In the selected use case, PROLOG can be used to process policy and source of evidence to produce evidence on policy violations based on processing on the semantic part of the information provided. This section describes study performed in this thesis to study several existing PROLOG extension [46] that support temporal logic.

Several existing PROLOG extension supporting temporal logic studied are:

1. Templog [47]

   Templog extends classical PROLOG to include temporal operators. Templog program is collection of temporal Horn clause and is interpreted with temporal SLD-resolution. Temporal logic used in Templog is first-order temporal logic with temporal operators including “next”, “always”, “eventually”, “until”, and “precedes”. The time model is considered discrete and linear. General temporal resolution system is used to evaluate temporal logic program. Temporal logic program contains information about facts and rules. The evaluation is considered as evaluating query to the temporal logic program as it is done in PROLOG. However, up to the time of this thesis, there is no publicly available implementation of Templog.

2. Temporal Prolog [48]

   Temporal Prolog is an extension of Prolog that can handle temporal constraints. It uses temporal constraint model for reasoning about time intervals and temporal relationships. Temporal relationships introduced in Temporal Prolog includes for example “before”, “after”, “overlap”, and etc. with time interval constraint. The reasoner is implemented in C language. However, the implementation can’t be found for downloading to solve collection process for selected use case in this thesis.

3. Chronolog [49]

Chronolog is logic programmic language based on discrete linear time temporal logic which extends PROLOG. It has temporal operators such as “first” and “next”. Semantics of Chronolog are developed using temporal interpretation introduced by Herbrand [49]. When compared with Templog, Chronolog seems to lack of expressive power. Yet, the implementation of Chronolog is not available to be used for solving the collection process for selected use case in this thesis.

4. MonPoly [29]

MonPoly is monitoring tool for checking if events on log are complied with policy defined. It is developed based on monitor algorithm introduced in [27]. In MonPoly, policies are expressed in MFOTL formula. The monitoring is implementing algorithm proposed in [27] by taking information about policy in form of MFOTL formula and information about system activities in form of log file. It then reports any violations to policies based on information in log file.

5. Metatem [50]

Metatem can be used to evaluate temporal formula. It is imperative language for executing temporal logic. Execution of temporal logic in Metatem is based on general form about antecedent (about the past) and consequent (about present or future). The form is basically interpreted as “if antecedent holds then execute consequent”. In Metatem, behaviour of system component is described as temporal rules while occurrence of an action is considered as proposition. In Metatem, there are temporal connectives like “next”, “last”, “always in future”, “sometime in future”, “always in past”, “sometime in past”, “since”, and “until”. However, based on the definition of general form used by Metatem, the imperative execution is more on enforcement of consequents to happen rather than monitoring if consequent is not executed. Therefore, it is not that suitable for the selected use case.

The study of existing PROLOG extensions supporting temporal logic shows that MonPoly is the most appropriate and can be used for evidence collection process within accountability policy monitoring in cloud service. The usage of MonPoly then requires some preprocessing to prepare the inputs for the tool.

3.12. A-PPL Translation

Accountability policy is expressed in A-PPL. This representation involves semantics that is processed through an A-PPL engine which is based on Linear Time Temporal Logic (LTL). The policy monitoring tool that is used in this Thesis i.e. MonPoly, on the other hand, processes policy that is expressed in Metric First-Order Temporal Logic (MFOTL). Based on the study comparison that is conducted in section 3.10. LTL & MFOTL, it can be seen that these two types of logic differs in some areas. In processing the selected use case in this

Thesis, MFOTL is used to express the policy defined for the use case which later is processable in MonPoly monitoring tool to collect evidence on policy violation.

As it is explained in section 3.3. Accountability Evidence Collection, the policy definition starts from A-PPL where it already satisfies the requirements for accountability policy as proposed in [24]. This definition is used for next processing. A translation is, therefore, needed to have this A-PPL policy in MFOTL as used in MonPoly monitoring tool. This section describes how the translation from A-PPL and MFOTL is performed and implemented.

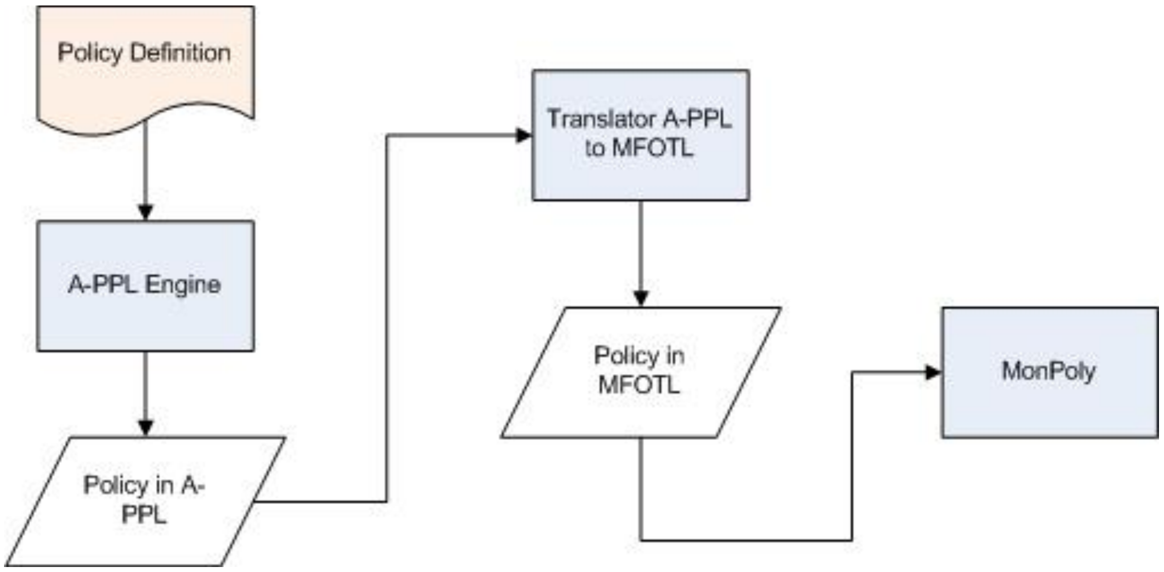

Figure 3.15. A-PPL Translation and MonPoly

Figure 3.15. shows how the definition of policy is used in evidence collection as part of framework of accountability evidence collection. MonPoly as policy monitoring tool to collect evidence receives, as input, policy definition in MFOTL. This representation is translated by a A-PPL to MFOTL translator. The A-PPL format is produced by A-PPL engine from human defined policy.

Figure 3.16. later shows how the A-PPL to MFOTL translator is implemented. The translator receives 2 inputs i.e. policy definition in A-PPL XML format and control parameters. A-PPL XML is a document produced by A-PPL engine proposed in [24]. The control parameters are to define parameters needed for a specific use case. For example how to translate an action from A-PPL term into observed-use case term. Therefore, this translator can be used for any other use case as long as the parameters in the control parameters are correctly set up. Translator firstly read A-PPL elements from the input XML and feeds in the generator of MFOTL formula. Each MFOTL formula generated will be appended to policy definition XML file.

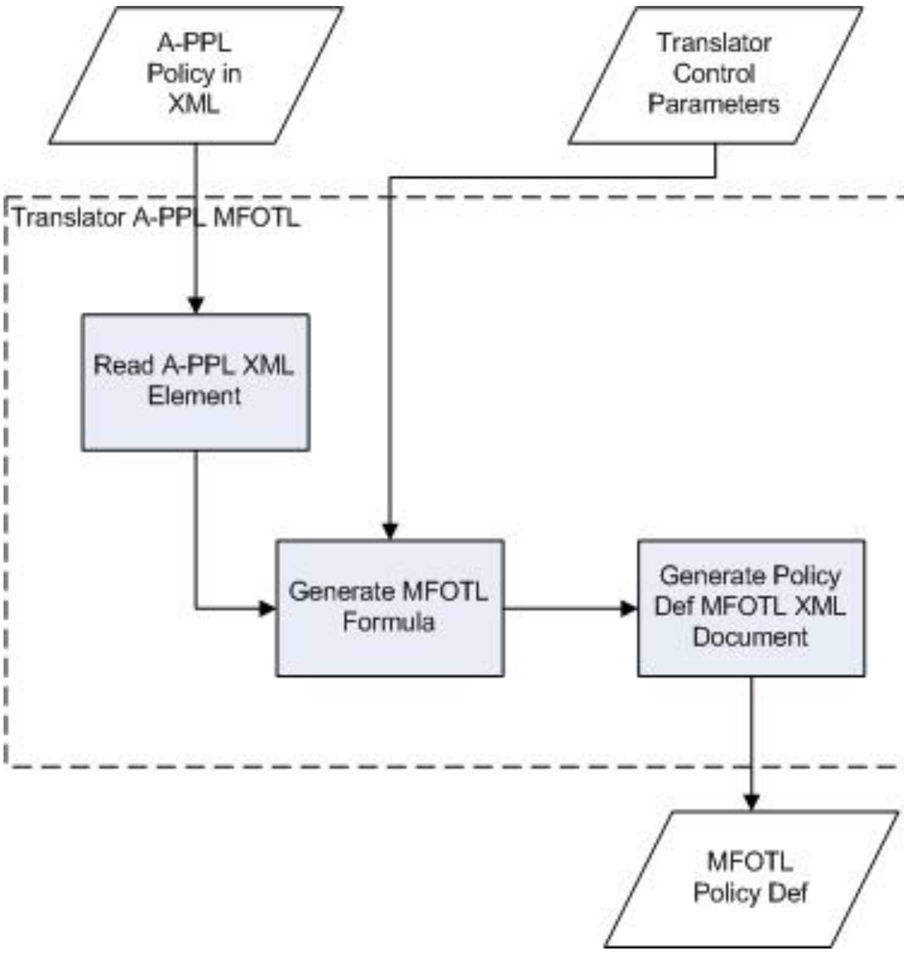

Figure 3.16. A-PPL Translator

Figure 3.17. shows the pseudocode of A-PPL to MFOTL translator. It basically reads all obligations in A-PPL XML document and iterates over the obligations to generate MFOTL formulas and appends them sequentially to MFOTL XML output document. In each iteration, trigger and action elements of an obligation are read from input XML. Within each trigger and action, several attributes such as role, subject, and action are read and tied with the associated trigger and action. These attributes determine how the translator translate A-PPL policy to MFOTL formula based on the selected use case which information is put in the control parameters. A MFOTL formula is constructed once the triggers and actions with their attributes are extracted from A-PPL XML. This generated formula is appended to policy definition XML file to be used by MonPoly monitoring tool.

```
Input <- A-PPL XML
Output <- MFOTL XML
Read obligations from Input
For each obligation X
      Read triggers for obligation X from Input
      Read actions for obligation X from Input


      For each trigger Y
```

```
            Read attributes (Role, Subject, Action)
      EndFor
      Combine triggers to collection T of triggers

      For each action Z
            Read attributes (Role, Subject, Action)
      EndFor
      Combine actions to collection A of actions

      Construct MFOTL formula F for X based on T and A


      Append formula F to Output
EndFor
```

Figure 3.17. A-PPL Translator Pseudocode

3.13. MFOTL Formula

MonPoly monitoring tool processes policies expressed in metric first-order temporal logic. For the selected use case, all obligations defined are then also need to be formulated in MFOTL. Following list shows MFOTL expression of each policy defined in section 3.2. Use Case and Policies. Same relation or event terms are used as those used in A-PPL expression and system event logs. The syntax for MFOTL follows as what explained in section 2.8. Temporal Logic (& MFOTL).

1. As a data controller, the hospital needs to provide a policy on what data is collected and for what purposes

$$\square \forall f.DCcollect(f) \rightarrow \blacklozenge_{[0,t]} processInfo(f)$$

2. As a data controller, the hospital must ask the data subjects (patients) explicit consent for collecting and processing personal data

$$\square \forall f.DCcollect(f) \vee \square \forall f.DCprocess(f) \rightarrow \blacklozenge_{[0,t]} askConsent(f)$$

3. As joint data controllers, the relatives must ask the data subjects (patients) explicit consent for collecting and processing personal data

$$\square \forall f.collect(f) \vee \square \forall f.process(f) \rightarrow \blacklozenge_{[0,t]} askConsent(f)$$

4. As a data controller, the hospital must, upon request, provide evidence to the data subjects on their personal data processing activities

$$\Box \forall f.access(f) \vee \Box \forall f.forward(f) \vee \Box \forall f.delete(f) \rightarrow \Diamond_{[0,t]} \log(f)$$

$$\Box \forall f.evidencereq(f) \rightarrow \Diamond_{[0,t]} evidencecollect(f)$$

5. As a data processor, the primary service provider must log all access to personal data

$$\Box \forall f.access(f) \rightarrow \Diamond_{[0,t]} \log(f)$$

6. As a data processor, the primary service provider must, upon request, provide evidence to the data controller (hospital) on its personal data processing activities

$$\Box \forall f.access(f) \vee \Box \forall f.forward(f) \vee \Box \forall f.delete(f) \rightarrow \Diamond_{[0,t]} \log(f)$$

$$\Box \forall f.evidencereq(f) \rightarrow \Diamond_{[0,t]} evidencecollect(f)$$

7. As a data processor, the primary service provider must, upon request, provide evidence to the data controller (hospital) on the correct and timely deletion of personal data

$$\Box \forall f.delete(f) \rightarrow \Diamond_{[0,t]} \log(f) \wedge \Diamond_{[0,t]} notify(f)$$

3.14. Evidence Collection with Pyke

Another approach on performing evidence collection is to use pure Prolog approach. With this approach, processing of policies is treated as first order logic processing. This is because that, the time information tied with every event in the system is attribute to the event. Therefore, in processing the event in terms of collecting evidence, the thing that is interested is the matching of event with policies. This is comparable with approach used in MonPoly where the time information is considered as temporal information on the temporal logic used to describe the system. In Prolog approach case, evidence collection can be performed by using first order logic processing. Policies defined for the selected use can be expressed in first order logic. Prolog is one of the programming languages that are based on first-order logic. The main processing of policies with regards to evidence collection is then performed using Prolog language.

Several implementations of Prolog reasoner exist. One of them is Pyke which is introduced in section 2.11. Pyke. Pyke implements inference engine based upon Prolog in Python environment. In our use case, Pyke is used to solve the problem on evidence collection. More specifically, Pyke inference engine is used on processing policies as rules and source of evidence as facts. The inference engine is used to derive additional information on facts based on the rules defined.

Collection process using Pyke is done by firstly defining policies in first-order logic as rules. In addition to that, source of evidence is prepared as facts. Following Figure 3.18. shows how collection process is performed using Pyke.

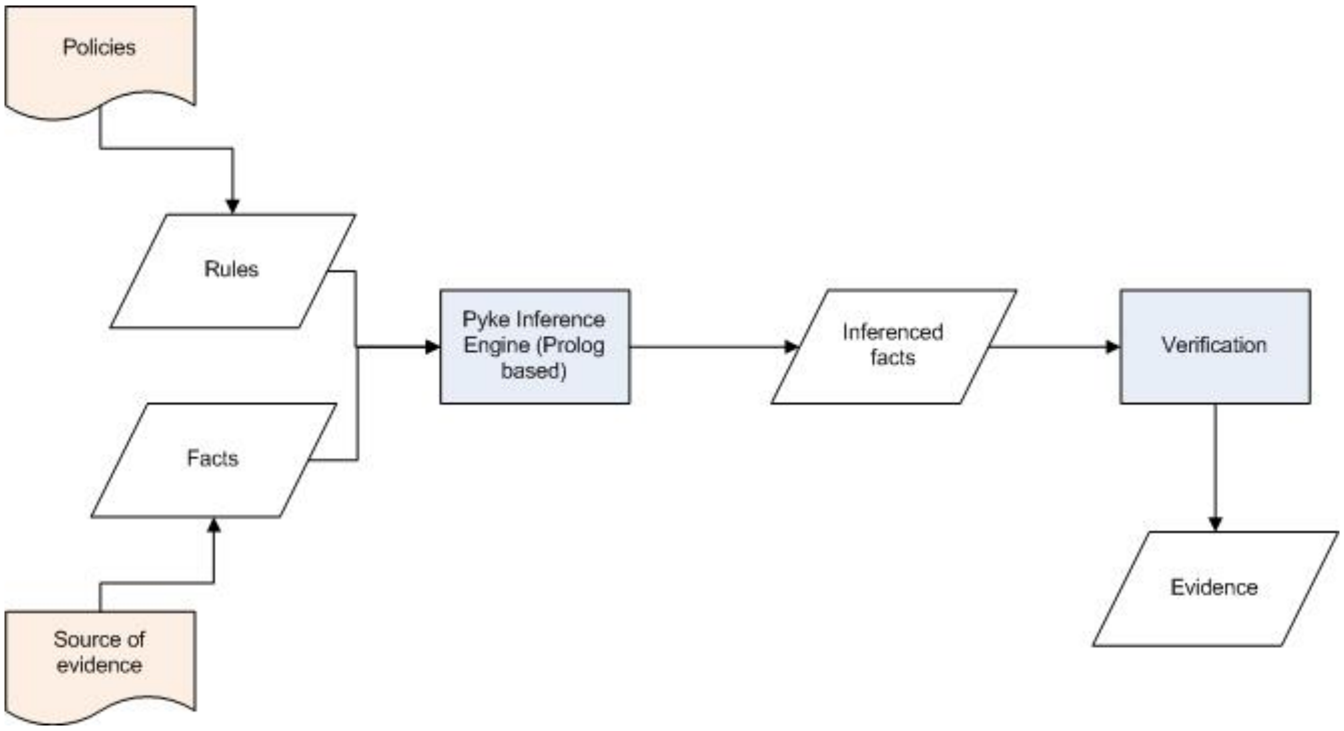

Figure 3.18. Evidence Collection with Pyke

Pyke uses its inference engine to produce additional facts based on policy rules and source of evidence. These results are going through verification process before is taken as evidence. The verification is the process of deciding whether events in source of evidence are complied with policies. Verification is based on the result produced by Pyke inference engine.

Each policy defined for the selected use case i.e. healthcare service in cloud is defined as rule in Pyke krb file. Following list shows the rule definition for each policy mentioned in section 3.2. Use Case and Policies.

1. O1: As a data controller, the hospital needs to provide a policy on what data is collected and for what purposes

```
Obligation_1
    foreach
      source.DCCollect($instance)
    assert
      source.DCDefinePurpose($instance)
```

2. O2: As a data controller, the hospital must ask the data subjects (patients) explicit consent for collecting and processing personal data

```
Obligation_2_1
    foreach
        source.DCProcess($instance)
      assert
        source.askConsentDS($instance)

Obligation_2_2
      foreach
        source.DCCollect($instance)
      assert
        source.askConsentDS($instance)
```

3. O3: As joint data controllers, the relatives must ask the data subjects (patients) explicit consent for collecting and processing personal data

```
Obligation_3_1
      foreach
          source.RelativeCollect($instance)
      assert
          source.RelativeAskConsentDS($instance)

Obligation_3_2
      foreach
          source.RelativeProcess($instance)
      assert
          source.RelativeAskConsentDS($instance)
```

4. O4: As a data controller, the hospital must, upon request, provide evidence to the data subjects on their personal data processing activities

```
Obligation_4_1
      foreach
          source.accessDS($instance)
      assert
          source.DCLog($instance)

Obligation_4_2
      foreach
          source.deleteDS($instance)
      assert
          source.DCLog($instance)

Obligation_4_3
      foreach
          source.forwardDS($instance)
      assert
          source.DCLog($instance)

Obligation_4_4
      foreach
          source.DCevidReqReceived($instance)
      assert
          source.DCevidCollect($instance)
```

5. O5: As a data processor, the primary service provider must log all access to personal data

```
Obligation_5
      foreach
          source.accessDS($instance)
      assert
          source.DPLogAccess($instance)
```

6. O6: As a data processor, the primary service provider must, upon request, provide evidence to the data controller (hospital) on its personal data processing activities

```
Obligation_6_1
      foreach
          source.accessDS($instance)
      assert
          source.DPLog($instance)

Obligation_6_2
      foreach
          source.deleteDS($instance)
      assert
          source.DPLog($instance)

Obligation_6_3
      foreach
          source.forwardDS($instance)
      assert
          source.DPLog($instance)

Obligation_6_4
      foreach
          source.DPevidReqReceived($instance)
      assert
          source.DPevidCollect($instance)
```

7. O7: As a data processor, the primary service provider must, upon request, provide evidence to the data controller (hospital) on the correct and timely deletion of personal data

```
Obligation_7
    foreach
      source.deleteDS($instance)
    assert
      source.notify($instance)
      source.DPLogDelete($instance)
```

All events used in the rules are based on the definition given in section 3.7. Event Generation. Therefore source of evidence produced by test case system can also be used in this evidence collection process. In addition to that, comparison between temporal logic and first order logic used in MonPoly and Pyke consecutively is possible. The definition of rules is based on the Pyke's rule which is to use foreach-assert syntax as explained in section 2.11. Pyke.

3.15. Normalization

The collected evidence needs to have standard format and contains enough information to support the evidence for description and later processing such as verification. Therefore, in order to have that, a step in the framework of accountability evidence collection is added after evidence collection process which is called evidence normalization. The format should be machine-readable so that automatic processing of evidence is possible. Later in verification process, the evidence is matched against policies. Thus the normalization must introduce enough information supporting this process but with simple form that simplify the verification process.

Several attributes are needed for the evidence to have it described properly and processable in the next step of the framework of accountability evidence collection. Those suggested attributes are (adapted from [24]):

- A1: descriptor of action or operation

  This attribute contains information about action or operation that the evidence is referring to i.e. action that is detected to not comply with obligations or policies.

- A2: actor, operator or component identifier

  This attribute contains information about actor who initiates the action defined in A1.

- A3: metadata (timestamps, location)

  This attribute provides details for each action defined in A1. This information is extracted from the system itself. It may contain information that sufficiently describe the evidence and is useful for further processing such as verification.

- A4: policy identification

  This attribute provides information about policy which is the link between the evidence and associated policy. This also gives way for further processing of the evidence such as verification.

With those attributes defined for an evidence, the collection evidence is going through a normalization step which collects all information related to the attributes. The normalization process takes the attribute value from the source of evidence and policies as its input. It wraps up all the attributes within each collected evidence.

Figure 3.19. shows how the normalization is performed. With the information about what attributes to have in each evidence, the normalization process collects attribute's value from system information and policies.

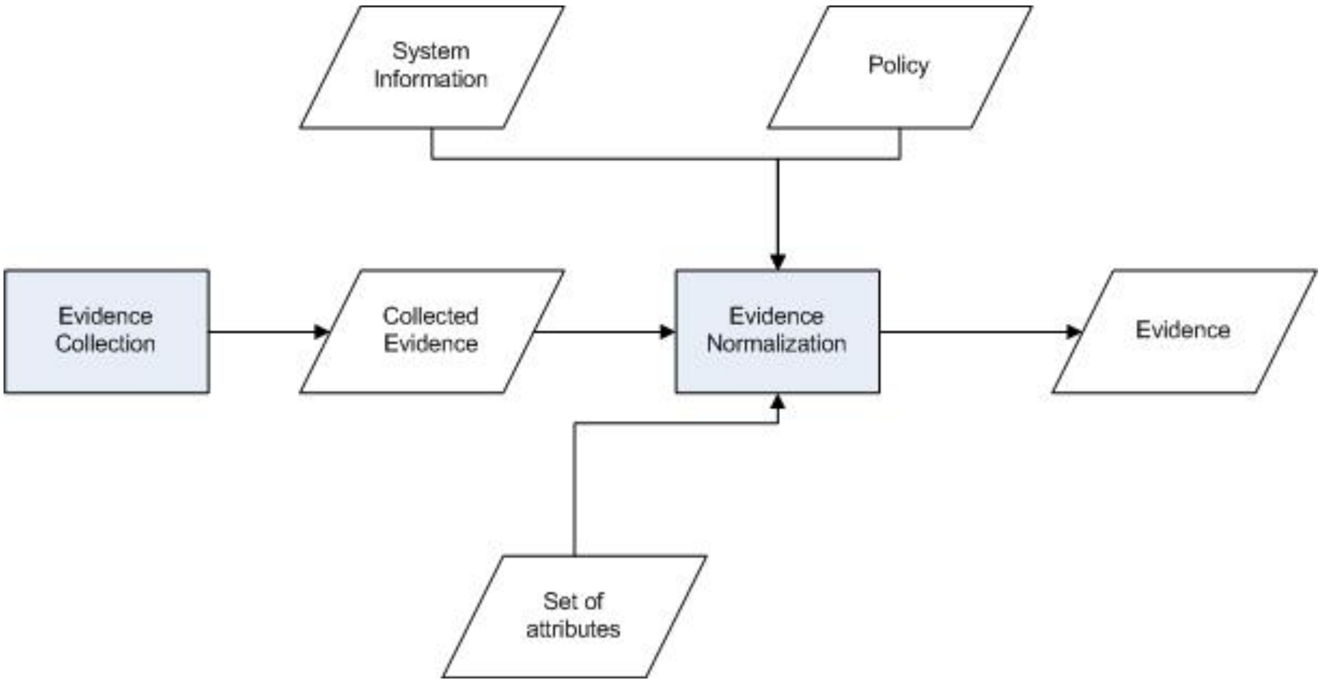

Figure 3.19. Evidence Normalization

3.16. Verification Process

The evidence collection process with MonPoly produces evidence on policy violation. Based on the framework of accountability evidence collection, this evidence is to be verified to determine whether it is valid with respect to the policy defined for the selected use case and associated source of evidence from where the evidence is collected from.

Figure 3.20. shows how the verification process in proposed framework of accountability evidence collection is implemented. It uses decomposition approach to determine validity of the evidence. It determines whether evidence is confirmed or is negated by other information. The determination of a level is done by executing determination on level below it whether it is confirmed or negated. Therefore, the evaluation is in bottom-up direction. In the policy compliance checking situation, the verification of evidence, i.e. evidence provided for any policy violation, is executed by checking confirmation whether the information provided is really showing that the policy is violated or by checking if the negation of the checking itself holds (which mean the evidence does not valid). Violating a policy in terms of trigger-action pair in accountability policy representation means that trigger events are not triggering action events to occur. It is expected that action events occur within some predefined conditions.

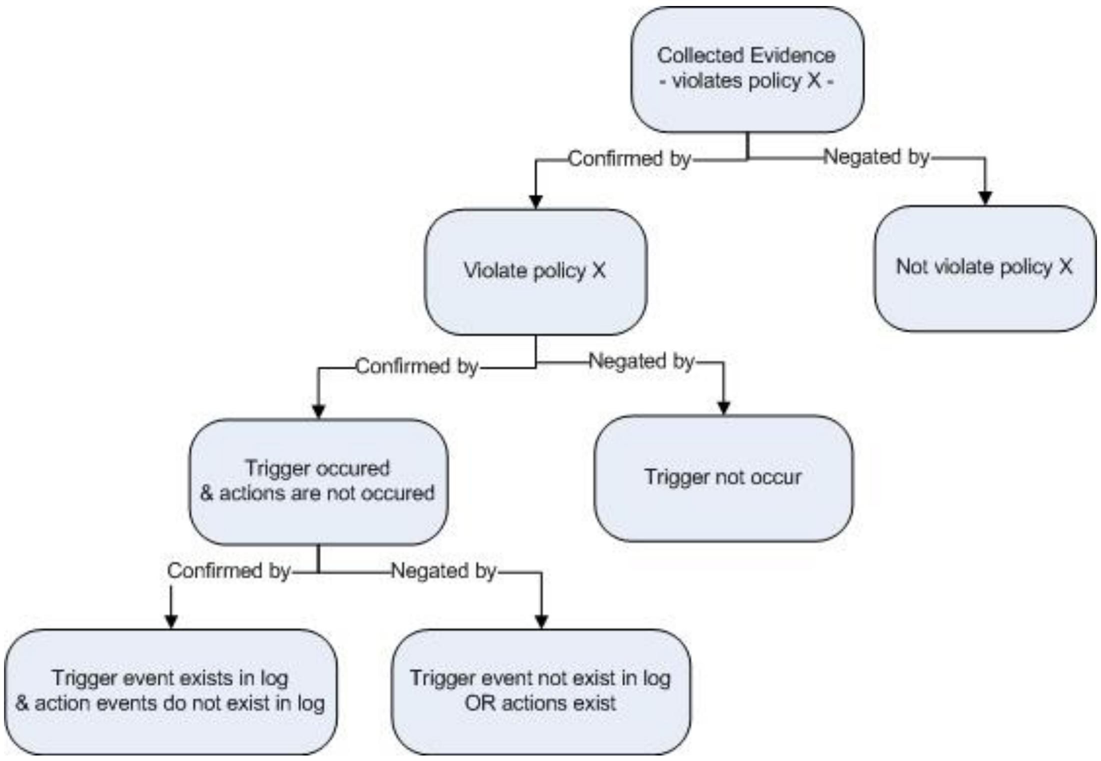

Figure 3.20. Verification Method

Generally the evidence is validated by processes depicted in figure 3.21. The confirmation statement is checked whether it holds. If it holds, then the evidence being verified is accepted. If it does not hold, the associated negation statement is checked whether it holds. If it holds, then the evidence is rejected because it is not valid, supported by the information in the negation statement. If it does not hold, then it is undecidable whether the evidence is valid or not.

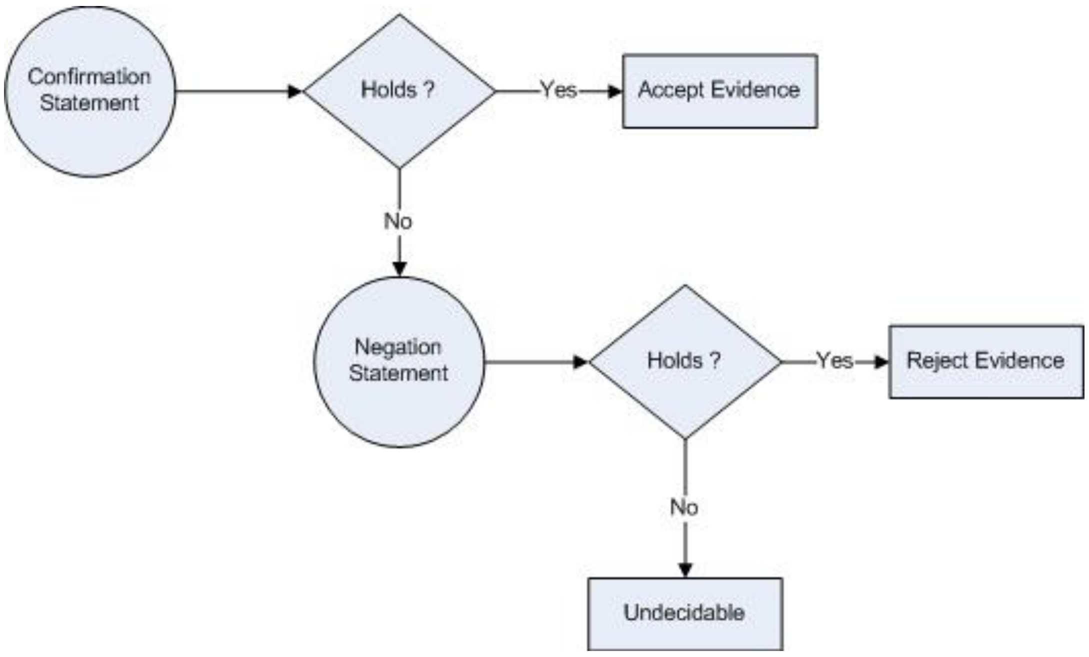

Figure 3.21. Verification Principle

4. Result & Analysis

This section presents results of running the implementation of framework of evidence collection. The results presented include source of evidence collected from test system where events are generated from implemented event generator, MFOTL formula generated from A-PPL translator, several cases on detecting violations to policies using evidence collection tool, and also case on evidence verification process. Comparison between two approaches used i.e. MFOTL with MonPoly and Prolog (FOL) with Pyke is discussed. At last, scalability analysis on the proposed approach on the framework of evidence collection is presented.

4.1. Source of Evidence

The test case system, which is set up using VMs to simulate the selected use case i.e. healthcare service in cloud, has logs that are used as source of evidence in the framework of accountability evidence collection. Through a running of this system, test data is collected for the purpose of testing the proposed method for evidence collection. The event generation explained in section 3.7. Event Generation is used to automatically run the simulated system to generate events that will be logged in the system logs. These logs are collected as source of evidence. As explained in the proposed method and implementation, all events or operations performed in the system (VMs) are logged in the system log. These logs have different format between each other. Therefore, normalization is performed on each log to produce common format and that is processable in the next step. As MonPoly monitoring tool is used in evidence collection process, all these normalized logs are needed to be converted to format processable in MonPoly.

Following example shows example of system logs that are collected as the test system is running. The logs are collected from each VM acting as each entity in healthcare service in the cloud. The entities are data subject (patient), data controller (hospital and relative), and data processor (cloud service provider). All the events are corresponding to operations that are performed in healthcare service in cloud. Log shown below is the merging result from all logs collected from all VMs to form complete view of the system run. Each event is given tag so that it is human-readable with respect to associated defined policies. In the log, there is information about time of event occurrence, user performed the event, and data subject personal data identification.

```
Apr 18 17:32:16 user1-VirtualBox notify-send: DCEvidReqReceived 152.94.0.201 152.94.0.206
            person_66
Apr 18 17:32:27 user1-VirtualBox notify-send: accessDS 152.94.0.201 152.94.0.180 person_66
Apr 18 17:32:21 user2-VirtualBox system-logging: DCEvidCollect person_66
Apr 18 17:32:31 user2-VirtualBox system-logging: DCLog person_66
Apr 18 17:32:33 user3-VirtualBox system-logging: DPLog person_66
Apr 18 17:32:44 user3-VirtualBox system-logging: deleteDS person_66
Apr 18 17:32:53 user3-VirtualBox system-logging: DPLogDelete person_66
Apr 18 17:32:57 user3-VirtualBox system-logging: DPLog person_66
Apr 18 17:32:39 user3-VirtualBox scp-auth: user-3 PWD=/home/user-3 user=root
           COMMAND=/usr/bin/scp /home/user-3/MyData/Data user-4@152.94.0.210:/home/user-4/MyData
           forwardDS person_66
Apr 18 17:32:33 user3-VirtualBox system-logging: DPLog person_66
Apr 18 17:32:43 user3-VirtualBox system-logging: DPLog person_66
Apr 18 17:32:15 user4-VirtualBox notify-send: RelativeCollect 152.94.0.210 152.94.0.201
           person_15
Apr 18 17:34:59 user2-VirtualBox notify-send: askConsentDS 152.94.0.206 152.94.0.201 person_78
Apr 18 17:34:52 user2-VirtualBox system-logging: DCProcess person_78
Apr 18 17:34:56 user2-VirtualBox notify-send: askConsentDS 152.94.0.206 152.94.0.201 person_78
Apr 18 17:35:04 user2-VirtualBox notify-send: DCCollect 152.94.0.206 152.94.0.201 person_78
Apr 18 17:36:55 user3-VirtualBox system-logging: deleteDS person_59
Apr 18 17:36:54 user3-VirtualBox notify-send: Notify 152.94.0.180 152.94.0.201 person_59
Apr 18 17:37:00 user3-VirtualBox system-logging: DPLogDelete person_59
Apr 18 17:37:04 user3-VirtualBox notify-send: Notify 152.94.0.180 152.94.0.201 person_59
Apr 18 17:37:03 system-logging: user3-VirtualBox DPLog person_59
```

The collected logs which will be the basis for source of evidences are then normalized. Following Figure 4.1. shows result of normalization of above log as explained in the section 3.15. Normalization. Based on this normalization, all events' information is in uniform format and is processable in next processing step.

```
Apr 18 17:32:16 1397835136 notify-send: user1-VirtualBox DCEvidReqReceived person_66
Apr 18 17:32:27 1397835131 notify-send: user1-VirtualBox accessDS person_66
Apr 18 17:32:21 1397835141 system-logging: user2-VirtualBox DCEvidCollect person_66
Apr 18 17:32:31 1397835135 system-logging: user2-VirtualBox DCLog person_66
Apr 18 17:32:33 1397835133 system-logging: user3-VirtualBox DPLog person_66
Apr 18 17:32:44 1397835135 system-logging: user3-VirtualBox deleteDS person_66
Apr 18 17:32:53 1397835139 system-logging: user3-VirtualBox DPLogDelete person_66
Apr 18 17:32:57 1397835138 system-logging: user3-VirtualBox DPLog person_66
Apr 18 17:32:39 1397835132 scp-auth: user3-VirtualBox forwardDS person_66
Apr 18 17:32:33 1397835133 system-logging: user3-VirtualBox DPLog person_66
Apr 18 17:32:43 1397835133 system-logging: user3-VirtualBox DPLog person_66
Apr 18 17:32:15 1397835135 notify-send: user4-VirtualBox RelativeCollect
person_15
Apr 18 17:34:59 1397835288 notify-send: user2-VirtualBox askConsentDS person_78
Apr 18 17:34:52 1397835291 system-logging: user2-VirtualBox DCProcess person_78
Apr 18 17:34:56 1397835295 notify-send: user2-VirtualBox askConsentDS person_78
Apr 18 17:35:04 1397835295 notify-send: user2-VirtualBox DCCollect person_78
Apr 18 17:36:55 1397835412 system-logging: user3-VirtualBox deleteDS person_59
Apr 18 17:36:54 1397835413 notify-send: user3-VirtualBox Notify person_59
Apr 18 17:37:00 1397835413 system-logging: user3-VirtualBox DPLogDelete person_59
Apr 18 17:37:04 1397835413 notify-send: user3-VirtualBox Notify person_59
Apr 18 17:37:03 1397835414 system-logging: user3-VirtualBox DPLog person_59
Apr 18 17:37:06 1397835415 system-logging: user3-VirtualBox DPLogDelete person_59
Apr 18 17:36:59 1397835416 system-logging: user2-VirtualBox DCLog person_59
```

Figure 4.1. Normalized Source of Evidence Example

For processing in MonPoly and Pyke, the log is converted into format processable in those tools. However, reference to the original entry is kept so that detail information can be retrieved whenever needed. As explained in 2.10. MonPoly Monitoring Tool and 2.11. Pyke,example of the processable source of evidence is shown in following Figure 4.2.

```
@1397835131 accessDS (person_66)
@1397835132 forwardDS (person_66)
@1397835133 DPLog (person_66)
@1397835133 DPLog (person_66)
@1397835135 RelativeCollect (person_15)
@1397835135 DCLog (person_66)
@1397835135 DCLog (person_66)
@1397835135 deleteDS (person_66)
@1397835136 DCEvidReqReceived (person_66)
@1397835138 DPLog (person_66)
@1397835138 DCEvidCollect (person_66)
@1397835139 DPLogDelete (person_66)
@1397835288 askConsentDS (person_78)
@1397835291 DCProcess (person_78)
@1397835295 askConsentDS (person_78)
@1397835295 DCCollect (person_78)
@1397835412 deleteDS (person_59)
@1397835413 Notify (person_59)
@1397835413 DPLogDelete (person_59)
@1397835413 Notify (person_59)
@1397835414 DPLog (person_59)
@1397835415 DPLogDelete (person_59)
@1397835416 DCLog (person_59)
```

Figure 4.2. Processable Log Example

4.2. MFOTL Formula

This section shows the results of translation from A-PPL to MFOTL performed by the translator explained in section 3.12. A-PPL translation on each policies expressed in A-PPL in section 3.4. Policy Representation in A-PPL. This expression in MFOTL is processable in MonPoly tool. The relations used in the MFOTL formula corresponds to all events generated in the test case system to simulate the selected use case i.e. healthcare service in cloud. Each event is the simplification of operation performed in healthcare service because only occurrence of the event and relationship between events are interested instead of the detail on the event. The value in <DEF> XML element shows MFOTL formula for each policy processable in MonPoly. As explained, it is translated from trigger-action extension proposed by A-PPL. Basically relations in the left side of temporal construct in the translated MFOTL formula corresponds to triggers in the A-PPL representation. The triggers are translated into relations defined for the whole system. While relations in the right side of temporal construct corresponds to actions in the A-PPL representation.

1. O1: As a data controller, the hospital needs to provide a policy on what data is collected and for what purposes

```
<POLICY>
          <ID>O1</ID>
          <DEF>DCCollect(?r) IMPLIES ONCE[0,5M]
             DCDefinePurpose</DEF>
</POLICY>
```

2. O2: As a data controller, the hospital must ask the data subjects (patients) explicit consent for collecting and processing personal data

```
<POLICY>
          <ID>O2</ID>
          <DEF>DCCollect(?r) OR DCProcess(?r) IMPLIES ONCE[0,5M]
                         askConsentDS</DEF>
</POLICY>
```

3. O3: As joint data controllers, the relatives must ask the data subjects (patients) explicit consent for collecting and processing personal data

```
<POLICY>
          <ID>O3</ID>
          <DEF>RelativeCollect(?r) OR RelativeProcess(?r) IMPLIES
                        ONCE[0,5M] RelativeAskConsentDS</DEF>
</POLICY>
```

4. O4: As a data controller, the hospital must, upon request, provide evidence to the data subjects on their personal data processing activities

```
<POLICY>
          <ID>04</ID>
          <DEF>accessDS(?r) OR deleteDS(?r) OR forwardDS(?r)
                      IMPLIES EVENTUALLY[0,5M] DCLog</DEF>
          <DEF>DCevidReqReceived(?r) IMPLIES EVENTUALLY[0,5M]
                      DCevidCollect</DEF>
</POLICY>
```

5. O5: As a data processor, the primary service provider must log all access to personal data

```
<POLICY>
          <ID>05</ID>
          <DEF>accessDS(?r) IMPLIES EVENTUALLY[0,5M]
                       DPLogAccess</DEF>
</POLICY>
```

6. O6: As a data processor, the primary service provider must, upon request, provide evidence to the data controller (hospital) on its personal data processing activities

```
<POLICY>
          <ID>06</ID>
          <DEF>accessDS(?r) OR deleteDS(?r) OR forwardDS(?r)
                   IMPLIES EVENTUALLY[0,5M] DPLog</DEF>
          <DEF>DPevidReqReceived(?r) IMPLIES EVENTUALLY[0,5M]
                   DPevidCollect</DEF>
</POLICY>
```

7. O7: As a data processor, the primary service provider must, upon request, provide evidence to the data controller (hospital) on the correct and timely deletion of personal data

```
<POLICY>
          <ID>07</ID>
          <DEF>deleteDS(?r) IMPLIES EVENTUALLY[0,5M] notify</DEF>
          <DEF>deleteDS(?r) IMPLIES EVENTUALLY[0,5M]
                       DPLogDelete</DEF>
</POLICY>
```

The results show that the translator can be used to transform A-PPL format in XML into MFOTL format in XML. However, it is required that the A-PPL format has standard list of triggers and actions. It is because that by knowing the lists, it is then possible to build the translation table used in the translator. It is also that by knowing them, it is possible to link with the events being monitored in the source of evidence. Without consistent defined format in A-PPL, it is not possible to get this automatic translation work. Therefore, it is important that A-PPL representation format is well defined and the information is consulted to the translator.

4.3. Detecting Violation to Obligation with MonPoly

MonPoly monitoring tool outputs all events in the log that are not complied with the policies. In the implementation of framework of accountability evidence collection, all those events may be potential evidences that after verification process can be considered as evidences. In the selected use case, the logs are source of evidences which are collected from system logs. The policies represent obligations that need to be followed by each entity in the system. MonPoly takes policies and source of evidence as input and outputs each event in the source that is violating the policies.

Following Figure 4.3. shows example output of violation detection by running MonPoly against source of evidence and policies. Suppose that in order to test the monitoring tool, we intentionally break some of the policies defined. We expect that events that are intentionally executed to violate the policies to be detected by the tool. To do so, policy indicator in the event generation tool is used to control the creation of such event. Example of source of evidence used in this run is the one shown in section 4.1. Source of evidence above.

```
5
@1397835131 (time-point 1): (person_66)
3
@1397835135 (time-point 5): (person_15)
```

Figure 4.3. Output of Collection with MonPoly

The output of MonPoly monitoring above shows that there are two events that are violating policies defined for the case. The first one is the event occurred in time `1397835131` which is indicated as violating policy with identification `5.` Second event occurred in time `1397835135` which is indicated as violating policy with identification 3. Looking at definition of policy with identification 5 and 3 which is shown below

- O5: As a data processor, the primary service provider must log all access to personal data
- O3: As joint data controllers, the relatives must ask the data subjects (patients) explicit consent for collecting and processing personal data

```
<POLICY>
        <ID>O5</ID>
        <DEF>accessDS(?r) IMPLIES EVENTUALLY[0,5M] DPLogAccess</DEF>
</POLICY>
<POLICY>
        <ID>O3</ID>
        <DEF>RelativeCollect(?r) OR RelativeProcess(?r) IMPLIES
                                 ONCE[0,5M] RelativeAskConsentDS</DEF>
</POLICY>
```

Every access to data subject's personal data must be logged and before collecting or processing the personal data, relative of data subject must ask consent to the data subject. In the first violation, the access event is not triggering the logging event. This can be seen from the source of evidence collected as log from the system. That's why, the access event is considered as potential evidence to show that policy with identification 5 is violated. Same case in the second violation, the collect event executed by data subject's relative is not preceded by consent event. The related collect event is collected as evidence to proof violation to policy with identification 3.

4.4. Detecting Violation to Obligation with Pyke

Taking the same set of data i.e. source of evidences as the one used in MonPoly testing, this section shows the result of detecting violation to obligations with Pyke. As explained in section 3.14. Evidence Collection with Pyke", the obligations are formalized in first order logic as Pyke's rules. The source of evidence where evidence is collected from is treated as facts in Pyke.

Following shows some of the obligations defined for the selected use case i.e. healthcare service in cloud. These obligations are to be monitored against logs and every violation to the obligations is stored as evidence as a violation proof.

- O5: As a data processor, the primary service provider must log all access to personal data

  In Pyke, this obligation is defined in the krb file as:

```
Obligation_5
      foreach
          source.accessDS($instance)
      assert
          source.DPLogAccess($instance)
```

- O3: As joint data controllers, the relatives must ask the data subjects (patients) explicit consent for collecting and processing personal data

  In Pyke, this obligation is defined in the krb file as:

```
Obligation_3_1
      foreach
          source.RelativeCollect($instance)
      assert
          source.RelativeAskConsentDS($instance)

Obligation_3_2
      foreach
          source.RelativeProcess($instance)
      assert
          source.RelativeAskConsentDS($instance)
```

Logs as source of evidence used is the one shown in section 4.1. Source of Evidence. Running Pyke using the rules and facts, gives following output:

```
DCLog (person_66)
DPLogAccess (person_66)
DPLog (person_66)
DCLog (person_66)
DPLog (person_66)
RelativeAskConsentDS (person_15)
DCLog (person_66)
DPLog (person_66)
notify (person_66)
DPLogDelete (person_66)
DCEvidCollect (person_66)
askConsentDS (person_78)
DCDefinePurpose (person_78)
askConsentDS (person_78)
DCLog (person_59)
DPLog (person_59)
notify (person_59)
DPLogDelete (person_59)
```

Pyke matches every event term in the “if” clause of every policy with events in the logs. Each time a match is found, it asserts all events terms in “then” clause as new facts. The result above is asserted new facts based on running Pyke on the source of evidences and policies. As explained in section 3.14. Evidence Collection with Pyke, this result is to be checked in the verification process which will use this new information to check consistency of events in the log against the policies defined for them.

Following Figure 4.4. shows the result of verification process where the output is considered as evidence for each violation to the obligation.

```
Obligation_5, @1397835131 accessDS (person_66)
Obligation_3, @1397835135 RelativeCollect (person_15)
```

Figure 4.4. Output of Collection with Pyke

Each detected violation is tied to the respective obligation. In this way, the link between evidence and obligation is established. In later, further processing taking the evidence as input is possible. More information can also be explored from that basis.

4.5. Test cases and Results

This section shows several test cases and their results in addition to the results shown in previous section about detecting violation to obligations with MonPoly and Pyke. The sample test cases picked up for testing the approaches and tools are:

1. Detecting if Data Controller (Hospital) does not follow obligations on processing Data Subject (Patient)’s personal data

   In this test case, several obligations defined for the selected use case i.e. healthcare service in cloud are related. Among all the obligations defined for the use case, obligations related to this test care are:

   - O1: As a data controller, the hospital needs to provide a policy on what data is collected and for what purposes

   - O2: As a data controller, the hospital must ask the data subjects (patients) explicit consent for collecting and processing personal data

   - O4: As a data controller, the hospital must, upon request, provide evidence to the data subjects on their personal data processing activities

   Source of evidence is taken from logs where some of the events are generated leading to violation to one or some obligations. This is possible because the event generation has policy indicator that can control how the events are generated in relation to the policy. Explanation can be found in section 3.7. Event Generation. Example of source of evidence used in this test case is shown below.

```
@1397835131 accessDS (person_66)
@1397835132 DPLogAccess (person_66)
@1397835132 forwardDS (person_66)
@1397835133 DPLog (person_66)
@1397835133 DPLog (person_66)
@1397835133 RelativeAskConsentDS (person_15)
@1397835135 RelativeCollect (person_15)
@1397835135 DCLog (person_66)
@1397835135 DCLog (person_66)
@1397835135 deleteDS (person_66)
@1397835136 DCLog (person_66)
@1397835136 DCEvidReqReceived (person_66)
@1397835138 DPLog (person_66)
@1397835139 DPLogDelete (person_66)
@1397835288 askConsentDS (person_78)
@1397835291 DCProcess (person_78)
@1397835295 DCCollect (person_78)
@1397835412 deleteDS (person_59)
@1397835413 Notify (person_59)
@1397835413 DPLogDelete (person_59)
@1397835413 Notify (person_59)
@1397835414 DPLog (person_59)
@1397835415 DPLogDelete (person_59)
@1397835416 DCLog (person_59)
```

Running the evidence collection with MonPoly and Pyke, i.e. monitoring policies on source of evidence example, gives the output as following.

```
Obligation_4, @1397835136 DCEvidReqReceived (person_66)
Obligation_2, @1397835295 DCCollect (person_78)
```

Verification process on the potential evidence results in evidences showing as proof that Data Controller has violated obligation O4 and O2. In the example, trigger event for collecting evidence is not triggering the related action event to be executed which results in violation to obligation O4. On the other hand, trigger event for collecting personal data is also not triggering related action event to ask for data subject consent which also results in violation to obligation O2.

2. Detecting if Data Processor (Cloud Provider) does not follow obligations on processing Data Subject (Patient)'s personal data

   In this test case, several obligations defined for the selected use case are related. Among all the obligations defined for the use case, obligations related to this test case are:

- O5: As a data processor, the primary service provider must log all access to personal data
- O6: As a data processor, the primary service provider must, upon request, provide evidence to the data controller (hospital) on its personal data processing activities
- O7: As a data processor, the primary service provider must, upon request, provide evidence to the data controller (hospital) on the correct and timely deletion of personal data

The same as previous test case, example of source of evidence is collected from system logs where some events are generated to violate one of some obligations. These violations are to be detected by the approach or tool used in this work. Example of source of evidence used for this test case is shown below.

```
@1397835131 accessDS (person_66)
@1397835132 forwardDS (person_66)
@1397835133 DPLog (person_66)
@1397835133 DPLog (person_66)
@1397835133 RelativeAskConsentDS (person_15)
@1397835135 RelativeCollect (person_15)
@1397835135 DCLog (person_66)
@1397835135 DCLog (person_66)
@1397835135 deleteDS (person_66)
@1397835136 DCLog (person_66)
@1397835136 DCEvidReqReceived (person_66)
@1397835138 DPLog (person_66)
@1397835138 DCEvideCollect (person_66)
@1397835139 DPLogDelete (person_66)
@1397835288 askConsentDS (person_78)
@1397835291 DCProcess (person_78)
@1397835293 askConsentDS (person_78)
@1397835295 DCCollect (person_78)
@1397835412 deleteDS (person_59)
@1397835413 DPLogDelete (person_59)
@1397835414 DPLog (person_59)
@1397835415 DPLogDelete (person_59)
@1397835416 DCLog (person_59)
```

Running the evidence collection with MonPoly and Pyke, i.e. monitoring policies on source of evidence example, gives the output as following.

```
Obligation_5, @1397835131 accessDS (person_66)
Obligation_7, @1397835413 DPLogDelete (person_59)
Obligation_7, @1397835415 DPLogDelete (person_59)
```

Verification process on the potential evidence results in evidences showing as proof that Data Controller has violated obligation O5 and O7.

3. All obligations defined are followed by every entity in the cloud system which means there is no violation detected

   Several test cases have been executed using the evidence collection tool proposed in the framework on detecting violations to policies. This test case, however, tries to catch a scenario when there is no violation to policies i.e. all entities involved in the service chain are complied with all obligations set up for the system. A result of no evidence collected is expected as the output from the evidence collection tool to confirm that the tool works for both cases i.e. when there are violations and when there is no violation.

   Source of evidence is collected from system logs where previously event generator is used to execute events on the system by following all obligations defined for the use case. Collected source of evidence used in this test case is therefore looks like as following (example)

```
@1397835131 accessDS (person_66)
@1397835132 DPLogAccess (person_66)
@1397835132 forwardDS (person_66)
@1397835133 DPLog (person_66)
@1397835133 DPLog (person_66)
@1397835133 RelativeAskConsentDS (person_15)
@1397835135 RelativeCollect (person_15)
@1397835135 DCLog (person_66)
@1397835135 DPLog (person_66)
@1397835135 deleteDS (person_66)
@1397835136 DCLog (person_66)
@1397835136 DCEvidReqReceived (person_66)
@1397835137 DCEvidCollect (person_66)
@1397835138 DPLog (person_66)
@1397835139 DPLogDelete (person_66)
@1397835288 askConsentDS (person_78)
@1397835291 DCProcess (person_78)
@1397835293 askConsentDS (person_78)
@1397835295 DCCollect (person_78)
@1397835412 deleteDS (person_59)
@1397835413 Notify (person_59)
@1397835413 DPLogDelete (person_59)
@1397835413 Notify (person_59)
@1397835414 DPLog (person_59)
@1397835415 DPLogDelete (person_59)
@1397835416 DCLog (person_59)
```

   Running the evidence collection with MonPoly and Pyke, i.e. monitoring policies on source of evidence example, gives null output. This means that there is no violation detected for any policies. This result matches with the scenario used where events are generated by taking information that all entities follow all related obligations.

4. Modifications to collected evidence will result in unsuccessful evidence verification

This test case is executed for testing verification process of evidence resulted from collection process. As explained, the verification process is intended to verify if the collected evidence is valid in accordance to the source of evidence taken into processing and the policies defined for them. Modification to evidence, addition of information or deleting of evidence should result in invalid evidence.

For testing purpose, the result from a running of collection process is collected and modified. For example, the result of collection process is as following

```
Obligation_4, @1397835136 DCEvidReqReceived (person_66)
Obligation_2, @1397835295 DCCollect (person_78)
```

With source of evidences shown below

```
@1397835131 accessDS (person_66)
@1397835132 DPLogAccess (person_66)
@1397835132 forwardDS (person_66)
@1397835133 DPLog (person_66)
@1397835133 DPLog (person_66)
@1397835133 RelativeAskConsentDS (person_15)
@1397835135 RelativeCollect (person_15)
@1397835135 DCLog (person_66)
@1397835135 DCLog (person_66)
@1397835135 deleteDS (person_66)
@1397835136 DCLog (person_66)
@1397835136 DCEvidReqReceived (person_66)
@1397835138 DPLog (person_66)
@1397835139 DPLogDelete (person_66)
@1397835288 askConsentDS (person_78)
@1397835291 DCProcess (person_78)
@1397835295 DCCollect (person_78)
@1397835412 deleteDS (person_59)
@1397835413 Notify (person_59)
@1397835413 DPLogDelete (person_59)
@1397835413 Notify (person_59)
@1397835414 DPLog (person_59)
@1397835415 DPLogDelete (person_59)
@1397835416 DCLog (person_59)
```

If the result is modified to

```
Obligation_5, @1397835131 accessDS (person_66)
Obligation_4, @1397835136 DCEvidReqReceived (person_66)
Obligation_2, @1397835295 DCCollect (person_78)
```

The verification process run against the result will output that first entry of the collected result is not valid according to given source of evidence and set of policies. Detail of verification process can be found in section 3.16. Verification Process.

```
Not valid
Valid
Valid
```

4.6. Comparison

The main difference between approach taken in MonPoly and Pyke is the support for temporal logic. In MonPoly, the logic used to express the obligations is metric first-order temporal logic which support temporal expression. With this logic, temporal relationship can be expressed. Accountability obligations sometimes need this to express temporal relationship between events. On the other hand, Pyke uses Prolog which is pure logic language that does not support temporal logic. Using Prolog in expressing obligations limits the expressiveness of the obligations since there is no support for expressing temporal relationship. This support gives effect to the performance and practicality in terms of increasing data size. In metric first-order temporal logic which is used in MonPoly, the policy monitoring to collect evidence is taking more times than processing Prolog as used in Pyke. Temporal logic processing gives more time on the overall processing. In addition to that, processing with Pyke which uses Prolog gives more practicality as it is built on Python framework where additional features can be easily added.

In Pyke, the mechanism used is to assert new facts into knowledge base every time there is a match on the checking performed on the existing fact with respect to the rules. Then, all these new facts are being verified to check the consistency with given information about the system. In MonPoly, the policy compliance is done on the fly on the source of evidence which can be considered as facts. The policy compliance is controlled by the definition of obligations as rules.

4.7. Scalability & Limitations

Methods proposed in the framework of evidence collection are tested in terms of scalability i.e. the ability to perform with larger data set. Method in the framework that is in interest for this test is the one used in the collection process. The data set in this case is the size of logs as source of evidence to be processed in the policy monitoring methods used in the collection process. As explained before, there are two approaches used in the collection process i.e. processing as MFOTL and as FOL. MFOTL processing is using MonPoly and FOL processing is using Pyke. The purpose of this is to see how those two approaches perform in relation to increasing data size.

Two main aspects are tried to be related and tested. They are size of data set and execution time in terms of collecting evidence from the data set. Size of data set is calculated as number of distinct events contained in the processed log. The test is performed on a PC with specification as following:

- Processor: Intel Core i7 @ 2.70GHz, 64-bit OS
- RAM: 8 GB

Test is conducted on several data set sizes. On each test, execution time for the collection process is recorded both for the first approach (using MFOTL) and second approach (using FOL).

Following Table 4.1. shows test result of collection process using MFOTL (with MonPoly) in relation to increasing data size. Number shown in the execution time is the average from execution time recorded from several runs for particular data size.

Size of logs (# events)	**(Average) Execution Time – collection with MFOTL (seconds)**
25000	**~ 1**
50000	**~ 1,7**
75000	**~ 2,5**
100000	**~ 3,5**
200000	**~ 5**

Table 4.1. Evidence Collection with MonPoly - Execution Time Table

The table values can be drawn in the graph as shown in following Figure 4.5.

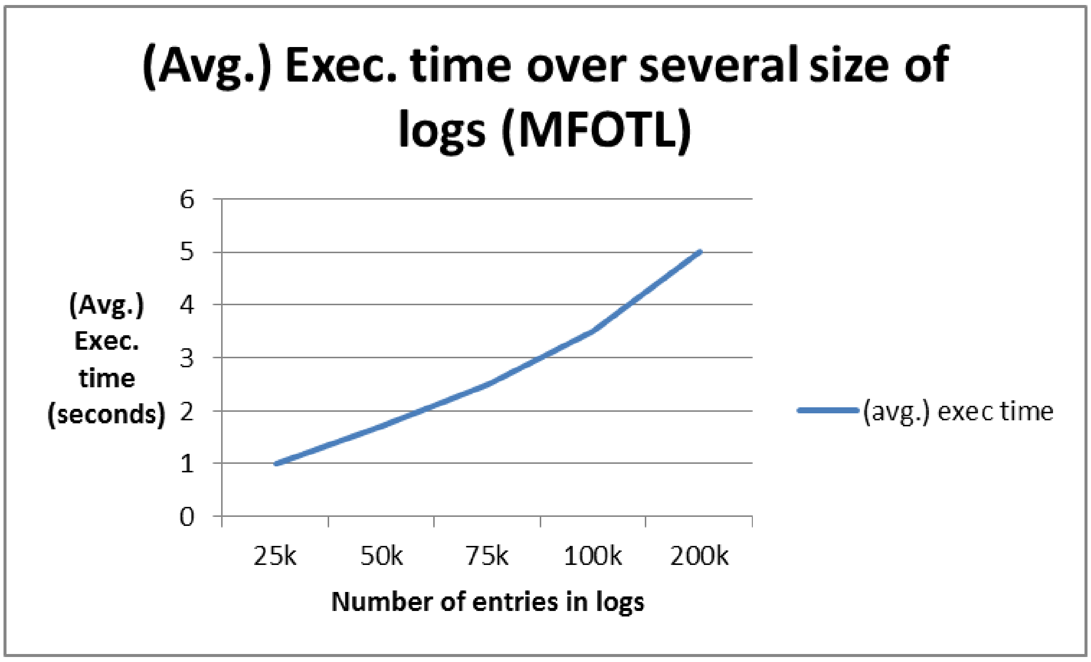

Figure 4.5. Evidence Collection with MonPoly - Execution Time Graph

Following Table 4.2. shows test result of collection process using FOL (with Pyke) in relation to increasing data size.

Size of logs (# events)	(Average) Execution Time – collection with FOL (seconds)
25000	~ 0.7
50000	~ 1,5
75000	~ 2
100000	~ 3,1
200000	~ 4

Table 4.2. Evidence Collection with Pyke - Execution Time Table

The table values can be drawn in the graph as shown in following Figure 4.6.

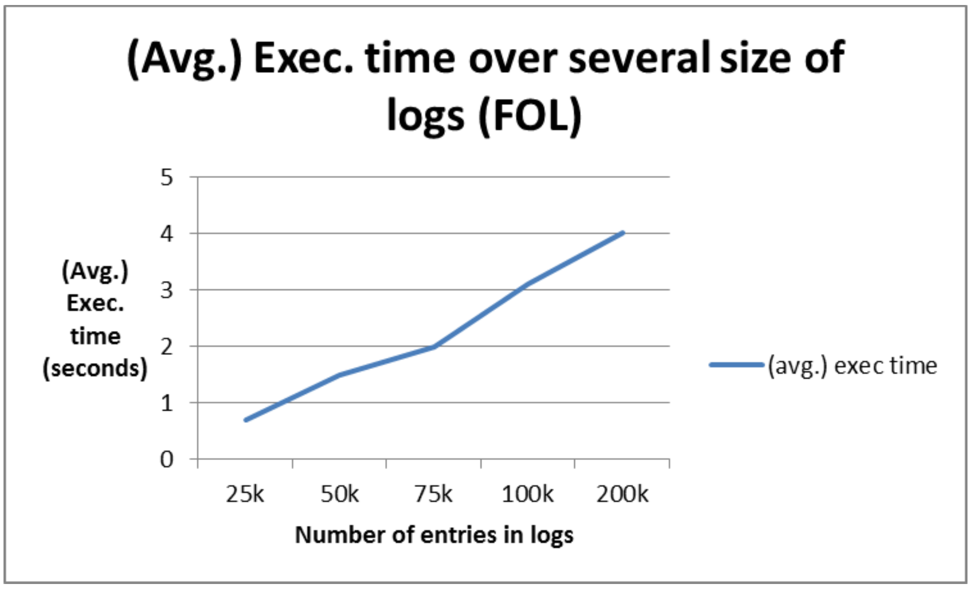

Figure 4.6. Evidence Collection with Pyke - Execution Time Graph

With the information from the test, the performance in terms of execution time of each approach can be concluded. Preliminary testing seems to indicate that the approach using FOL with Pyke gives better result in terms of execution time. This is because that the FOL processing does not involve temporal processing which gives simpler processing process. With the increasing data size which will be the case when the policy monitoring is performed on cloud service with several entities, FOL is more practical although not more expressive in expressing obligations.

Apart from the size of data set, the complexity of obligations to be expressed in the policy representation language affects the performance of the collection process in terms of time. It is reflected in the test result shown above where execution time of approach using MFOTL is higher than of approach using FOTL. MFOTL has more expressiveness than FOL as it supports temporal logic expression.

The collection processes proposed i.e. using MFOTL with MonPoly and using FOL with Pyke have limitations. MonPoly processes logs as source of evidence and obligations in its own format. This format more or less introduces additional processes needed to pre-process the input. In addition, some of the information contained in logs or obligations may be loss in collection process as they are not included. The same limitation also exists when using FOL with Pyke in the collection process.

As explained in section 3.12. A-PPL Translation, a translation from A-PPL to MFOTL (used in MonPoly) or FOL (used in Pyke), is needed. This translation introduces complexity in performing the whole collection process. With different set of events, obligations, and terms, the translation becomes more complex.

5. Conclusion & Future Works

Evidence collection has been identified as one of the key element to provide accountability in cloud service. A framework of evidence collection is proposed to collect evidence about events in the cloud service with relation to obligations defined for it.

The implementation of the framework of evidence collection is executed on a selected use case of cloud service. Healthcare service in the cloud is selected as the use case. In this use case, there are several entities involved in the cloud service. Data subject whose data is uploaded and processed in the cloud is the patient. Data controller is the entity which is responsible for the processing of data subject's data in the cloud, in this case the data controller is hospital. Patient's relatives are also considered as Data Controller. The last entity involved is Data Processor. Data Processor includes all cloud service providers that processes or stores Data Subject's data.

The framework of evidence collection consists of several steps. The steps are source of evidence identification and collection. After source of evidence is collected, they are pre-processed in preparation for the evidence collection process following it. In the evidence collection process, evidence is collected from the source of evidences based on the obligations that are defined for the cloud service. Any violations to the obligations are recorded as evidence for further audit process. Evidence collected is then verified with available information to test its validity.

In order to test the proposed method on the framework of evidence collection and on the selected use case, a test system is set up. The system simulates the selected use case i.e. healthcare service in cloud. All sources of evidences for processing are collected from a running of this system. This source of evidences is then used to be fed in as data for the evidence collection process.

The main element in the framework of evidence collection is evidence collection. This process collects evidence about events that break the obligations. Evidence collection is mainly processing source of evidences against obligations. The process involves logic processing. The obligations are represented in logical languages as rules and the source of evidences is represented as facts. Two approaches on evidence collection are used and compared. First approach processes obligations and source of evidences as temporal logic. It uses MonPoly tool which is policy monitoring tool process obligations in MFOTL (Metric First-Order Temporal Logic) formula. Second approach processes obligations and source of evidences as first order logic (FOL) in Prolog using Pyke tool.

Based on the work carried out in the A4Cloud, to express accountability policies, A-PPL (Accountable-PrimeLife Policy Language) language is used. This is the basis used in this Thesis to represent the obligations. Then translator to MFOTL is implemented to translate the A-PPL policies into processable format in the collection process.

Nomalization process on the collected evidence is discussed. Several basic and important information that must be tied to the evidence are listed. The implementation on framework of evidence collection's steps also involves putting the normalization in work i.e. normalized the collected evidence to the format specified.

Verification process is discussed. As one of the last steps in the framework, verification test validity of collected evidence based on the information about system's events and related obligations. This is the basis for taking the collected evidence for further processing.

One of the potential future works for the framework of evidence collection is to propose how the evidence is stored. The evidence also must contain chain of information that build up the information about events executed in the system. By proposing solid chain of information in the evidence, checking on the validity of the evidence in relation to, for example modification, is possible.

Another possible future work is to run the processes in the framework of evidence collection with big data processing concept. This means that the processes, for example evidence collection process, are performed using map reduce algorithm with several nodes of machines to achieve more scalable result. When the data size is huge, the usage of big data processing will reduce the time and resource needed for the process to complete.

References

[1] A4Cloud, “Identification of Evidence Types,” This work was supported by EU FP7 Accountability for Cloud and Other Future Internet Services (A4Cloud) Project, Grant No. 317550.

[2] D. Basin, F. Klaedtke and S. Muller, “Policy Monitoring in First-order Temporal Logic,” Department of Computer Science, ETH Zurich, Switzerland.

[3] R. Koymans, “Specifying Real-Time Properties with Metric Temporal Logic,” *Real-Time Systems,* vol. 2, no. Kluwer Academic Publishers, pp. 255-299, 1990.

[4] D. Basin, F. Klaedtke, S. Muller and E. Zalinescu, “Monitoring Metric First-order Temporal Properties,” ETH Zurich.

[5] Incident Management and Forensics Working Group, “Mapping the Forensic Standard ISO/IEC 27037 to Cloud Computing,” Cloud Security Alliance, June 2013.

[6] ISO 27037, “Guidelines for Identification, Collection, Acquisition and Preservation of Digital Evidence,” [Online]. Available: http://www.iso.org/iso/catalogue_detail?csnumber=44381, 2012.

[7] D. Garg, L. Jia and A. Datta, “A Logical Method for Policy Enforcement over Evolving Audit Logs,” CMU CyLab, May 2011.

[8] H. DeYoung, D. Garg, L. Jia, D. Kaynar and A. Datta, “Experiences in the Logical Specification of the HIPAA and GLBA Privacy Laws,” ACM, Carnigie Mellon University, 2010.

[9] C. Giblin, A. Y. Liu, S. Muller, B. Pfitzmann and X. Zhou, “Regulations Expressed As Logical Models,” *Legal Knowledge and Information Systems,* pp. 37-48, 2005.

[10] B. Schatz and A. Clark, “An Open Architecture for Digital Evidence Integration,” *AusCert Asia Pacific Information Technology Security Conference, R&D Stream,* May 2006.

[11] P. Turner, “Unification of Digital Evidence from Disparate Sources (Digital Evidence Bags),” *Digital Forensic Research Workshop,* 2005.

[12] M. Montanari, J. H. Huh, D. Dagit, R. B. Bobba and R. H. Campbell, “Evidence of Log Integrity in Policy-based Security Monitoring,” *IEEE,* 2012.

[13] D. Butin, M. Chicote and D. L. Metayer, “Log Design for Accountability,” *IEEE*

Security and Privacy Workshops, 2013.

[14] S. Trabelsi, A. Njeh, L. Bussard and G. Neven, “PPL Engine: A Symmetric Architecture for Privacy Policy Handling”.

[15] M. Henze, M. Grobfengels, M. Koprowski and K. Wehrle, “Towards Data Handling Requirements-aware Cloud Computing,” *IEEE on Cloud Computing Technology and Science,* 2013.

[16] C. A. Ardagna, L. Bussard, S. D. C. d. Vimercati, G. Neven, S. Paraboschi, E. Pedrini, F. S. Preiss, D. Raggett, P. Samarati, S. Trabelsi and M. Verdicchio, “PrimeLife Policy Language”.

[17] M. L. Krotoski, “Using Log Record Analysis to Show Internet and Computer Activity in Criminal Cases,” *United States Attorney's Bulletin,* vol. 59, no. 6, November 2011.

[18] Webopedia, “Cloud Service,” [Online]. Available: http://www.webopedia.com/TERM/C/cloud_services.html.

[19] Dialogic Corporation, “Introduction to Cloud Computing,” 2010.

[20] S. Pearson, “Toward Accountability in the Cloud,” *IEEE Internet Computing,* 2011.

[21] S. Pearson, V. Tountopoulos, D. Catteddu, M. Sudholt, R. Molva, C. Reich, S. F. Hubner, C. Millard, V. Lotz, M. G. Jaatun, R. Leenes, C. Rong and J. Lopez, “Accountability for Cloud and Other Future Internet Services,” *IEEE Cloud Computing Technology and Science,* 2012.

[22] Wikipedia, “Dropbox (Service),” [Online]. Available: http://en.wikipedia.org/wiki/Dropbox_(service).

[23] Dropbox, “Dropbox,” [Online]. Available: https://www.dropbox.com/.

[24] A4Cloud, “Policy Representation Framework,” This work was supported by EU FP7 Accountability for Cloud and Other Future Internet Services (A4Cloud) Project, Grant No. 317550.

[25] D. Basin, F. Klaedtke and S. Muller, “Monitoring Security Policies with Metric First-order Temporal Logic,” *SACMAT,* June 2010.

[26] Y. Venema, “Temporal Logic,” Royal Netherlands Academy of Arts and Sciences.

[27] D. Basin, F. Klaedtke, S. Muller and B. Pfitzmann, “Runtime Monitoring of Metric First-order Temporal Properties,” *Foundations of Software Technology and Theoritical Computer Science,* 2008.

[28] T. Rubsamen, C. Reich, A. Taherimonfared, T. Wlodarczyk and C. Rong, “Evidence for Accountable Cloud Computing Services”.

[29] E. Zalinescu, “MonPoly, A Monitor for MFOTL Specifications,” [Online]. Available: https://sourceforge.net/p/monpoly/monpoly/HEAD/tree/.

[30] D. Basin, M. Harvan, F. Klaedtke and E. Zalinescu, “MONPOLY: Monitoring Usage-control Policies”.

[31] P. Project, 2007. [Online]. Available: http://pyke.sourceforge.net/index.html.

[32] R. Natarajan, “20 Linux Log Files that are Located under /var/log Directory,” August 2011. [Online]. Available: http://www.thegeekstuff.com/2011/08/linux-var-log-files/.

[33] Stanford University, “Propositional Logic,” 2007. [Online]. Available: http://logic.stanford.edu/classes/cs157/2007/notes/chap02.html.

[34] J. Ouaknine and J. Worrell, “Some Recent Results in Metric Temporal Logic,” *Springer,* pp. 1-13, 2008.

[35] Y. Liu, “A Survey and Analysis on Metric Temporal Logic,” The University of Texas at Austin.

[36] F. Laroussinie, “Expressiveness of Temporal Logics,” CNRS & ENS Cachan, France.

[37] P. Hunter, J. Ouaknine and J. Worrell, “Expressive Completeness for Metric Temporal Logic,” University of Oxford.

[38] Y. Hirshfeld and A. Rabinovich, “Quantitative Temporal Logic,” Tel Aviv University.

[39] A. Pnueli, “The Temporal Logic of Programs,” University of Pennsylvania.

[40] S. Konur, “A Survey on Temporal Logics for Specifying and Verifying Real-time Systems,” *Front Computer Science,* 2012.

[41] G. Simko and J. Sztipanovits, “Active Monitoring Using Real-time Metric Linear Temporal Logic Specifications,” Vanderbilt University.

[42] V. Rybakov, “Linear Temporal Logic with Until and Next, Logical Consecutions,” *Annals of Pure and Applied Logic,* vol. 155, pp. 32-45, 2008.

[43] P. Schnoebelen, “The Complexity of Temporal Logic Model Checking,” *Advances in Modal Logic,* vol. 4, pp. 1-44, 2002.

[44] J. Ouaknine and J. Worrell, “On the Decidability and Complexity of Metric Temporal Logic over Finite Words,” *Logical Methods in Computer Science,* vol. 3, pp. 1-27, 2007.

[45] Wikipedia, “Prolog,” [Online]. Available: http://en.wikipedia.org/wiki/Prolog.

[46] M. A. Orgun and W. Ma, “An Overview of Temporal and Modal Logic Programming”.

[47] M. Abadi and Z. Manna, “Temporal Logic Programming,” *J. Symbolic Computation,* vol. 8, pp. 277-295, 1989.

[48] T. Hrycej, “A Temporal Extension of Prolog,” *J. Logic Programming* , vol. 15, pp. 113-145, 1993.

[49] M. A. Orgun, W. W. Wadge and W. Du, “Chronolog(Z): Linear-time Logic Programming,” *IEEE Computer Society,* pp. 545-549, 1993.

[50] H. Barringer, M. Fisher, D. Gabbay, G. Gough and R. Owens, “METATEM: An Introduction,” *Formal Aspects of Computing,* vol. 7, pp. 533-539, 1995.