THE ‘SCHEMA-LAST’ APPROACH: DATA ANALYTICS AND THE INTELLIGENCE LIFE-CYCLE

A thesis submitted in partial fulfilment of the the requirements of the

Degree of Doctor of Philosophy

Facutly of Education, Science, Technology and Mathematics

Neil Brittliff

May 2015

© Copyright by Neil Brittliff 2015
All Rights Reserved

FORM B

Certificate of Authorship of Thesis

Except where clearly acknowledged in footnotes, quotations and the bibliography, I certify that I am the sole author of the thesis submitted today entitled:

The 'Schema Last' Approach: Data Analytics and the Intelligence Life-Cycle

I further certify that to the best of my knowledge the thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

The material in the the thesis has not been the basis of an award of any other degree or diploma except where due reference is made in the text of the thesis.

The thesis complies with University requirements for a thesis as set out in Gold Book Part 7: examination of Higher Degree by Research Theses Policy, Schedule Two (S2).

_____

Signature of Candidate

_____

Signature of chair of the Supervisory Panel

_____

Date

Abstract

Discovering information and knowledge from big volumes of data is a problem that confronts many Intelligence Agencies within Australia and possibly throughout the world. Specifically, the Australian Crime Commission (ACC) was faced with the problem of how to deal with the large amounts of intelligence data it was receiving and to collate and analyze this data. Moreover, the existing approach of 'data cleansing' was proving to be ineffectual, inefficient and could no longer cope with increasing amounts of data the agency was receiving. A different approach had to be found to replace the previous solution that involved transforming the data into a form that could loaded into a highly structured relational system. In addition to the transformation, much of the data was discarded simply because the data did not comply with the schema definition.

The thesis will investigate existing methodologies or the 'Schema-First' Approach in relation to intelligence collation and analysis. The thesis will demonstrate that the 'Schema-Last' Approach in combination with the 'Big Data' platform could be applied when the data is retrieved and analyzed and not when the data is first ingested. It will be shown that the 'schema-last' allows for new and novel approach to entity resolution and data fusion.

The approach of fusing data where the format, structure or quality cannot be guaranteed is now a real issue with many government and private organizations. If the structure and quality can be guaranteed then there is no issue. However, in the case of the ACC this is not the case and organizations are compelled by law to hand over data extracts. In many cases, data obtained this way is often not easily processable or able to be fused with the data sets.

The 'schema-last' is a new approach to the way data is utilized within the Intelligence Life-cycle. This approach applies a schema to the data only when the data is required not when the data is first acquired. In addition, it will be shown how this approach provides

the foundation to explore, exploit, analyze and fuse data without losing any data integrity or provenance, and the approach is an improvement on existing similar approaches. It will also be shown how this approach can be extended to an ontological representation of the data and, like a schema, ontological structures could be applied when the data is analyzed not on data ingestion.

A new approach requires a new platform to store, retrieve the data to replace the relational normalized representation currently in use. It will be shown that the Big Data platform is not only used to store the large data volumes but provide the mechanisms to support the 'schema-last' approach for data analytic. Other potential solutions will be discussed and how these approaches would prove to be deficient compared to 'schema-last'. Finally, the 'Minerva' application utilizing the 'schema-last' approach proved to be a successful implementation. This resolved many of the data quality issues and enabled analytics to be performed against data that could not be processed by the previous relational based system. The proposed novel 'Schema-Last' model is validated against several possible implementations, however further work is required resulting from questions encountered in this research.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A number of interesting research *questions* with regard to data confronting the intelligence community, in particular the Australian Crime Commission (ACC) and other Australian Intelligence Agencies. The ACC recognizes the value of data (see Appendix B) and that both data science and data analytics provide the foundation for a successful investigation.

The agency currently has over 1200 distinct data collections. A collection is a request by the agency for an organization to supply a copy of a requested data set. The ACC has powers of coercion in that an organization is compelled to hand over the data otherwise face prosecution. Not all data is received by ACC's coercive powers. Often data is freely given by organizations to aid in investigations. There is nothing in the ACC coercive powers that allows the ACC to specify the format of the data. There are also budgetary constraints placed upon the ACC, which has led to an overall reduction in the ACC's workforce. The ACC challenge in regards to data processing, provides a fertile environment for the research investigation which motivated the research covered in this thesis.

The ACC established the Fusion Data Centre which is responsible for the collection and analysis of data received by the organization. The focus of this is to enhance the capability and productivity of the Fusion Data Centre which is seen as a strategic centre of excellence. This is confirmed by a media release which is as follows:

> "To truly have impact against serious and organized crime, we must first discover and understand the national and international picture of its networks,

> methodologies, and the full range of vulnerabilities it exploits. We must then translate this into effective responses.
>
> To do this, we need rich, contemporary, and comprehensive criminal intelligence. Building this intelligence picture and identifying organized crime trends and weaknesses is the Crime Commission's core business.
>
> Much of the Commission's intelligence is housed in our National Criminal Intelligence Fusion Capability, which brings together subject matter experts, analysts, technology and big data to identify previously unknown criminal entities, criminal methodologies, and patterns of crime (Australian Crime Commission, 2012)."

1.1 Motivation

A major challenge that faces all law-enforcement and intelligence-gathering organizations is accurately and efficiently analyzing the growing volume of data that relates to crime, which includes cyber attacks, money laundering and any form of criminal behaviour that can be classified as serious or organised. Data mining is seen as the tool that will enable criminal investigators (who may lack training in data mining and analysis) to explore large data sets as described by Hsinchun Chen, Wingyan Chung, Jennifer Jie Xu, Gang Wang, Yi Qin and Michael Chau (Chen et al., 2004).

Intelligence collection for strategic assessments in law enforcement has traditionally relied on the initiative and guile of the analysts. Furthermore, it has become self-evident that the insatiable desire for data to allow police forces, intelligence agencies and the government to make accurate and timely decisions has never been as important as it is today. Unless relevant data and intelligence is known to the analyst, it is possible that the end product could 'miss the point' or fail to identify new and potential threats to the community.

The plethora of potential data available to organizations such as the ACC now has the potential to flood the existing intelligence processing infrastructure. The problem has now entered the Big Data world and this presents new challenges to the agency. Furthermore,

the data ranges from pristine to messy and with the large amounts of data received by the ACC it is no longer possible to develop models to predict criminal behavior or recognise potential threats with the existing analytical approaches available to them.

However, with the large amounts of data introduces another problem, that is how to deal with messy data. Much of the data requires some form of cleaning to turn the original source ultimately into knowledge and as stated by Kenneth Neil Cukier and Viktor Mayer-Schoenberg:

> "We can learn from a large body of information things that we could not comprehend when we used only smaller amounts (Cukier and Mayer-Schoenberger, 2013)."

Finally, new techniques and methodologies need to be discovered to cope with the data and intelligence demands placed upon the ACC. It was apparent that the data **deluge** was not going to abate and the ACC must adapt or develop a novel approach to address data volume, variety and processing.

1.1.1 The Data Volume Challenge

With the ever-increasing volume of data available to the investigators and analysts it is important that the results are presented in concise fashion and that they are not overwhelmed by data (Darren and Choo, 2014).

As stated by Darren Quick (Trends and Issues in Criminal Justice published by the Australian Institute of Criminology):

> "(The increases in available data) gives rise to a variety of needs, including:

- A more efficient method of collecting and preserving evidence.
- A capacity to triage evidence prior to conducting full analysis.
- Reduced data storage requirements.
- An ability to conduct a review of information in a timely manner for intelligence, research and evidential purposes.

- An ability to archive important data.
- An ability to quickly retrieve and review archived data.
- A source of data to enable a review of current and historical cases (Darren and Choo, 2014)."

The 'Schema-Last' approach is a replacement for the Extract Transform Load used by many organizations to preprocess data prior to data analysis. The approach is now employed at the ACC to address the problem of providing intelligence to other organizations in a timely manner. In addition, there was no consistent view of the data within the organization. As Yang and Wu found:

> "Many techniques are designed for individual problems, such as classification or clustering, but there is no unifying theory. However, a theoretical framework that unifies different data mining tasks including clustering, classification, association rules, etc., as well as different data mining approaches (such as statistics, machine learning, database systems, etc.), would help the field and provide a basis for future research (Yang and Wu, 2006)."

The thesis will examine existing solutions and propose a new and novel approach to deal with the data volume and variety faced by many organizations.

1.1.2 The Data Value Challenge

Data is seen as an asset within an organization and, if exploited correctly, can be used to gain a competitive advantage over their rivals. In addition, data can be used to capture concepts

The purpose to cleanse or not cleanse can best be expressed by utilizing the **Beliefs**, **Desires** and **Intentions** (Bratman, 1999) methodology based upon Michael Bratman's theory of human practical reasoning (Castanedo, 2011). Michael Bratman's theory can be applied to data analysis where the data modeller should take the following into consideration:

Beliefs Beliefs provide the *inference rules* to process and manipulate the incoming data. These rules can be captured and reused for new data sets or reapplied to existing data sets.

Desires Desires represent how the data can best be used or processed. Desires represent the motivation behind the data and are generally expressed as an accomplishment. For example, this data can be used to determine if this person is 'on the move' or 'up to no good'.

Intentions Intentions represent the deliberate use of the data and how the data is to be used. For example, the data may be obtained so that it enhances the intelligence surrounding a particular criminal organization.

However, there is a cost to data cleansing and that is:

- Inconsistent processing where cleansing involves human assessment.
- The elimination or removal of unwanted tokens within a field can lead to the introduction of errors and inconsistencies.
- The introduction of human judgment which in turn may lead to errant assumptions that result in decisions based on false or misleading data interpretation.

Jimmy Lin and Dmitiry Ryaboy state:

> "That a major problem for the data scientist is to *flatten the bumps* as a result of the heterogeneity of data (Lin and Ryaboy, 2013)."

There are several big data implementations on how to fuse the data and deal with the potential variability of each data source. If there is only one data source then this is not an issue, however if data comes in a number of formats or various schema definitions there is a need to deal with this issue. Each data source may have a different format for the data, may not have split the 'name' into the constituent components such as first, middle and last names, may have a different number of fields than any other data source. The most important issue with data variability is to ensure the integrity of the data. What does this mean and how can data integrity be retained? If the data has a single field that contains the entire name it may not be possible to split the name into the constituent components of first, middle or last. For example the name, *David Jones* could mean the first name is *David* and the last name is *Jones* or that the first name is *Jones* and the last name is *David*. Perhaps further

examination of the data relating to the next few names within the data set may reveal that the *David* is the first name and *Jones* is the last name. It may not be impossible to draw any conclusion pertaining to the actual order of the names.

A more difficult example concerns the date of birth or any date within the data set. This is similar to the Y2K problem where the year was represented by two digits and the century part of the date is omitted. In addition, some nationalities represent the date with the month as the first significant part, followed by the day of the month and then the year. In Australia, most dates begin with the day of month followed by the month and then the year.

1.2 Thesis Questions

The thesis will focus primarily on the collation, process and analysis phase of the Intelligence Life-Cycle and dissemination process is not covered or questioned by the thesis. The thesis will examine the impact of **Big Data** on the Intelligence Life-Cycle and how existing processes can adapt to the torrent of data faces by the Intelligence community. The thesis will answer the following research questions:

1.2.1 Data Management and Stewardship

The thesis will address the following major questions pertaining to data management and stewardship:

- How best to *ingest* data received from external sources?
- How to process data in a timely manner?
- How to retain data *provenance* to ensure that all data can be traced back to the original source?
- How to ensure that data is not changed or that the data meaning is lost through modification and transformation?

1.2.2 Data Quality

The thesis will address the following questions pertaining to data quality:

- How to deal with data sets with messy or noisy data values?
- How to deal with data sets with no *identifiable* primary key?
- What if the time and man-power taken to *clean* and *collate* data exceeds the agency's processing capability?
- How to deal with data values that have an ambiguous value or meaning?

1.2.3 Data Fusion

The thesis will address the following questions pertaining to data fusion:

- How to provide *consistent* fused view between the data sets contained within the agency?
- How to fuse and analyze data on demand?
  - How to fuse unrelated eclectic data sources into a single coherent view. The **fused view** will provide a unique perspective of the data for further analysis.
  - How can the **fused view** be explored and exploited?

1.2.4 Data Processing

The Fusion Data Centre was created as part of a national initiative to improve data collection quality and intelligence. These three initiatives are part of a New Program Proposal (NPP) which led to the formation of the Fusion Data Centre. The thesis will demonstrate how this new approach to data fusion can achieve the following:

- Proactive discovery of unknowns.
- Real-time target monitoring and the alerting of target activity.

- Vulnerability identification of communities or individuals.

Initially, Big Data was not seen as technology that can be used to address these initiatives. However, the Advance Analytics Team began the process and research to determine how Big Data, Search and Match and Big Data Analytics can be used to address each of them. It took a number of years to build this capability and it was not clear at the outset what was the most appropriate approach to be taken to deal with the data volume.

1.2.5 Problem Statement

The ACC recognized in 2010 that there needs to be **fusion** capability whereby the agency can exploit the numerous data sets that the agency obtains through its coercive powers. Problem statements identified as recognized by the ACC which were assigned to the Fusion Data Centre can subdivided into a number distinct problem statements which are:

1. Lack of an agreed ideal end state for the fusion capability.

2. Lack of core data management function and data management regime around bulk data holdings.

3. Data entry functions are cumbersome and time consuming due to inflexible data structures. As a consequence this has reduced the ability to ingest new data sources due to large amounts of time and in turn may impede active operations.
   "Ingestion delays of several months can render the data obsolete. The backlog for data ingestion is growing rapidly and results in significant delays (Australian Crime Commission, 2012)."

4. Search and discovery capabilities are highly ineffective; due to lack of connectedness of data silos across different systems and networks. Early consideration of the key problems in the agency identified an inability to answer "what do we know?" and as a result, an **Advance Search Capability** was developed.

5. Excessive time spent collating data rather than spending time analyzing the data.

6. Identities are only able to be matched by converting data to a standard format across all data sources. There is an inability to handle *messy* data where the data was either poorly structured or contained a variety of data formats.

7. Lack of a single collaborative platform for discovery, collation and analysis of data holdings. The approach taken by many analysts is to use *Analyst Notebook* and *Microsoft Excel.* They have been the primary analysis tools used by analysts. These tools do not have access to all data sources available within the agency.

8. Lack of sufficient basic analysis tools available enterprise wide, including social network analysis (SNA), temporal data mining and Geo-spatial analysis.

9. Detection of previously unknown high risk entities is limited to data matching processes due to a lack of time contiguous data sets.

10. Collected data that is not managed according to an agreed process and security model.

11. Detection of previously unknown high risk entities is limited to data matching processes that cannot take advantage of the complete data sets.

12. Internal alerting capabilities where Persons Of Interest (POIs) can be monitored.

13. External alerting capabilities from partner agencies, will enable external agencies to have the ability to monitor POIs and report the results back to the ACC.

14. Improve real-time access to data. This also includes the ability of Social Network Analysis (SNA) to identify groups or cliques, identify network density and identify possible POIs.

15. Lack of capacity to develop advanced analytic tools. The NCTL (National Criminal Target List) and the validation of the ACC's TRAM (Threat Risk Assessment Model)

1.2.6 The ACC's Advance Analytics Section

The ACC in 2010 formed the Advance Analytics Section was set up to establish the capabilities as specified in the previous section. The section was primarily created to resolve these problems. It become apparent that not a single technology would solve all these problems or take a single approach. The Advance Analytics Section would investigate alternatives to the traditional Data Mining Techniques and be responsible for the capture, collection and collation of any data received by the ACC. The Advance Analytics Section would also offer advice on how best to deal with data in terms of storage and accessibility. The above problems motivated the research questions in this thesis (as acknowledged by the CIO at the ACC see Appendix A.2).

1.3 Hypothesis

The thesis will propose a new approach to minimise the need for data cleansing so that data does not have to undergo an extensive transformation process before the data is of any use. In addition, the 'Schema-Last' approach offers ontological support to allow for the creation of complex abstract models to enhance the description of the data. The hypothesis is that 'Schema-Last' is an effective tool in the pursuit of effective data models and *fuse* data sources.

In addition, no data or data source should be discarded. The premise of the 'Schema-Last' approach is to retain all data in its original state and that a Schema describes each data element and should be fluid enough that new knowledge pertaining to the data descriptions or *meta-data* can be reapplied to other data sources. Essentially, the 'Schema-Last' approach *schema* can evolve when new knowledge becomes available relevant to the data that the 'Schema' is representing.

Therefore a loosely specified 'Schema' or only a applying a 'Schema' when required is a superior approach to the more traditional rigid 'Schema' structure or *coercing* the data to fit the Schema.

1.4 Thesis Overview

The thesis will describe a model that can represent data and provide the platform for any future analytical process. The 'Schema-Last' approach consists of the following phases see (Figure 1.1):

Collating the process of collation and storage of data with reference to the Intelligence Life-Cycle.

Exploring the index strategies and how the 'Schema-Last' models capture the structure and meaning of the data. How these models can be used to explore and exploit the data.

Matching the process of the application of algorithms to match entities from different data sources.

Fusing the fusion or reduction of one or more data sources.

Analyzing the final phase where the models and data mining techniques can be applied on the fused or reduced data sources.

Finally, a case study which confirmed the 'Schema-Last' approach and this approach reduced the existing **back-log** of uncollated data. The case study utilized a formal nomenclature that describes the data structures and this applied to various phases of the Intelligence Life-Cycle.

1.4.1 Proposed Computational Architecture

The 'Schema-Last' Approach provides a new and novel approach to data collation and processing. The approach expands on existing approaches and can also take advantage of existing data mining algorithms. The approach defines the following:

- A non-prescriptive 'Schema' definition language;
- A reference implementation to collate raw data within a semantic data store;

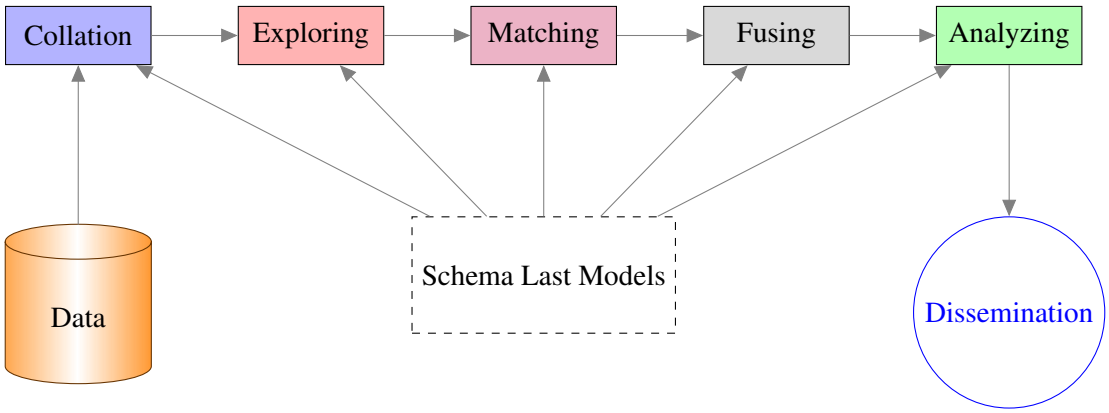


Figure 1.1: The 'Schema-Last' Approach architecture

- Indexing strategies based on the models and artefacts defined as part of the Schema-Last Approach;
- Algorithms that utilize the 'Schema-Last' Approach to combine multiple search results into a consolidated view;
- Data matching algorithms that utilize the 'Schema-Last' approach.

The thesis will show how the application of current technologies which include columnar data base implementation such as Cassanda and Apache HBase utilising triple store technologies which include Apache Jena can form the foundation for the 'Schema-Last' Approach. In addition, Apache Solr provides the capability to effectively index and explore the raw data. The thesis will explore a potential extension to RDF path expression to process and store tabular data represented by the RDF list structure. In addition, the thesis will propose a **columnar** implementation of the RDF triple store to support the 'Schema-Last' Approach.

1.4.2 Current Practices within the 'Intelligence Life Cycle'

The 'Schema-First' Approach as described in Chapter 4 is the dominate technology for the collation and initial processing of raw data. The assumption made by many data miners

that data is clean and ready for processing. As stated by Tamraparni Dasu and Theodore Johnson:

> "Current books on data mining and analysis usually focus on the last stage of the analysis process (getting the results) and spend little time on how the data exploration and cleaning is done. Usually, their primary aim is to discuss the efficient implementation of the data mining algorithms and the interpretation of results. However, the true challenges in the task of data mining are:
>
> - Creating a data set that contains the relevant and accurate information and
> - Determining the appropriate analysis techniques.
>
> In our experience, the tasks of exploratory data mining and data cleaning constitute 80% of the effort that determines 80% of the value of the ultimate data mining results ... However they assume that the data has already been gathered, cleaned, explored and understood (Dasu and Johnson, 2003)."

The current practice is to *clean* or *scrub* the raw data or assume the data is valid. The 'Schema-Last' Approach accepts the 'as is' and applies models upon the data. The data indexes represent the cleaned data but there is never any modification of the raw data.

1.5 Thesis Organization

The remainder of the thesis is as follows: The Intelligence Life-Cycle and Big Data technology, in particular the consequence of the Big Data 'revolution' has on the collection and processing in both Chapter 2 and Chapter 3.

Chapter 4 will survey the current modeling techniques ('Schema-First' Approach) and how these techniques influence data quality. The Volume Challenge will then describe the 'Schema-Last' approach and how this this approach is different from the 'Schema-First'.

The next chapter, Chapter 5 chapter will also describe the language and nomenclature used to express the Schema in 'Schema-Last. In addition Chapter 5 will identify the rules, the artefacts and the relationships between the artefacts that comprise the 'Schema-Last'

Approach. The chapter will also describe how the 'Schema-Last' Approach can be applied to ontological structures.

Chapter 6 will show how best to explore **Big Data** through the use of **Big Indexes** and how the 'Schema-Last' Approach can be applied to determining the optimum indexing strategy.

The next chapter - Chapter 7 - will then expand on Chapter 4 on how entity resolution is supported by the 'Schema-Last' Approach and how data matching algorithms can benefit from the 'Schema-Last' Approach.

Data fusion will be discussed in Chapter 8 and how data fusion and data reduction can utilize the concepts described in Chapter 5 and Chapter 6. In addition, how the 'fused' data sources can be be presented as a single coherent view to support advance searching capabilities.

Chapter 9 will describe how the ACC benefited from the 'Schema-Last' Approach and how other Intelligence Agencies have expressed a desire to utilize this modeling technique for their data sources. This chapter will examine existing triple store implementations and how this compares with three reference implementations. The third reference implementation utilized the Apache Jena ARQ as a framework to provide the SPARQL support. Three strategies are proposed in the thesis and each strategy is evaluated.

The final chapter - Chapter 10 - will propose extensions to the modelling techniques described in this thesis and how other disciplines would benefit from the 'Schema-Last' Approach.

Chapter 2

The Intelligence Life-Cycle

2.1 Big Data-Driven Intelligence

Intelligence is an integral part of the ACC remit and is used to identify and monitor new criminal and existing known targets. The intelligence cycle is the process of developing unrefined data from multiple data sources then analyzing the 'fused' data sources. The ACC and many other law enforcement agencies see that **Big Data** enables the collection to store and process data at a unprecedented rate that is only going to increase. An integral process of the intelligence cycle is the collection and processing of raw data. In addition, the scale, complexity and the changing nature of intelligence data can make it impossible to stay in front without the aid of technology to collect, process and analyze big data. As stated by the Joint Committee on Law Enforcement:

> "The ACC itself serves as the nexus between Australia's law enforcement, policing and national security agencies by facilitating the flow of criminal intelligence across these domains. As criminal intelligence is the 'core business' of the ACC, it uses a range of methods to collect, use and share criminal intelligence drawing on a variety of sources including law enforcement, policing, national security, government and private sector bodies and its own investigations of organized criminal activity. It coordinates national information sharing and, while emphasizing the importance of working in partnership to derive

intelligence, the ACC also has the power to conduct its own operations and investigations (Joint Committee on Law Enforcement 2013)."

2.2 What is Intelligence

Intelligence is an essential component of law enforcement capability. It supports the decision-making process and provides the tool for law enforcement leaders to make accurate and timely decisions. Intelligence can be classified into four categories which are (Quarmby, 2004):

Basic (background) intelligence: (What has happened?) This type of product background intelligence, usually encyclopaedic in nature, provides a broad range of baseline information and intelligence. While not futures based, such products provide a useful historical start point for analysis of futures.

Current Intelligence: (What is happening?) Specific assessments related to the status and significance of an ongoing operational threat, event, environmental condition or indication of illicit activity. This is the who? what? when? how? of an assessed threat.

Warning Intelligence: (What may happen in the future?) This intelligence determines future threats for law-enforcement, intelligence agencies and policy makers to focus their attention on up and coming threats to the community.

Estimate Intelligence: (What could occur?) Estimate intelligence is an attempt to project probable future developments within the law enforcement environment. This includes an assessment of key change agents or drivers that may cause disturbance within the community.

Both Estimate and Warning Intelligence deal with mid or long range events where 'estimate intelligence' predicts future developments whilst 'warning intelligence' focus on potential threats to the community at large. The 'Schema-Last' Approach methodology provides the framework to collect, analyze and process the raw data collected as part of the Intelligence

Life-Cycle. Furthermore, it will be shown that the collection, collation and processing of data is critical to the production of intelligence products. The process can be described as *agile* and the products produced from one investigation can form part of a large investigation or in turn may result in new investigations.

Intelligence is fundamental to any investigation and data collections can drive the investigation in any number of directions. The role of data mining and in particular data analytics has become an integral part of any police investigation. Systems can be developed that continuously monitor data collections to alert law enforcement agencies of any activity that may cause harm to the community (Warning Intelligence).

2.3 The Intelligence Life-Cycle

The Intelligence Life-Cycle (see Figure 2.1) central focus is data and data exploitation. The Intelligence Life-Cycle begins with the identification of possible data source, the collection and collation of the data, the analysis and application of models upon this data, the production and dissemination of situation reports and finally an evaluation and review of the entire Intelligence Life-Cycle. However, the collation and processing phase of the Intelligence Life-Cycle as it will be shown must deal with:

- messy and noisy data;
- structured, semi-structured, and unstructured data;
- tabular and highly linked data.

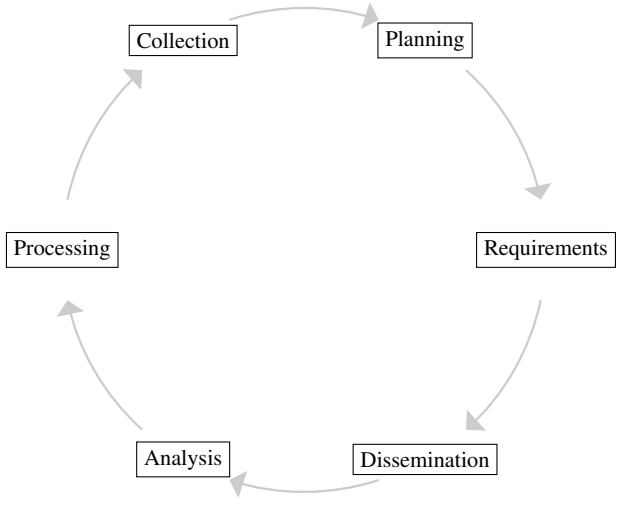


Figure 2.1: The Intelligence-Life Cycle

The Australian Criminal Intelligence Model described in Appendix C is used by most law enforcement agencies in Australia. The Australian Institute of Criminal Intelligence (Darren and Choo, 2014) describes the Intelligence Life-Cycle (shown in Figure 2.4) and identical to the Australian Criminal Intelligence Model. This 'Schema-Last' Approach supports the Australian Criminal Intelligence Model and provides the frame work to support this model, specifically the collation and processing phases.

2.4 Intelligence Collation and Collection

The collation of intelligence is central to the delivery of assessments that add value to a case or the profile of a person or organization. Effective intelligence relies on three criteria (Ratcliffe, 2008):

Reliable: It is essential to convince the clients that they can rely on the data which underpins the conclusions that are presented to them. That means that the process should be sufficiently comprehensive to convince the client that all the appropriate collection requirements, or questions, have been pursued and intelligence from all relevant sources gathered.

Valid: The collected data and intelligence may reliably reflect what is known but does this amount to a true or full understanding of the topic? An intelligence collection process becomes a self-fulfilling prophecy unless any limits on the capability to fill such gaps are acknowledged. A key element is the ability to recognize and place in context a single fragment of intelligence that provides insight ant that a vast array of data does not.

Timely: Even if an assessment is well founded it is of little use to decision-makers if it is not available at the time that a decision is required.

To satisfy the above approach, a system must be in place to enable the collected intelligence to be collated and processed with no interference to the data itself. If the process does pervert the data then the intelligence is unreliable.

The importance of developing suitable collection and collation capabilities are twofold:

- The development of a repository whereby existing knowledge is available to further an investigation. Without this knowledge all intelligence operations would require the commencement of a new or independent collection process.

- The data repository can identify gaps within the collection process.

The collection of intelligence is central to the delivery of assessments that are useful. Oliver Higgins (Higgins, 2009) identified two distinct approaches pertaining to intelligence collections: the bottom-up and top-down. The differences between the approaches are identified in Table 2.1.

The bottom-up approach is well suited to the intelligence and Knowledge Discovery Process (KDP) model which was first discussed in 1989. Different models by Fayyad (Fayyad et al., 1996) were suggested starting with a process model. The common factor of all data-driven discovery process is that knowledge is the final outcome of this approach. The process has defined four approaches (see Figure 2.2):

1. Traditional KDP approach. This approach is widely used by most KDP modelling innovators. Starting with KDP process modelling, many of KDP models used the same process flow including most of the following steps: business understanding, data understanding, data processing, data mining/modelling, model evaluation, and deployment/visualization.

2. Ontology-based KDP approach. This approach is the integration of ontology engineering and traditional KDP approach steps. Three directions were identified in this approach: Ontology for KDP, KDP for Ontology, and the integration of both previous directions .

3. Web-based KDP approach. This approach mainly deals with web log analysis. It is mainly similar to traditional KDP approach, but it has some unique steps to deal with *web transaction* logs.

4. Agile-based KDP approach. This approach is the integration between agile methodologies and KDP traditional methodologies.

Category	Top-Down	Bottom-Up
Characteristics	A generic and open-ended process whereby collection assets are actively directed to fill gaps in knowledge.	A passive process based on collation of information produced by everyday policing and from tactical collection plans.
Strengths	More dynamic and therefore timely. Able to throw light on unknowns. Provides answers to 'why' questions and not just inferences.	The quickest results by collating what is known rather than collating anything new. Provides insights about emerging trends and possibly identifies unknowns.
Weaknesess	Findings may lack validity due to lack of context and empirical content. Risk that the process may be self-fulfilling, collecting intelligence against a static set of requirements.	Risks that the process may be self-fulfilling - not suited to advancing understanding of known unknowns.
When to use?	Criminal's activity (including market indicators such as price), associations, attitude and capacity. Filling known unknowns. Indentifing unknown criminals.	To gain community intelligence or intelligence about prices in illegal markets at street level, new or changing popularity of criminal methods, suspects whose profile and impact are growing. This includes the correct identification of criminals and identifying criminal activity where the actual identification of the criminals has not been established.

Table 2.1: Typology of collection models

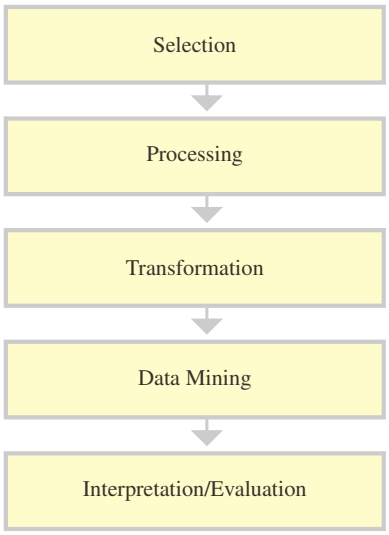


Figure 2.2: Knowledge Discovery Process

Whatever the approach, albeit traditional, ontology-based, web-based or agile the 'Schema-Last' Approach supports all four models. The role data as input to the Knowledge Discovery Feedback Loop (see Figure 2.3) drives the knowledge discovery process.

2.5 The Impact of Big Data on the Intelligence Life-Cycle

Intelligence and the information world is rapidly changing. Back in the 1970s and 1980s storage capacity was one of the metrics that altered the way both data is stored and retrieved. In fact, it was as though the data belonged to the organization and that the comparatively small data storage was sufficient to perform analysis and to draw conclusions (Setty, 2013). In turn this has affected the decision-making process in the ACC and many other intelligence agencies. Big Data Technology has the potential to make the decision-maker able to process huge data volumes that were not possible only ten years ago. The term 'Big Data' appeared for the first time in 1998 in a Silicon Graphics (SGI) slide deck by John Mashey with the title '**Big Data and the Next Wave of Infra Stress**'. The origin of the term 'Big Data' is due to the fact that individuals and organizations are creating a huge amount of data every day and this trend is not likely to decline anytime soon.

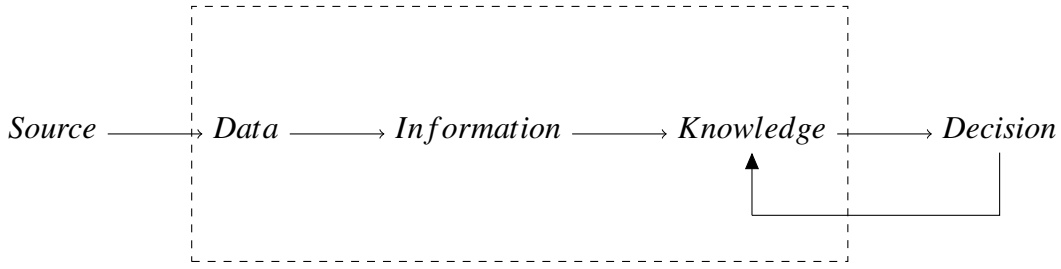


Figure 2.3: Knowledge Discovery Feedback Loop (Alnoukari and Sheikh, 2012)

Alnoukari and Sheikh (Alnoukari and Sheikh, 2012) defined knowledge discovery as concerns with the entire knowledge extraction process, including how data is stored and accessed, how to use efficient and scalable algorithms to analyze massive data-sets, how to interpret and visualize the results, and how to model and support the interaction between human and machine. Therefore, the impact of 'Big Data' has meant that the available data has had a dramatic impact on how data is gathered, stored and analyzed. New tools such as Hadoop were developed to process and analyze these large data stores. 'Big Data' has now meant that records are no longer required to be deleted and that archived data is now available on nearline storage (storage that is quickly accessible by a robot but lacks the performance of online storage) rather than relegated to tape backups that are difficult to retrieve and process.

2.5.1 Technology impact on the Intelligence Life-Cycle

The Intelligence-Life cycle is not only driven by government policy, knowledge gaps, but also driven by technology. The collation phase is restricted by the available storage and collation processing tools. 'Big Data' is an emerging technology that can store and process large data volumes as part of the collation phase. 'Big Data' is more than a marketing term but is used to classify type of data storage whereby the storage demands exceed traditional technologies. Usually, the traditional technology refers to relational databases such as IBM's DB2, Oracle's DBMS or Microsoft's SQL server and these products cannot meet the demands of data volume. Data is an integral part of the collection and collation phase

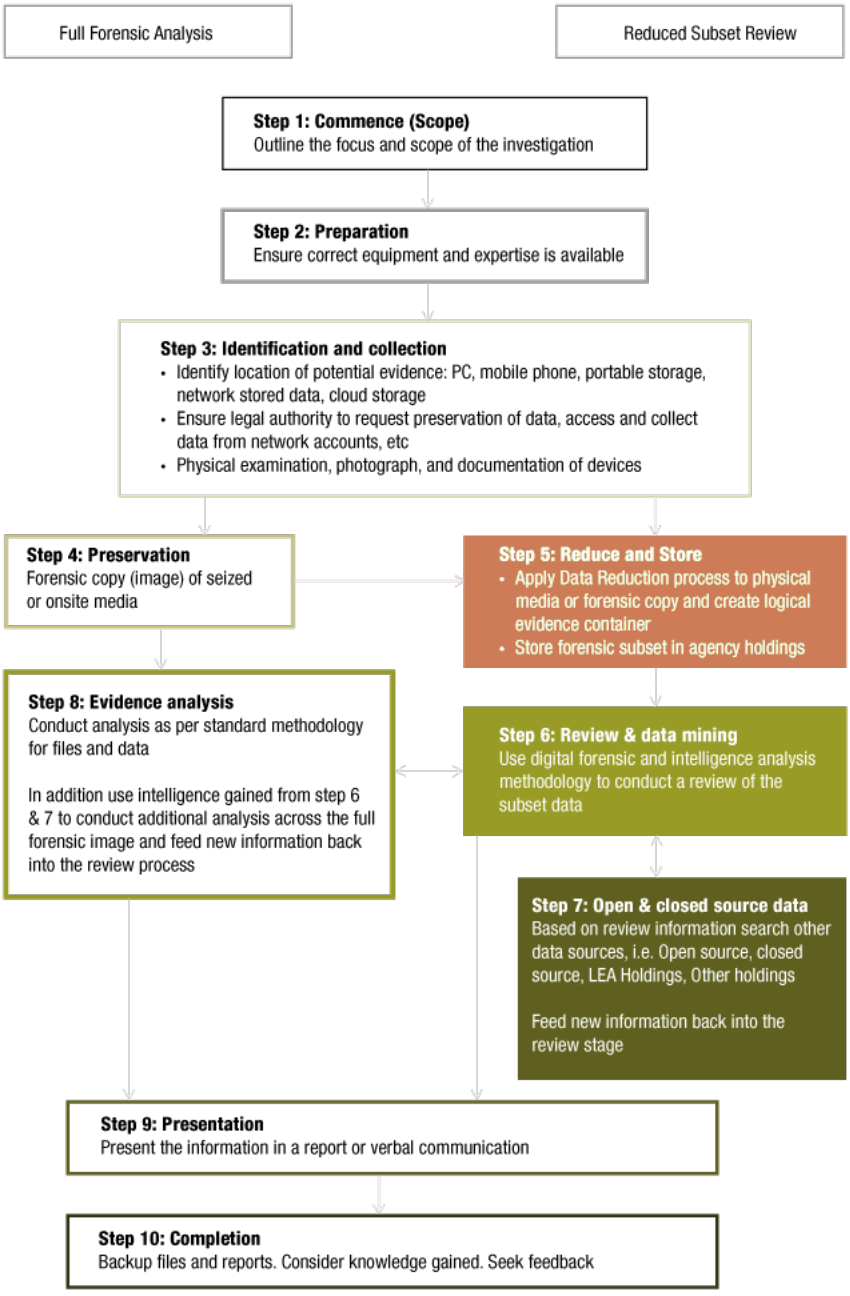


Figure 2.4: Intelligence-Life Cycle (Darren and Choo, 2014)

of the Intelligence Life-Cycle and this has meant new techniques are required to process this torrent.

> "We resist giving a concrete definition since that would limit it. But basically, it refers to the idea that we have so much more information these days that we can apply new techniques to it, to spot useful insights or unlock new forms of economic value. There are things we can do with a large body of data that we simply could not do when it was in smaller amounts. In our book, we identify three features: more, messy and correlations (Dumbill, 2012)."

Another factor is that the cost of storage has reduced significantly and in some way technology has driven the collection phase of the intelligence. Moore's law can be summarized as:

> "In a nutshell, Moore's law says that every two years, computer capacity (memory, speed and so on) increases by a factor of 2. How does this apply to 'Big Data'? It seems like big data is also growing exponentially, nobody will contest this statement (Dumbill, 2012)."

As Shugart (Shugart, 2012) defines the space cost per year:

> "The data confirms it: there is a very strong exponential correlation in space/cost ratio (r=0.9916). Over the last 30 years, space per unit cost has doubled roughly every 14 months (increasing by an order of magnitude every 48 months). "
>
> The regression equation allowed Shugart to calculate the reduction of storage costs is defined:

$$cost = 10^{-2.052(year-1980)+6.304}$$

The significant reduction of cost pertains to storage, memory and the overall infrastructure. Furthermore, **cloud computing** has allowed law enforcement organizations to store their data in a cost effective manner and they are no longer required to purchase and operate their own computer infrastructure.

Date	Drive Description	Size (MB)	Cost	$GB
1980 July	North Star	18	$4,199.00	$233,000.00
1981 September	Apple	5	$3,500.00	$700,000.00
1981 November	Seagate	5	$1,700.00	$340,000.00
1981 December	VR Data Corp.	6.3	$2,895.00	$460,000.00
1983 December	Corvus	6	$1,895.00	$316,000.00
1983 December	Xcomp	10	$1,895.00	$190,000.00
1983 December	Corvus	20	$3,495.00	$175,000.00
1983 December	Davong	10	$1,650.00	$165,000.00
1983 December	Xcomp	16	$2,095.00	$131,000.00
1984 March	Percom/Tandon	5	$1,399.00	$280,000.00
1984 May	Tecmar	5	$1,495.00	$299,000.00
1984 May	Corvus	11	$2,350.00	$214,000.00
1984 May	CTI	11	$1,995.00	$181,000.00
1984 May	Davong	10	$1,645.00	$165,000.00
1984 May	Pegasus (Great Lakes)	23	$1,845.00	$80,000.00
1985 July	First Class Peripherals	10	$710.00	$71,000.00
1987 October	Iomega	20	$1,199.00	$60,000.00
1989 March	Western Digital	40	$1,199.00	$36,000.00
1995 January	Seagate	2,900	$2,899.00	$990.00
1996 June 10	Western Digita	1,600	$399.99	$295.00
1996 August 14	IBM	1,760	$379.99	$263.00
1996 September	Quantum	3,200	$469.00	$173.00
1997 August 24	Western Digital	2,100	$279.99	$153.00
1997 December 3	Maxtor	7,000	$579.99	$95.30
1998 January 16	Quantum	6,400	$479.99	$86.30
1999 February 27	Quantum	19,200	$512.46	$30.70
1999 May 27	Fujitsu UDMA	17,300	$369.00	$24.50
1999 December 1	Quantum IDE	18,200	$348.00	$22.00
2000 February 1	Fujitsu	27,300	$375.00	$15.80
2001 April 25	Fujitsu 5400 Rpm UDMA-100	40,000	$199.00	$5.71
2002 September 20	Western Digital 7200 Rpm Ultra ATA-100	60,000	$139.99	$2.68
2003 November 29	Maxtor 7200 Rpm IDE	120,000	$144.88	$1.39
2004 December 4	Barracuda 7200RPM, Internal ATA/100	400,000	$280.00	$0.70
2005 December 12	Hitachi Deskstar 7K250 250GB	250,000	$130.00	$0.52
2006 December 27	Samsung 80GB	80,000	$35.00	$0.44
2007 June 24	Seagate 250GB	250,000	$100.00	$0.40
2008 January 13	Beyond Micro Monster Mobile 1TB	1,000,000	$270.00	$0.27
2009 July 24	HITACHI 0A38016 7200 RPM SATA 3.0Gb/s	1,000,000	$74.99	$0.07

Table 2.2: Cost per gigabyte over time for ‘small systems’ (Shugart, 2012)

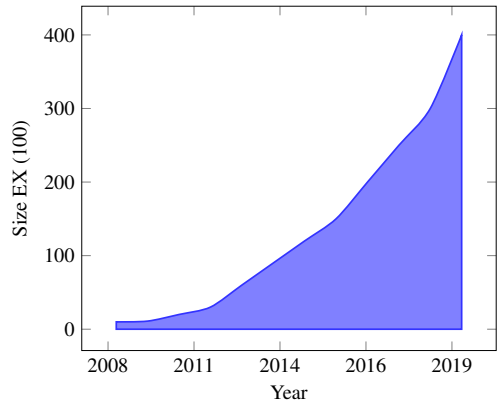


Figure 2.5: Cost of storage (Shugart, 2012)

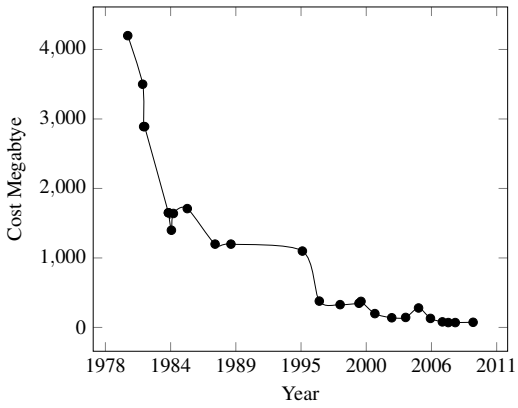


Figure 2.6: Cost per gigabyte over time (Shugart, 2012)

2.5.2 Cloud Storage and the Intelligence Life-Cycle

'Big Data' has driven the adoption of cloud solutions for many law enforcement organizations. Cloud computing offers a cost-effective way to support 'Big Data' technologies and the advanced analytics applications that can drive the intelligence Life-Cycle value. There are security concerns with storing data in the cloud, however the Australian Attorney General's Department have commenced the process in establishing a protected cloud network to allow agencies such as the ACC to store and process data within a secure cloud environment. As yet there is no firm Australian Government policy on what data can be stored within a cloud environment or the required classification.

Cloud offerings provide the government a unique opportunity to store endless amounts of data and not be concerned with the day-to-day management of the data. There is a symbiotic relationship between big data and cloud and David S. Linthicum describes this:

> "The use of big data technology, such as those that leverage map-reduce, will provide the most bang for the buck on cloud-based platforms. Indeed, I would declare that big data is clearly one of the first killer applications for PaaS (Platform as a Service) and IaaS (Infrastructure as a Service) public cloud computing platforms (Linthicum, 2013)."

With the ever increasing demand placed upon an organisation's infrastructure as a result of big data storage the cloud offers an alternative storage capability that traditional information technology infrastructures can no longer support.

2.6 'Data Variety' within the Collation Phase

Variety or variability of data poses one of the greatest challenge to any 'Big Data' implementation. If there is only a single data source then this is not a problem and is easy to deal with. However, if there is a wide variety of data sources all of which have different formats then this can pose a real challenge for any organization. Data Variety could pose the greatest problem faced as a Survey in 2014 of 100 data scientists (see Figure 2.7) has identified. The problem is variety as stated by Thor Olavsrud:

"A survey of (100) data scientists finds that a majority of them believe their work has grown more difficult as a result of the rapidly increasing variety of data sources they need to draw upon, and nearly a quarter feel Hadoop is not suited to the analytics they need to perform (Olavsrud, 2014)."

2.6.1 What is Data?

The concepts datum, designation, and value are defined as (ISO/IEC, 2012):

datum: designation whose concept is a value.

designation: association of a concept with a signifier that denotes it.

value: concept with a defined notion of equality for it.

A designation is a general notion, as there are special kinds depending on the subject field. In terminology, there are terms and appellations. A term is a linguistic designation of a general concept, and an appellation is a linguistic designation of an individual concept . The subject field of data has its own kind of designation: a datum is a designation whose concept is a value. Sometimes, when using the idea of designations, people refer to the signifier as

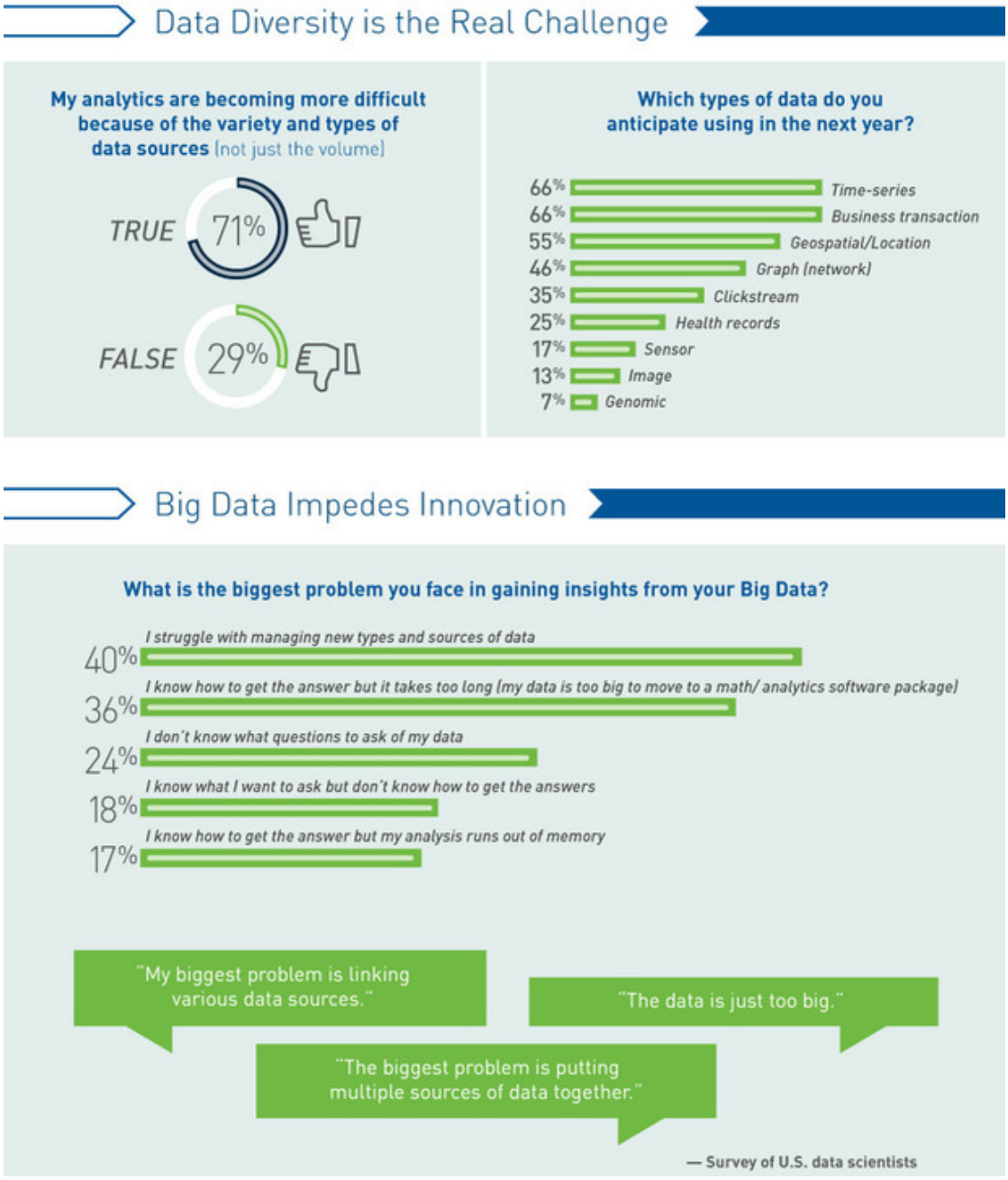


Figure 2.7: 'Big Data' variety survey (Olavsrud, 2014)

the designation, but this is incorrect. There are three basic parts to designations: the concept (which is a construct of the mind and can stand on its own), the signifier (which can stand on its own, for instance a symbol), and the designation (the association of the signifier to the concept for example: "this word-symbol-etc X means the concept "). Likewise, sometimes when using the idea of values, people refer to the signifier as the value, but this, too, is incorrect: the value refers to the concept portion of the datum and not its signifier. For example, one can easily discuss the notion of the value of seventeen-ness, a number, (a concept) independent of any particular signifier, which can be a numeral (a kind of signifier used to designate numbers).

2.6.2 The Messiness of Data

The rate of data growth world-wide is increasing at a rate never seen before:

> "From 2005 to 2020, the digital universe will grow by a factor of 300, from 130 exabytes to 40,000 exabytes, or 40 trillion gigabytes (more than 5,200 gigabytes for every man, woman, and child in 2020). From now until 2020, the digital universe will about double every two years (EMC, 2012)."

Therefore using all available data within the world is not a feasible proposition with the ever increasing volume of data coming into this world. The uses of data has also changed and no longer is data used for its intended purpose.

The value proposition for each data source can be quite different, as obtained by the ACC, for example, data sets may be classified as *low* or *high* signal data sources. Low signal data sources in themselves do not provide any useful indicators but could be used to confirm an entity's address, date of birth, property ownership and so on. An entity's membership of low signal data source in itself is not a form of intelligence. High signal data sources would be further analyzed and an entity's membership may indicate a potential threat or indicate unlawful activity that would require further analysis.

The data-gathering phase may be necessary because some of the data you need may never have been collected. Data can be acquired externally from public databases or internal databases that exist within the organization. Part of the data collection *phase* is to identify and record all the different data sources (Larose, 2014). This should include:

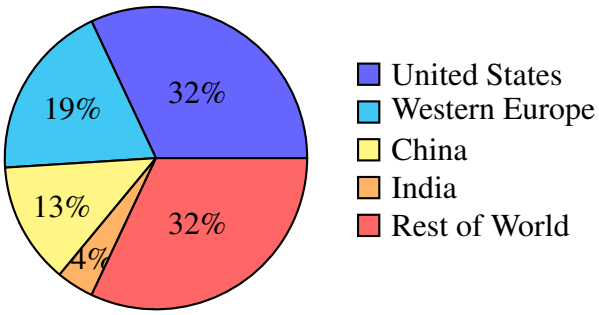


Figure 2.8: The geography of the digital universe - 2012 - (Gantz and Reinsel, 2012)

- Source of data could be from an external organization or from an internally generated from an application.
- Owner of the data.
- Person/organization responsible for maintaining data and ensuring the integrity of the data.
- How the data was obtained for example, was the data obtained under duress or an act of parliament or given freely to the organization.
- Cost of the data (if purchased).
- Storage organization (for example, Cassandra, HBase, Oracle database, VSAM file, etc.)
- Size in tables, rows, records, *et cetera*.
- Size in bytes of the data and number of records if the data is structured that way.
- Any special security requirements pertaining to data access.
- Restrictions or caveats on use.
- Any related data-sets.
- The location of the data or any geospatial reference.

- The time or validity period of the data.

The meta-data (data about data) can be stored with the data source or even separately. The notion that meta-data is 'data about data' is incomplete because meta-data can be descriptive data about things other than data, such as artefacts and hardcover books, is inaccurate (because all data is about some other data). The essential characteristics of meta-data include: it is descriptive data, and that it is descriptive about something. For example, if P is data and if P then Q represents the descriptive relationship such that P describes Q, then P is meta-data about Q. If there is no relationship from P to Q, then P is no longer meta-data, i.e., P is merely data, because meta-data is always relative to the object of description. Or, stated differently, P only becomes meta-data once its descriptive relationship to Q is established. The application of meta-data in reference to 'Schema-Last' Approach can be found in Section 5.4.10.

2.6.3 Data Munging

Data munging, or sometimes referred to as data wrangling, means taking data that is stored in one format and changing it into another format. Analysts regularly wrangle data into a form suitable for computational tools through a tedious process that delays more substantive analysis. There are tools both interactive and command line that can assist data transformation. Analysts must still conceptualize the desired output state, formulate a transformation strategy, and specify complex transforms.

> "Analysts regularly wrangle data into a form suitable for computational tools through a tedious process that delays more substantive analysis. While interactive tools can assist data transformation, analysts must still conceptualize the desired output state, formulate a transformation strategy, and specify complex transforms (Guo et al., 2011)."

Data munging is a part of the **collation** phase of the Intelligence **Life-Cycle**. Much time is consumed by the analysis to transform the data from the *raw* state to a more palatable state suitable for ingestion and for further processing.

2.6.4 Noisy Data

Generally, there are two types of noise sources (Wu, 1995): (a) attribute noise; and (b) class noise. The former is the errors that are introduced in the attribute values of the instances. There are two possible sources for class noise (Zhu et al., 2003):

contradictory examples: that is the same examples with different class labels.

***mis-classifications*:** instances labelled with incorrect classes or names.

Data is often delivered in idiosyncratic formats designed for human consumption. Often the data source itself may contain data that has no intelligence significance. However, once those data items are eliminated, if those data items did have any significant value, then this is lost. An organizational policy should not throw any data out no matter how irrelevant the data may seem at the time.

Another attribute of the data is the actual data format. The data format may actually be significant intelligence data. For example, the date format used by North Americas is in the form MM/DD/YYYY where the month precedes the day and year. In contrast, Australia adopted a different date format.

With the data sizes that agencies such as the ACC must deal with, noise in 'Big Data' is a real issue and any assumptions applied to the data's representation can be incorrect.

2.7 Data Dimensions

A data set can contain some of these characteristics if not all of them. It is important to determine what the data is and what does the data represent. Not all data sets fit neatly into one of the defined classifications. It is possible that the data sets have characteristics of both temporal, geospatial or graph.

2.7.1 Temporal

Data that is classified as a temporal adds a time dimension to each row within the data set. Temporal or time-series data represents the state of an 'object' at a specific point in time.

Generally, each record incorporates a single time stamp to identify the time the record was created, the time of the transaction, or any other temporal dimension.

2.7.2 Snap Shot

Unlike temporal data, all the data within the data set represents a single point in time. A series of snap shots adds the temporal dimension to the data in that each snap shot represents the state of the data for that time period. Usually meta-data is used to capture the time period of the snap shot.

2.7.3 Geospatial

This classification adds a geospatial dimension that is a longitude and latitude to each row in the data set. A number of visualization tools exist that can interpret the longitude and latitude coordinates and apply terrain images, polygons and overlay street maps and gazetteer data.

2.7.4 Graph/Semantic

This is perhaps the most difficult data to classify. Data stored in a relational data base contains some of the characteristics of graph data in that primary and foreign keys are used to represent a relationship between one or more rows in a data. This type of data classification enables the analyst to apply Social Network Analysis techniques upon the data.

2.7.5 Feed and Real-time

Feeds are becoming more common as a form of data acquisition. This is where data is streamed into the data-base in real time. Feeds usually represent a history transaction and each row within the feed is time-stamped.

2.8 Cross Industry Standard for Data Mining

The Cross Industry Standard for Data Mining mirrors the Intelligence-Life Cycle. Both frame works address the need to prepare and model data, understanding the business of the organization, the purpose of collecting and processing the data.

The Cross Industry Standard Process for Data Mining (CRISP–DM) states that a given data mining project has a life cycle consisting of six phases, That is, the next phase in the sequence often depends on the outcomes associated with the preceding phase. There may be further data preparation phase for further refinement before moving forward to the model evaluation phase. The six phases are as follows (see Figure 2.9):

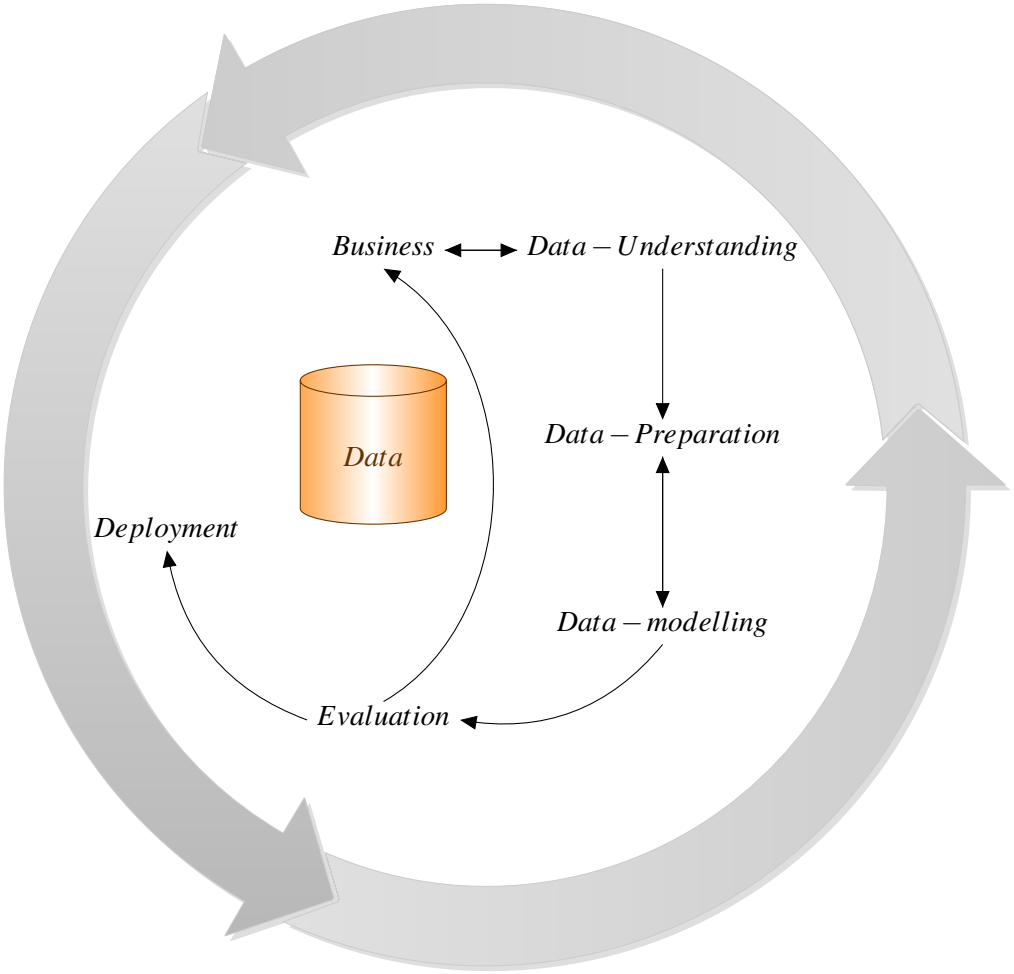
1. **Business understanding phase:** The first phase in the CRISP–DM standard process may also be termed the research understanding phase . Enunciate the project objectives and requirements clearly in terms of the business or research unit as a whole. Translate these goals and restrictions into the formulation of a data mining problem definition and prepare a preliminary strategy for achieving these objectives.

2. **Data understanding phase:** This is where the data is collected. Using exploratory data analysis to familiarize yourself with the data, and discover initial insights. Evaluate the quality of the data. and select interesting subsets that may contain actionable patterns.

3. **Data preparation phase:** This is the most labor-intensive phase and covers all aspects of preparing the final data set, which will be used for subsequent phases, from the initial, raw, dirty data. Select the cases and variables you want to analyze that are appropriate for your analysis. Perform transformations on certain variables, if needed and clean the raw data so that it is ready for the modelling tools.

4. **Modelling phase:** Select and apply appropriate modelling techniques and Calibrate model settings to optimize results. Often, several different techniques may be applied for the same data mining problem.

5. **Evaluation phase**: The modelling phase has delivered one or more models. These models must be evaluated for quality and effectiveness before we deploy them for use



(Guo et al., 2011)

Figure 2.9: CRISP-DM framework

Business Understanding	**Data Understanding**	**Data Preparation**	**modelling**	**Evaluation**	**Deployment**
Determine Business Objectives	Collect Initial Data	Select Data	Select modelling Techniques	Evaluate Results	Plan Deployment
Assess Situation	Describe Data	Clean Data	Generate Test Design	Review Process	Plan Monitoring and Maintenance
Determine Data Mining Goals	Explore Data	Integrate Data	Build Model	Determine Next Steps	Produce Final Report
Produce Project Plan	Verify Data Quality	Format Data	Assess Model		Review Project

Table 2.3: Generic tasks within the CRISP-DM reference model

in the field. Determine whether the model in fact achieves the objectives set for it in the initial phase and the determination whether some important facet of the business or research problem has not been accounted for sufficiently.

6. **Deployment phase:** Model creation does not signify the completion of the project. There is a need to make use of created models according to business objectives. An example of a deployment would be :

   (a) Generate a report for management or external partners.

   (b) Implement a parallel data mining process in another department or organization.

   (c) Implement a model to monitor any unusual activity.

The CRISP-DM tasks are shown in Table 2.3. The Intelligence Life-Cycle mirrors the CRISP-DM model in many ways. The business understanding task is analogous to the *requirements* and *collection* phase; the data understanding and data preparation task is performed as part of the *collation* and *process* phase.

Overall both methodologies complement each other, whilst the CRIPSP-DM reference model requires that the data description and summarization be a concise description of characteristics of the data, generally in an elementary and aggregated form. This form is carried over to the modelling, evaluation and final deployment phases. The Data description

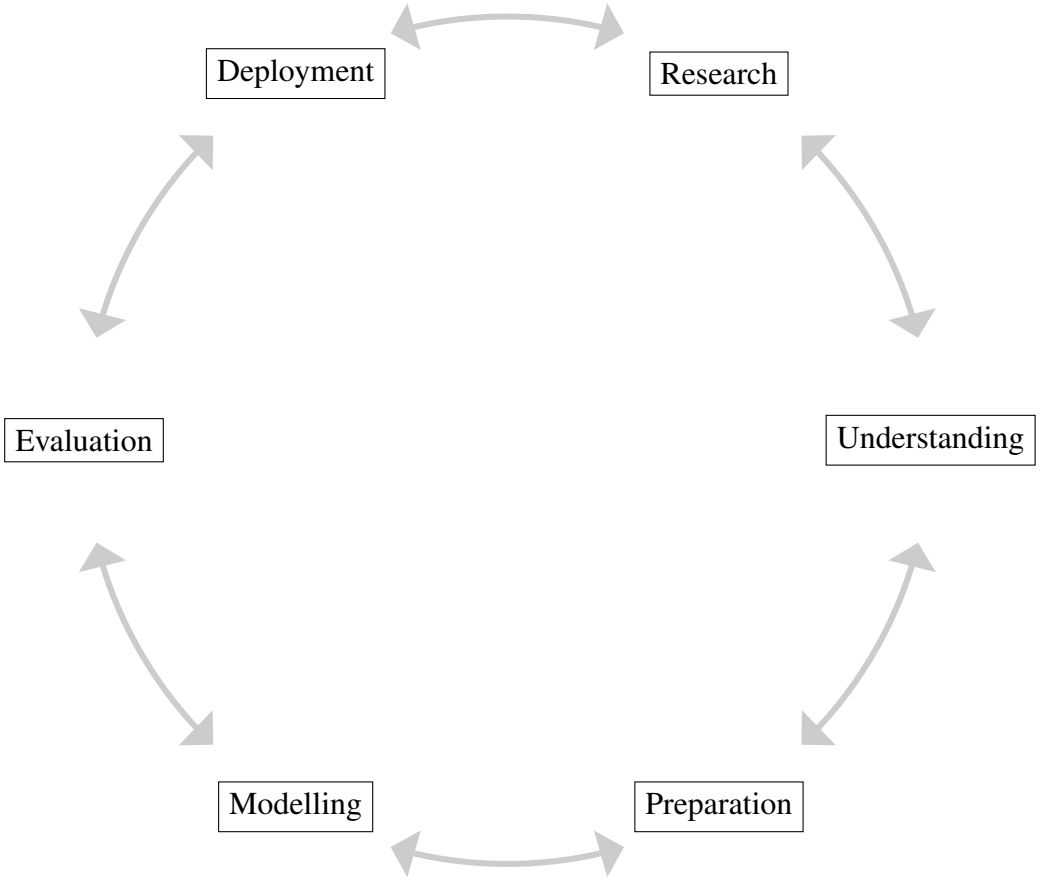


Figure 2.10: CRISP-DM phases

and the summarization are usually known before hand and form input to the data mining operation. The same applies to the Intelligence Life-Cycle where the outputs are known and form an intrinsic part of the intelligence of law enforcement agencies.

2.9 Intelligence Products

Both the CRISP-DM and Intelligence Life-Cycle share much in common. Finally the use of intelligence products is the analysis of various forms of data in such a way that it can be used to find new unknowns or identify potential new targets. The simplest example of this is that a list of known names as input which matched against the various data stores produces a result of known activity. This is referred to as *washing* where potential knowledge is formed about an individual's activity based on data within the intelligence data base. If relationships are known within the data then these relationships allow for basic Social Network Analysis to be performed: for example density measures, *k core* analysis which forms part of their knowledge discovery. This new knowledge can be published and presented in the form of an intelligence product whereby the analyst can utilise this to further their investigation.

2.9.1 Dependency Analysis

Dependency analysis consists of finding a model that describes significant dependencies (or associations) between data items or events. Dependencies can be used to predict the value of a data item or add information on other data items. Although dependencies can be used for predictive modelling, they are mostly used for understanding. Dependencies can be strict or probabilistic in that relationships were ascertained from unreliable sources. However, dependency analysis is important to any investigation and can shed light on how certain actions were undertaken by known targets.

2.10 Summary

This chapter has described the Intelligence Life-Cycle and the demands of data in particular reference to the collection and collation phase. The emergence of ‘Big Data’ has a significant impact on the way data is managed and processed within the Intelligence Life-Cycle in particular reference to the collection, collation and process phases. ‘Big Data’ technology has removed many of the barriers that restricted what could be gathered and how this data could be analyzed. Furthermore, a new approach to deal with data variability needs to be defined. The next chapter will examine the ‘Big Data’ perspective and the impact of ‘Big Data’ products and technologies on the Intelligence Life-Cycle.

Chapter 3

The ‘Big Data’ Perspective

The ‘Big Data’ perspective is the first important task to address in order to make the ‘Big Data’ analytics efficient and cost effective. The early detection of the ‘Big Data’ characteristics can provide a cost effective strategy to many organizations to avoid unnecessary deployment of ‘Big Data’ technologies. The data analytics on some data may not require ‘Big Data’ techniques and technologies; the current and well established techniques and technologies may be sufficient to handle the data storage and data processing. Hence what is required is an early analysis and understanding of the data characteristics for classification.

> “Big data is high-volume, high-velocity and high-variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making (Gartner, 2014).”
>
> “Big data is data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or does not fit the strictures of your database architectures. To gain value from this data, you must choose an alternative way to process it (Dumbill, 2012).”
>
> “The problem with big-data-as-technology is that:

1. it is vague enough that every vendor in the industry jumped in to claim it for themselves

2. and everybody 'knew' that they were supposed to elevate the debate and talk about something more *business-y* and useful (Elliott, 2013)."

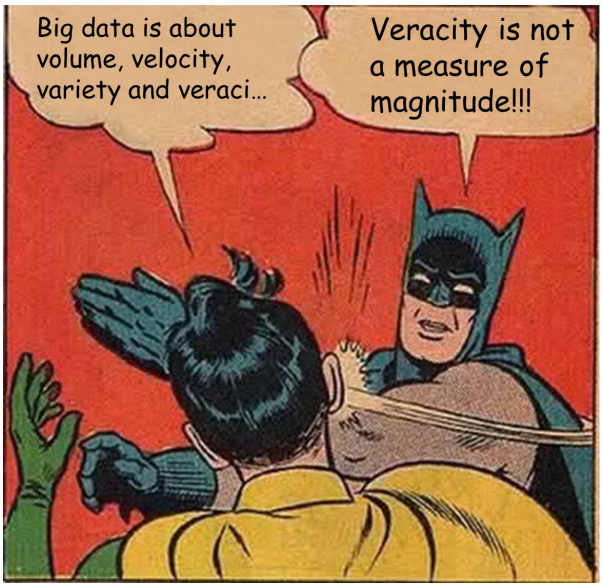


Figure 3.1: Veracity is not a measure of magnitude

'Big Data' has often been described as a classification of technologies designed to specifically address the problems of data Variety, Velocity and Volume or V^3 (Gartner, 2014):

Variety 'Big Data' extends beyond structured data, including unstructured data of all varieties: text, audio, video, click streams, log files and more.

Velocity Often time-sensitive, big data must be used as it is streaming in to the enterprise in order to maximize its value to the business.

Volume 'Big Data' comes in one size: large. Enterprises are awash with data, easily amassing terabytes and even petabytes of information.

Often **veracity** and **value** are used to describe Big Data where **veracity** was introduced by IBM International and **value** by Oracle Corporation. However as Figure 3.1 shows neither are measures of magnitude.

The industry has seen this as an opportunity to create a new class of products specifically to address these issues. The plethora of products that class themselves as 'Big Data' is forever continuing and old database products are reclassifying themselves as 'Big Data' and an example of this is IBM's DB2.

Compared to defining a metric to measure the Big Data characteristics in V^3 space, it is much easier to develop a metric in C^3space using mathematical and statistical tools. In

C^3space the cardinality defines the number of records in the dynamically growing dataset at a particular instance (see Figure 3.2). The continuity defines two characteristics and they are (Suthaharan, 2012):

- representation of data by continuous functions, and
- continuously growth of data size with respect to time.

The complexity defines three characteristics and they are:

- large varieties of data types;
- high dimensional dataset; and
- the speed of data processing is very high.

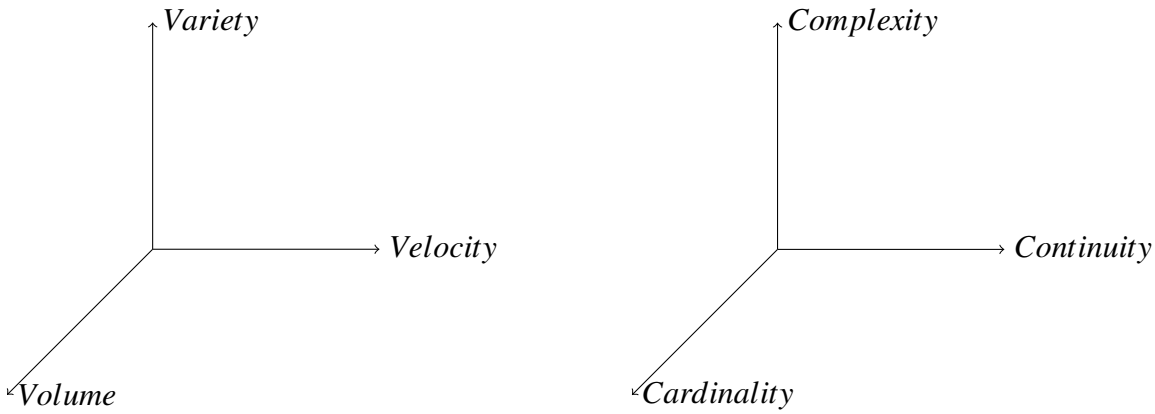


Figure 3.2: Three 'V's compared to the three 'C's

3.1 'Big Data' Characteristics

'Big Data' implementations share the following characteristics:

In addition, the importance of the **timeliness** of the data cannot be underestimated. A 'Big Data' implementation has the following characteristics:

- Fast insertion time: records must be added quickly.

- Fast random retrieval time.
- Fast sequential read time where it is necessary to read the data *in-situ.*
- Data replication: this is where the data can be replicated or copied across one or more computers.
- Eventual consistency: the action of a record update, deletion or insertion is not necessarily immediate across all copies of the data.
- Data description process is iterative. More compute power means that it is possible to iterate models until they meet the specification. For example, building a model trying to find the predictors for certain customer behaviors. The capture process may begin with the extraction and analysis of a sample data set. This may support the initial data descriptive hypothesis or reject this hypothesis.
- Only programmatic interface are provided by the 'Big Data' vendors. In addition, there are limited user-friendly tools to explore the data sets. There is currently no standard query language with 'Big Data' as with relational databases. An exception to this rule is RDF triple stores.

'Big Data' implementations ***are not required*** to support the following:

- Row/Set Deletion: this is where records are deleted, some implementations for example *Cassandra* place an indicator on the record or records to indicate that the record has been deleted. This indicator is referred to as a *tombstone.*
- Atomic and isolated updates. Not all 'Big Data' implementations lock records or ensure data integrity. Certain implementations provide locks or alternate record version management. Generally implemented by a lock placed on record or records to ensure the integrity of the transaction. This is particularity important if the transaction is financial in nature. However, some 'Big Data' implementations do support this concept.
- The necessity of the *null* value to indicate a missing value.

- Most implementations provide some form of data consistency. However, there are a number of distinct consistency models. For example Cassandra is a 'Big Data' implementation and provides eventual consistency whereby replicated data is spread across multiple nodes and is *eventually* updated. HBase provides an immediate consistency model and updates are applied to all the replicated copies.
- Backups are not always implemented or practical due to the size of the actual 'Big Data' repositories and are split across any number of disks or nodes. This is considered as an alternative strategy to traditional backups.
- 'Big Data' implementations are data driven, unlike in relational databases, where the structure drives the data not where the data drives the structure. There is no longer the requirement for rigid data description to represent the data.
- It can use a lot of attributes. In the past, you might have been dealing with hundreds of attributes or characteristics of that data source. Now you might be dealing with hundreds of gigabytes of data that consists of thousands of attributes and millions of observations. Data analytics and processing is now happening on a larger scale.

With the rapid decline in storage prices and the increased availability of large cheap commodity hardware, the need to delete data due to restricted storage demands has come to an end. Many storage hardware devices are designed to expand as the needs of their customers grow.

3.2 'Big Data' Classifications

'Big Data' describes a set of technologies for the storage, manipulation and retrieval of large data sets. Usually these data sets exceed some limit that *current* technologies are unable to support. The large data volumes also require fast access times, however there are two distinct requirements in this area, the first being able to retrieve all or some of the data in a time that is pertinent to the decision time.

Data within a 'Big Data' store does not have to be tabular. There are four storage models that have dominated 'Big Data' implementations:

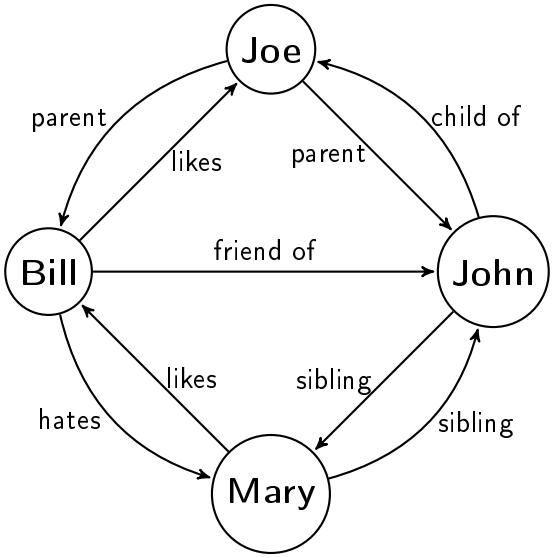


Figure 3.3: Linked structure representation

1. Tabular where data is stored as tables similar to the relational data bases as Postgreql, DB2, Oracle, SQLServer and MySQL.

2. Columnar where data is stored as columns rather than the rows. Example of this type of implementation include Apache Cassandra and Apache HBase.

3. Name/Value pairs where there are no rows but a collection of pairs. The name is always the index or identifies the associated value. Example of a name/value database is Berkeley DB.

4. Graphical where the data represented by a graph or a series of interconnected nodes and links between those nodes (see Figure 3.3). These fall into the following categories:

   - RDF/Triple (Semantic) Stores, and
   - Native Graphical Data Bases which include implementations as **Neo4j**.

Whatever the storage model their implementation must be able to deal with large amounts of data. 'Big Data' technologies are described as NoSQL (*not only* SQL) that

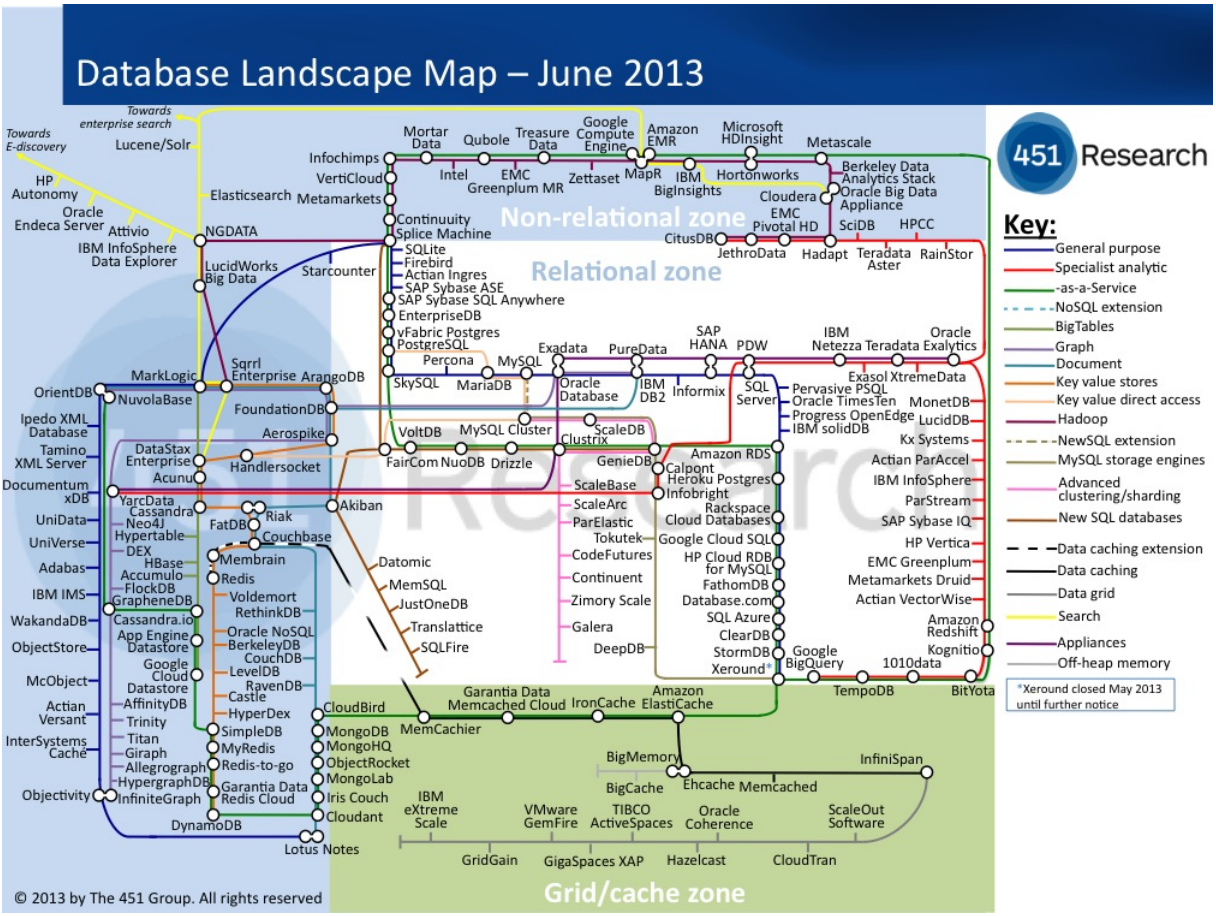


Figure 3.4: The database landscape

the SQL language support is optional. Generally, 'Big Data' products are purpose built to handle the three 'V' and sometimes at the expense of simple to use interrogation languages.

The number of potential 'Big Data' implementations is growing at an unprecedented rate and this cannot be better demonstrated as in Figure 3.4 which shows the plethora of products that either process and or store data. This does present problems for any potential analyst to attempt to determine what is the most appropriate product that may suite their needs.

3.2.1 NoSQL Classification

NoSQL is a whole new way of thinking about a database. NoSQL is not a relational database. The reality is that a relational database model may not be the best solution for all situations. The easiest way to think of NoSQL is that of a database which does not adhere to the traditional relational database management system (RDMS) structure (Menegaz, 2012). The delineation between NoSQL and SQL systems is that NoSQL systems are specifically designed to contain large data sets at the sacrifice of SQL like interrogation language. In many cases only an application programming interface (API) is provided to both interrogate and store data. Until a standard query language is announced by an international committee that will be the case for some time to come.

3.2.1.1 NoSQL Characteristics

Cassandra and HBase have borrowed much from the original **Bigtable** definition. In fact, whereas Cassandra descends from both Bigtable and Amazon's Dynamo, HBase describes itself as an 'open source **Bigtable** implementation.' As such, the two share many characteristics which include:

- NoSQL databases, a term which generally means that there is no SQL language support.
- Designed to manage extremely large data sets.
- Data is distributed amongst multiple nodes. This means all the data can be accessed from any one of these nodes.
- Near linear scalability. Therefore, doubling the capacity of the system requires doubling the capability of the throughput.
- Replication is used to safeguard data loss.

'Big Data' sets can be so large that it is not possible to fit them within a single data centre, in this case, clustering is used to disseminate the data across multiple nodes. These data nodes do not necessarily belong to the data owner. Cloud computing has made it possible

to out-source the management of the data to another organization. 'Big Data' relies on big infrastructure and with the emergence of cloud computing this has made it possible for small organizations to extend their infrastructure well beyond their capabilities.

3.2.2 Columnar NoSQL Databases

Columnar implementations are a class of 'Big Data' implementations that store data in columns rather than rows. This style of 'Big Data' repository was first introduced by Google with their *Big Table* implementation.

'Big Data' is described by Google as:

> "Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from back-end bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this paper we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable (Chang, 2006)."

The difference between relational and columnar databases is that data is stored in columns and not rows. However, columnar databases utilise a single key to relate all the columns together (see Figure 3.5). Unlike relational implementations compound keys are not permitted. Some implementations permit secondary indexes but only on predefined columns which is commonly referred to as column families. Columnar data bases are restricted to column families which for most implementations support *secondary* indexes where the column's name determines the index field and value. A limitation with this approach is that range queries are not possible and the rows cannot be returned in a specific order. Nonetheless, if range queries are not required *secondary* indexes provides an alternate access to the data.

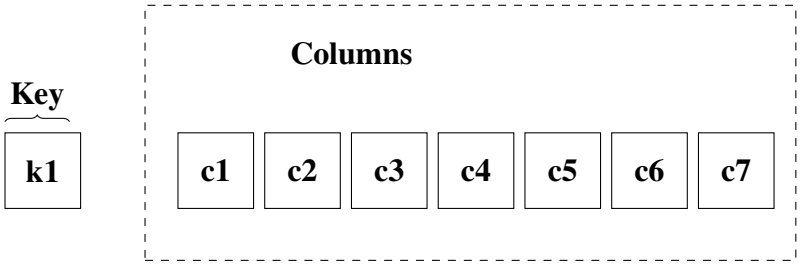


Figure 3.5: Columnar data storage representation

This provides some significant advantages for analytical processes and ad hoc queries and provides the additional benefit in that less disk space is used as with traditional RDBMS. Another advantage is that when selecting records in a query process, it is only necessary to read the columns that are included as qualifiers and not the entire row and stated by Bhatia and Patil "When columns are stored in a sorted sequence, the response speed improvement can be even greater (Bhatia and Patil, 2011)".

The primary disadvantage of columnar databases there is no established formal query language as with relational databases which is *structured query language* or SQL. Furthermore, the approach taken by vendors to secondary indexing strategies is inconsistent across Columnar implementations.

3.2.2.1 Cassandra

Cassandra was initially created by **Facebook** to provide search functionality for a user's mailbox. The source code was open sourced and released to the Apache Software Foundation. Its design was inspired by both Google's Bigtable and Amazon's Dynamo. It is considered to be a column data store, similar to a Google Bigtable or Apache HBase. Cassandra is part of the open source communicate and is still actively developed. In addition, Cassandra exposes a number of application programming (API) interfaces. The interface known as ***thrift*** provides a basic and yet complex interface to all the functions available to Cassandra.

Cassandra consists of the following key components (Datastax, 2014):

- Cluster: a container for one or more *keyspaces.* A Cassandra instance may contain only one cluster however the cluster may be spread across one or more computers.
- Keyspaces: a container for *column families.* A keyspace is analogous to a schema in a relational database.
- Column families: analogous to a table in a relational database which may contain an unlimited number of rows and up to four billion columns. Each row within a column family must have a single unique key to identify the row.
- Columns: A column is a name-value pair. The name may contain 64 Kilobytes of data and there is no real limitation to the size of the value.

3.2.2.2 Apache HBase

HBase is a column-oriented database management system that runs on top of HDFS. and well suited for sparse data sets. Unlike relational database systems, HBase does not support a structured query language like SQL; in fact, HBase is not a relational data store at all. HBase applications are written in Java much like a typical Map-Reduce application. Unlike Cassandra, HBase does support writing applications in Avro, REST, and Thrift. Each table contains rows and columns, much like a traditional database. Each table must have an element defined as a Primary Key, and all access attempts to HBase tables must use this Primary Key. HBase allows for many attributes to be grouped together into what are known as column families, such that the elements of a column family are all stored together. This is different from a row-oriented relational database, where all the columns of a given row are stored together. With HBase you must predefine the table schema and specify the column families. However, it is very flexible in that new columns can be added to families at any time, making the schema flexible and therefore able to adapt to changing storage requirements. Just as HDFS has a *Name-Node* and slave nodes, and Map Reduce has JobTracker (Zookeeper is used to perform this function) and Task-Tracker slaves, HBase is built on similar concepts. HBase has a master node that manages the cluster and region servers, stores portions of the tables and performs the work on the data. In the same way

that HDFS has some enterprise concerns due to the availability of the Name-Node, HBase is sensitive to the loss of its master node.

3.2.2.3 Apache Accumulo

Apache Accumulo is 'the new kid on the block' and similar to HBase in many ways except for one notable feature in that Accumulo can secure data at the individual cell level. This was a feature requested by the National Security Agency as security was overlooked by many columnar implementations. Apache Accumulo is based on Google's Bigtable design and is built on top of Apache Hadoop, Zookeeper, and Thrift.

3.2.2.4 Cloud Columnar Offerings

There are number of columnar databases that are worth mentioning but they differ because these implementations are columnar storage offered by cloud providers. This includes Amazon's **Redshift** and Google's **Bigtable.** Both are columnar storage options as part of their Cloud Platform. Amazon describes this platform:

> "Amazon Redshift is a fast, fully managed, petabyte-scale data warehouse solution that makes it simple and cost-effective to efficiently analyze all your data using your existing business intelligence tools. You can start small for just $0.25 per hour with no commitments or upfront costs and scale to a petabyte or more for $1,000 per terabyte per year, less than a tenth of most other data warehousing solutions (Amazon, 2014)."

3.3 The Semantic Web

The semantic database or 'triple store' is a technology that is gaining popularity and can be used to store semi-structured data in a graphical manner and record relationships between specific data items. A triple store is a purpose-built database for the storage and retrieval of triples, a triple being a data entity composed of subject-predicate-object.

Much like a relational database, one stores information in a triple store and retrieves it via a query language. Unlike a relational database, a triple store is optimized for the storage

$$Subject \xrightarrow{\text{Predicate}} Object$$

Figure 3.6: The triplet structure

and retrieval of triples. In addition to queries, triples can usually be imported/exported using Resource Description Framework (RDF) data descriptions and other formats. Some triple stores can store billions of triples.

RDF is a standard model for data interchange on the Web. RDF has features that facilitate data merging even if the underlying schema differs, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed.

RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a "triple"). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications. This linking structure forms a directed, labelled graph, where the edges represent the named link between two resources, represented by the graph nodes. This graph view is the easiest possible mental model for RDF and is often used in easy-to-understand visual explanations.

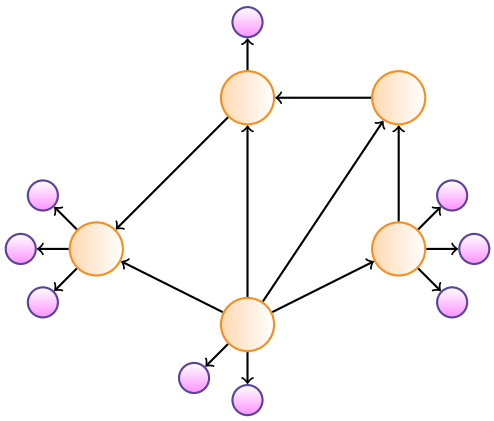


Figure 3.7: A directed graph structure

The triplet is a data structure composed of a subject, predicate and object where the predicate binds the subject to the object (see Figure 3.6). The subject and predicate must conform to the Uniform Resource Specification (URI) as described by Tim Berner-Lee in 1991 (Berners-Lee, 1991). The URI which is also referred to as a **resource** provides the mechanism to link or associate one or more triplets. The object component of the triplet serves dual purposes. If the object contains a **resource** this identifies a directed link between this triplet and other triplets within the graph (see Figure 3.7). However, if the

object contains a literal value then this is the data associated with the triplet. If the object does contain a literal value then additional information is stored to identify the literal's primitive type.

A special resource type, the blank resource is similar to to the URI but is merely used to group triplets together. Blank nodes can only be used as a subject or object resource identifier.

An extension to the triplet specification is the *graph* identifier. Like the subject and predicate, the graph identifier is represented by a resource. The graph identifier sometimes called the triplet's *context* provides a mechanism to group large numbers of triplets together. The context can be used as an extra *dimension* to the triplets contained within the graph. Uses of the context are as follows:

- Identify the source of the data. This can also be used to determine the triplet's provenance.
- A temporal dimension for the triplets.
- The owner of the triplets.
- The security (authorization) classification.

The W3C specification Resource Definition Framework (RDF) is a specification that describes structure and formal representation of the triplet structures (W3C Working W3C, 2014). The RDF necessary to describe the structure required to capture the ingredients that are contained within Chutney is shown in Figure 3.8.

W3C defines a language specification SPARQL (Simple Protocol and RDF Query Language) and is an RDF triplet querying language (W3C, 2008a). SPARQL provides a specification to retrieve triplets based on one or more graph patterns. An example of a SPARQL query is shown in Figure 3.9. The SPARQL language specification also extends to the format in which the triplets are returned. The formats include: XML, N3 and JSON. The basic SPARQL retrieval patterns that are necessary to implement the **select** statement are shown in Figure 9.6.

```
@prefix ex: <http://example.org/>.
@prefix rdf: <http://www.w3.org/...#>
ex:Chutney ex:hasIngredient ex:item1.
ex:Chutney ex:hasIngredient ex:i.
ex:item1 rdf:value ex:greenMango.
ex:item1 rdf:amount "50gm".
ex:item2 rdf:value ex:relish
ex:item2 rdf:amount "20gm"
```

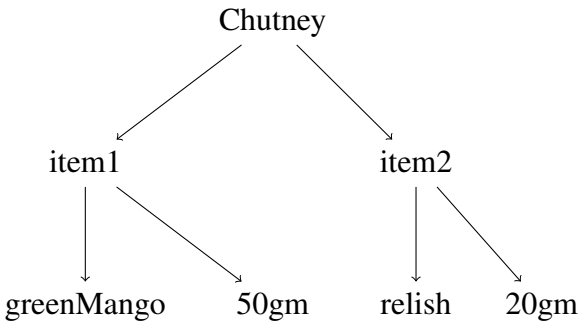


Figure 3.8: RDF to describe the contents of ‘chutney’ and visual representation

```
select ?subj ?pred ?obj
  where {
    {
      ?subj ?pred ?obj.
      ?store <fusion://map> ?subj.
    } union {
      ?subj ?pred ?obj.
      ?store <fusion://schema> ?subj.
    } union {
      ?subj ?pred ?obj.
      ?store <fusion://model> ?subj.
    }
}
```

Figure 3.9: SPARQL language example

3.3.1 RDF Data Structures

The Resource Definition Framework has identified and defined four major data structures which are the following:

- **Bag** - is a collection of triplets with no specific order.
- **Sequence** - is a collection of triplets where the order matters.
- **Alternate** - is a collection of triplets, however only one triplet can ever be selected.
- **List** - is a generic specification for a *linked* list structure (see Figure 5.21 as an example of a RDF linked list).

In addition to the standard RDF data structure, the triple store can store triplets and any linkage pattern (Hitzler et al., 2010) (see Figure 3.7 as an example of an undirected graph).

The RDF list structure is designed to accommodate large ordered lists . The list has a node that represents the list's head and is terminated by the **'nil**' RDF resource which represents the list's tail (W3C, 2014). Each RDF node within the list contains a link both to the data and to the next item within the list and these links are are represented by the RDF 'first' and RDF 'next' predicates respectively (see Figure 5.21). There is no restriction to the number of nodes contained within a linked list.

3.3.2 The SPARQL Language

SPARQL is a language that allow the interrogation and manipulation of triple stores. Even though SPARQL does resemble SQL, there is little in common except a few key words between the two languages. SPARQL has four retrieval constructs unlike SQL which only has one, the select statement. However, unlike SQL which deals with tabular data structures, SPARQL operates on graph data and each data item contains the name (subject), the value (object) and the relationship between the name and object (predicate). Another key difference between the relation database is that there is no need for a 'null' value in relation.

Data:

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
_:a  foaf:name   "Johnny Lee Outlaw" .
_:a  foaf:mbox   <mailto:jlow@example.com> .
_:b  foaf:name   "Peter Goodguy" .
_:b  foaf:mbox   <mailto:peter@example.org> .
_:c  foaf:mbox   <mailto:carol@example.org> .
```

Query:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { ?x foaf:name ?name .
         ?x foaf:mbox ?mbox }
```

Result:

name	mbox
"Johnny Lee Outlaw"	<mailto:jlow@example.com>
"Peter Goodguy"	<mailto:peter@example.org>

Figure 3.10: SPARQL example

3.3.3 The Triple Store

A triple store is framework used for storing and querying RDF data. Triple Stores provides a mechanism for persistent storage and access of RDF graphs. Recently there has been a major development initiative in query processing, access protocols and triple-store technologies. The number of triple stores being actively developed has increased from Jena and Sesame in the early 2000s to Garlik JXT, YARS2, BigOWLIM, Jena TDB, Jena SDB, Virtuoso, AllegroGraph, BigData, Mulgara, Sesame, Kowari, 3Store and 4Store. Others like BigOWLIM and AllegroGraph are commercially available and the rest are all free and open source. Only the freely available open source triple stores were considered and hence the choice of Jena SDB, Sesame, Mulgara and Virtuoso.

Triple stores can be divided into 3 broad categories which are: in-memory, native, non-memory non-native - based on architecture of their implementation. In-memory triple stores store the RDF graph in main memory and usually outperform their counterparts at the expense of not being able to store large data volumes. However, this classification is

useful to cache results from SPARQL queries or performing certain operations like caching data from remote sites or for performing inference analysis on small graph structures. Most of the in-memory stores have efficient reasoners available and can help solve the problem of performing inferencing in persistent RDF stores, which otherwise can be very difficult to perform. A second, now dominant category of triple stores is the native triple stores which provide persistent storage with their own implementation of the databases, for example, Bigdata, Virtuoso, Mulgara, AllegroGraph and Garlik JXT. The third category of triple stores, the non native non memory triples stores, are set up to run on third party databases for example Jena SDB which can be coupled with almost all relational databases like MySQL, PostgreSQL and Oracle. Oracle has recently announced an implementation based on BerkleyDB. Recently native triple stores, due to their superior load times and ability to be optimized for RDF, have gained popularity with intelligence organisations.

The approach taken is to acknowledge that generally most triplets have low predicate cardinality compared to subjects and objects. No current triple store implementation takes account of this. In addition, RDF structures such as list, bags, sequences and alternates are not treated differently than other predicate types. These W3C predefined data structures demonstrates the suitability of the Columnar Storage implementations suitable to contain triples and implement the SPARQL specification.

Ontological structures (Ontologies) are also key to many semantic web applications. Ontologies are formal logical descriptions, or models, of some aspect of the real-world that applications have to deal with. Ontologies can also be shared with other developers and researchers, making it a good basis for building linked-data applications. There are two ontology languages for RDF: RDFS, which is rather weak and does not represent complicated ontological structures, and OWL, which is much more expressive.

Not all triplet patterns need to be supported by the implementation. it is not uncommon that the object component of the triplet is accessible via an external index. For example, if the object contains a person's name SPARQL will only support an exact match and not consider name variations and misspellings. It is more likely that a specialized index is constructed specifically tailored to search a person's name. Sesame Lucene implementation allows an object value to be searched for phonetically. Therefore, the triple store

Storage Type	Implementations	Comment
Document Orientated	MongoDB	
Columnar	Cassandra HBase Accumulo	No RDF support
Key Value	Berkley DB Dynamo Redis Riak	Becoming more popular as an RDF platform
Graph Database	Neo4j OpenLink Virtuoso AllegroGraph	Some RDF Support via **Tinkerpop**
RDF Native	Bigdata	A Sesame SAIL exists
	BigOWLIM	
	Mulgara	A Sesame SAIL exists
	Virtuoso	Supports both Sesame and Jena

Table 3.1: Comparison of triple store storage implementations

Requirement	Virtuoso	Oracle	OWLIM	Allegro	Bigdata	Mulgara	4Store	Sesame	Stardog	B*	DB2	Jena
Open source	✓	×	×	×	✓	✓	✓	✓	×	×	×	✓
> 10 billion	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	×
Clustering	✓	✓	✓	✓	✓	✓	×	×	✓	✓	✓	×
SPARQL 1.1	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SPARQL Update	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	✓
Property Path	×	×	×	×	×	×	×	✓	×	×	×	✓

Table 3.2: Triple store implementations (Garshol, 2012)

serves two purposes in the representation of graph structures and capability to search object stores. Furthermore, the triple store can contain tabular structures similar to that of relational database implementations. Unlike relational database implementations, complex graph structures can be easily represented and stored within an RDF triple store. Also, with the addition of the Graph predicate this provides a mechanism to logically separate large inter-related linked structures. Implementations like R2RML and SDshare are not true triple stores in that they do not support the SPARQL language.

3.3.3.1 Apache Jena

The Apache Jena framework was originally developed by Hewlett Packard in their Bristol laboratories in 2010 and subsequently given to the Apache Foundation. Since leaving incubation status in April 2012, Apache Jena is now a top-level product. The framework contains a SPARQL parser (Jena ARQ) which transforms the SPARQL query into one or more triple patterns (see Table 9.6).

Jena provides a collection of tools and Java libraries to help you to develop semantic web and linked-data apps, tools and services. OWL and RDFS are both supported in Jena through the Ontology API, which provides convenience methods that know about the richer representation forms available to applications through OWL and RDFS. Jena has been extended to include **Lucene** search capabilities, geospatial support and a number of physical storage implementations are available via their TDB interface.

The Jena Framework *onion-ring* includes (see Figure 3.11):

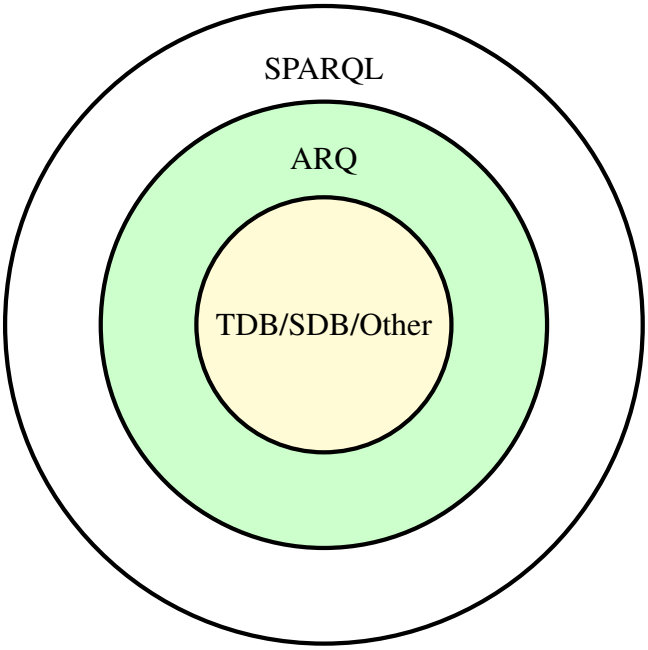


Figure 3.11: The Apache Jena platform

- an API for reading, processing and writing RDF data in XML, N-triples and Turtle formats.
- an ontology API for handling OWL and RDFS ontologies.
- a rule-based inference engine for reasoning with RDF and OWL data sources.
- stores to allow large numbers of RDF triples to be efficiently stored on disk. There are two 'out-of-the-box' engines which are: TDB is a file based triple database, and SDB uses an SQL database for the storage and query of RDF data.
- a optimized query engine compliant with the latest SPARQL specification. This query engine is known as ARQ and supports both TDB and SDB.
- services to allow RDF data to be published to other applications using a variety of protocols, including SPARQL and REST.

The Jena development team have developed a separate engine (ARQ) which processes SPARQL statements and translates the request to a database retrieval. This provides independence between the triplet storage and the storage of the triplets themselves. Jena does however provide a number of reference implementations: TDB is a native file based triple storage provider and SDB is an interface specification targeted to support relational databases.

3.3.3.2 OpenRDF - Sesame

Sesame is an open source Java framework for querying and storing RDF data; it was originally developed by the Dutch Company Aduna as a research prototype for the European Union research project On-To-Knowledge. Aduna describes Sesame as:

> "Sesame is an open source Java framework for storage and querying of RDF data. The framework is fully extensible and configurable with respect to storage mechanisms, inferences, RDF file formats, query result formats and query languages. Sesame offers a JBDC like user API, streamlined system APIs and a *RESTful* HTTP interface supporting the SPARQL Protocol for RDF (Aduna, 2013)."

Sesame is now an open-source project and has a large active development community. There are a large number of reference storage implementations which include MYSQL, PostgresDB, Oracle, 'Bigdata' and there own file base storage. Aduna's Sesame is a popular triple storage implementation providing the infrastructure and SPARQL support.

Sesame storage has their own SAIL implementation where the SAIL (**S**torage **A**nd **I**nference **L**ayer) is a low level System API (SPI) for an RDF store and *inferencer* (see Figure 3.12). The SAIL defines the abstraction layer between the SPARQL language processor and the triple storage.

3.3.3.3 Mulgara

Mulgara is a native RDF triple store written in Java and provides a Connection API that can be used to connect directly to the Mulgara store. Mulgara's **'load**' script can be used to *bulk*

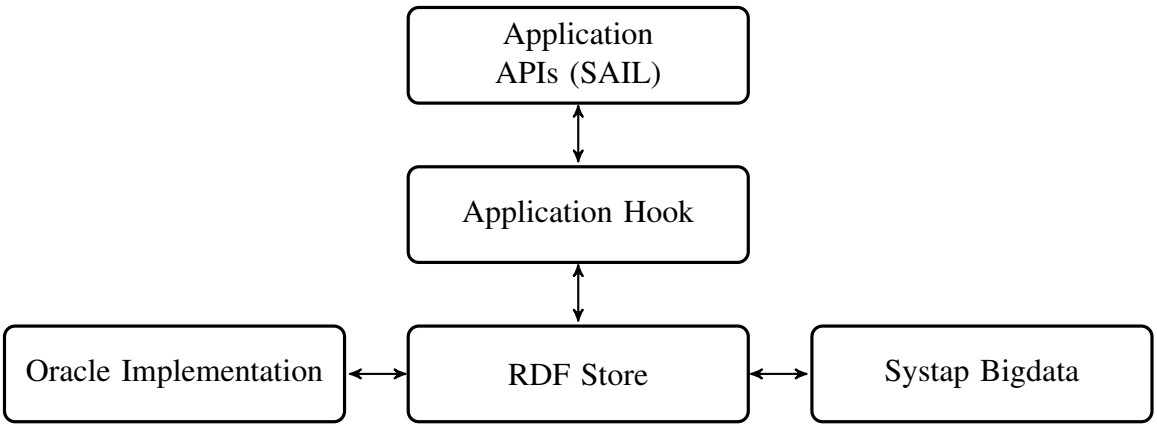


Figure 3.12: Sesame API SAIL structure

load RDF data into the triple store. The TQL shell which is a command line interface can be used to perform SPARQL queries and add, delete or modify triplets contained within the store. Mulgara also provides basic *graph* support.

3.3.3.4 Virtuoso

Virtuoso is a native triple store available in both open source and commercial licenses. It provides a command line *bulk* loader, a connection API, support for SPARQL and web server to perform SPARQL queries and can also upload data over this interface. A number of evaluations have tested Virtuoso and found it to be scalable to the region of one billion triplets. In addition to this, Virtuoso provides bridges for both the with Jena and Sesame platforms.

3.4 Summary

This chapter has explored the 'Big Data' world and how this has come to ultimately change how intelligence organizations such as ACC must deal with data management issues. There was a detailed examination of 'Big Data' products which includes both the columnar and graph classifications. The impact of 'Big Data' has a profound effect on the Intelligence Life-Cycle and the advent of the triple store which can both capture knowledge and process large data sets.

The next two chapters will focus on data variability and processing. There are two main identifiable approaches the 'Schema-First' and 'Schema-Last'. The 'Schema-Last' approach will be shown to be superior to the 'Schema-First' and how this approach significantly improved the collation and process of data within the Intelligence Life-cycle.

Chapter 4

'Schema-First': The Current Approach

Schema is from the Greek word meaning 'form' or 'figure' and is a formal representation of data model which has integrity constraints controlling permissible data values. This chapter will examine the existing data modelling techniques and how these techniques relate to data presentation and storage. Data is sometimes corrected to comply to a Schema. This approach is referred to as 'Schema-First' where changes to the Schema mean changes in the data and changes in the data will require changes made to the Schema. The usual start in designing an information system is to create a conceptual schema that models the data structure. After the conceptual schema is finalized, this schema is then implemented on a target platform such as a relational system, a hierarchical system, or an object-oriented system. To this end, the conceptual schema is transformed (mapped) to a schema on the chosen target platform. The entire (data) modelling process can be seen as a transformation process of data schema, where a data schema can be an Entity Relationship schema, an Object-Role Modeling schema, a relational schema, or any other internal representative schema.

Schema is used to describe relational tabular, hierarchical or graph structures. Usually, schema is used to identify how the data is to be stored or transported. For sources without schema, such as files, there are few restrictions on what data can be entered and stored, giving rise to a high probability of errors and inconsistencies. Database systems, on the other

hand, enforce restrictions of a specific data model (for example: the relational approach requires simple attribute values, referential integrity, *et cetera*) as well as application-specific integrity constraints.

A schema can be seen as an application. After the conceptual schema has been finalized, one can sometimes perform small (equivalence preserving) transformations on a schema which result in a schema that allows for a more efficient implementation. These transformations typically utilize the rich semantics and clarity of a conceptual schema. Performing such transformations after all schema definitions have already been mapped to a target platform usually becomes too complicated, since these schema definitions use less concise modeling concepts. This makes it both harder to define the transformations, and harder for the information system designers to track the transformations. Following the optimization transformations, the schema is consequently mapped to a target platform. For the different conceptual modeling techniques there are different mapping algorithms following varying styles and strategies. Once a conceptual schema has been transformed to some sort of representation for a target platform, this schema can sometimes be optimized even further. There are five key distinguishable classes of schema transformations:

1. Conceptual schema refinements.
2. Conceptual schema quality improvements.
3. (Conceptual) schema optimizations.
4. Conceptual to internal (logical) schema mapping
5. Internal schema optimizations.

For a more elaborate discussion on the classes of schema transformations, presently, the reverse process of the transformations in four and five also receives much attention in the database and information systems research community. This **reversed** process is referred to as *reverse engineering*. For this fairly new area also a wide range of strategies and algorithms exists. These algorithms all operate on the basis of a set of possible schema transformations and heuristics to best apply to them. The above discussed five classes either operate on a conceptual data schema or an internal schema of a given implementation

platform. The modelling process until the start of the internal schema mapping can be regarded as a journey through a universe of data schemas. This universe of data schemas must be rich enough so that the data schemas can include both details relevant for a conceptual presentation as well as the internal representations.

4.1 Schema Application

The 'Schema-First' may mean a loss of data quality at any one of these stages and reduce the applicability. These include (Chapman, 2005):

- Data capture and recording at the time of gathering.
- Data manipulation prior to digitization (label preparation), identification of the collection and its recording.
- Digitization of the data.
- Documentation of the data (capturing and recording the meta-data).
- Data storage and archiving.
- Data presentation and dissemination (paper and electronic publications, web-enabled databases, *et cetera*).
- Data use (analysis and manipulation). All these have an input into the final quality or 'fitness for use' of the data and all apply to all aspects of the data – the taxonomic or nomenclature portion of the data – the 'what', the spatial portion – the 'where' and other data such as the 'who' and the temporal 'when'.

The traditional approach is to describe data attributes prior to the data insertion (see Figures 4.1 and 4.2). A schema is a term that applies a *specification* to the data before any data can be inserted. For example, relational databases require a table structure to be defined which contains the column definitions that each row must conform to that specification. There are restrictions placed on what can and cannot be changed to the table schema. However, what is difficult to change is the structure of the data and the data objects are relationships.

Typically, the 'Schema-First approach utilizes a *normalized* data model. This set of data schemas is defined by the data schema language. This language must be rich enough to allow us to model both the conceptual aspects of a data schema as well as internal representation aspects into one single model.

```
create table sample (
    first_name varchar(256) not null primary key,
    surname varchar(256) not null primary key,
    date_of_birth date not null primary key
);
```

Figure 4.1: Example relational table definition

The 'Schema-First' model demands data analysis up front. The columns, rows and data types must be defined. In addition, foreign key relationships between the various tables must be known. In general the foreign keys must be supported by an index otherwise the search and update performance may be adversely affected.

4.2 Ontology First

Ontology is technology that can be described as knowledge about knowledge. The term is described as:

> "The word 'ontology' seems to generate a lot of controversy in discussions about AI. It has a long history in philosophy, in which it refers to the subject of existence. It is also often confused with epistemology, which is about knowledge and knowing.
>
> In the context of knowledge sharing, I use the term ontology to mean a specification of a conceptualization. That is, an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. This definition is consistent with the usage of ontology as set-of-concept-definitions, but more general. And it is certainly a different sense of the word than its use in philosophy (Gruber, 1993)."

```
<?xml version='1.0' encoding='UTF-8'?>
    <xs:schema xmlns:tns="http://search.fusion.acc.gov.au/services"
               xmlns:xs="http://www.w3.org/2001/XMLSchema"
               elementFormDefault="qualified"
               version="1.0" targetNamespace="http://search.fusion.acc.gov.au/services">
    <xs:element name="template">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="properties" type="tns:property"
                            nillable="true" minOccurs="0" maxOccurs="unbounded" />
                <xs:element name="declaration" type="tns:field" nillable="true"
                            minOccurs="0" maxOccurs="unbounded" />
                <xs:element name="specification" type="tns:field"
                            nillable="true" minOccurs="0" maxOccurs="unbounded" />
                <xs:element name="retrievalPackage" type="tns:retrievalPackage"
                            nillable="true" minOccurs="0" maxOccurs="unbounded" />
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required" />
        </xs:complexType>
    </xs:element>
    <xs:complexType name="retrievalPackage">
        <xs:sequence>
            <xs:element name="properties" type="tns:property" nillable="true"
                        minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
        <xs:attribute name="name" type="xs:string" use="required" />
    </xs:complexType>
    <xs:complexType name="property">
        <xs:sequence>
            <xs:element name="value" type="xs:string" nillable="true"
                        minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
        <xs:attribute name="name" type="xs:string" use="required" />
    </xs:complexType>
    <xs:complexType name="field">
        <xs:sequence>
            <xs:element name="value" type="xs:string" minOccurs="0" />
            <xs:element name="type" type="xs:string" minOccurs="0" />
            <xs:element name="properties" type="tns:property"
                        nillable="true" minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
        <xs:attribute name="name" type="xs:string" use="required" />
    </xs:complexType>
</xs:schema>
```

Figure 4.2: Example XSD definition

Most commercial products that support ontology structures apply ontology on data ingestion. In addition, modification to the ontology structure may require a complete data unload or reload. Products like Palantir and IBM's Analyst Notebook supported limited ontological structural changes in that additional structures may be added but once added are difficult or impossible to change.

4.3 Data Cleansing and the 'Schema-First' Approach

Data cleaning, also called data cleansing or scrubbing, deals with detecting and removing errors and inconsistencies from data in order to improve the quality of data. This is an integral part of the 'Schema-First' approach in that for any data to be ingested into a schema-managed repository must strictly comply to the rules laid out by the schema definition. Therefore, data that does not strictly comply to the schema definition cannot be ingested. Invalid data may simply be a result of misspellings during data entry, missing information or other invalid data. When multiple data sources need to be integrated, for example in data warehouses, federated database systems or global web-based information systems, the need for data cleaning increases significantly. This is because the sources often contain redundant data in different representations. In order to provide access to accurate and consistent data, consolidation of different data representations and elimination of duplicate information become necessary to represent and store the data:

> "Data cleansing is the process of analyzing the quality of data in a data source, manually approving/rejecting the suggestions by the system, and thereby making changes to the data. Data cleansing in Data Quality Services (DQS) includes a computer-assisted process that analyzes how data conforms to the knowledge in a knowledge base, and an interactive process that enables the data steward to review and modify computer-assisted process results to ensure that the data cleansing is exactly as they want to be done (Microsoft, 2012)."

A data cleaning approach should satisfy several requirements. First of all, it should detect and remove all major errors and inconsistencies both in individual data sources and when integrating multiple sources. The approach should be supported by tools to limit manual

inspection and programming effort and be extensible to easily cover additional sources. Furthermore, data cleaning should not be performed in isolation but together with schema-related data transformations based on comprehensive meta-data. Mapping functions for data cleaning and other data transformations should be specified in a declarative way and be reusable for other data sources as well as for query processing. Especially for data warehouses, a work-flow infrastructure should be supported to execute all data transformation steps for multiple sources and large data sets in a reliable and efficient way. As argued by David Ruppert an esteemed member of the American Statistics Association when commenting on the inconsistent results in relation to statistical sampling:

> "Data cleansing can be time-consuming and tedious, but robust estimators are not a substitute for careful examination of the data for clerical errors and other problems (Ruppert, 2002)."

Score cards as shown in Table 4.1 are a tool that can determine the range of values for a nominated field typically within a file or database. The tools can also be used to generate a standard set of values for the nominated field and then reapplied to the raw data to standardize the field's value.

4.4 'Schema-First' - Schema Definition Languages

Some schema languages are designed for example purpose, take OASIS The Common Schema Definition Language (CSDL) specification. This was designed specifically to describe the **OData** service. Other languages like DCOM, CORBA and other messaging service languages all have schema definition languages. The structure of X509 Certificates, PKCS7 messages and many other cryptographic artefacts are all described by a Schema Language and in this case it is Advance Syntax Notation 1 (ASN1).

Gender	Frequency
2	3
F	20
M	25
?	5
1	2

Table 4.1: Score card example

Variable Name	Description	Variable Type	Valid Values
ID	Identifier Number	Number	Numerals
Name	Complete name of the individual	String	A-Z, -, space(s)
DOB	The person's date of birth	Date	Dates > 1900
Gender	The gender of the person	Char	Values 'M' or 'F'
Address	The address of the individual	String	A-Z, 0-9, spaces, dash

Table 4.2: Typical data specification

The application of a schema requires the data to conform to a specific set of rules which may not all be encapsulated within the schema. For example table 4.2 is the schema specification used by the 'Schema-First' approach. Any data that does not comply with the specification would be rejected or require some modification. Furthermore, it would not be uncommon to modify data from data source where **gender field** could be '1' for male '0' for female to comply the data specification in Table 4.2 which would be 'M' for male and 'F' for female. If however the data contains '2' which is unknown then this value does not comply and some action is required to conform to the schema specification.

Generally, the schema can be modified dynamically where columns may be added, altered or dropped. However, the modification of relational table schema definitions requires special privileges which may not be granted to the analyst and person performing the data ingestion task. Another consideration is that the schema modification will alter data within the relational table even if involves the addition of a *null* field. Generally, relation data base designs comply to third normal form where data storage is optimized (Biskup, 1987). Code tables are used extensively to reduce storage and simplify data base updates.

4.5 Data Ingestion

Data ingestion is a process whereby data is taken from a initial state or data source and modified so that it can be stored and processed by the analytical indexes or consumed by other systems. The data ingestion is the initial process of the data collation phase of the Intelligence Life-Cycle. Many data base and third-party vendors provide tools to perform this process.

4.5.1 Human Cleansing

Often the data cleansing is a manual process where a human manually trawls through the data and corrects typographical errors or makes some determination of what the data represents. The data for example may be a list of ratepayers in a capital city. The ratepayer may be a household owner (owner occupier) or an organization. The council does not distinguish (or probably care) if the ratepayer is an individual or organization as long as the rates are paid. This does however present a problem where this may matter for an intelligence gathering. Is it David Jones the person or David Jones the organization. If this does matter to intelligence gathering then additional information would be required to determine the correct resolution.

4.5.2 Automated Cleansing

Automated cleansing is where a set of automated rules are applied to the data and can modify, merge or split the data into a format suitable for ingestion. *Regular expressions,* are suitable to determine, match or extract parts contained within a name. A challenge with this approach within the ACC is that the coercive powers **cannot** dictate to an organization the form or structure of the data. Organisations often present data over different time periods in a different structure. This poses a significant impost on any automated process, however the majority of the data does come in the form of a **comma separated file** (csv) which is relatively simple to automate. Some outliers are more difficult to process and may be impossible to automate. This process or technology is referred to as Extract, Transform and Load (ETL).

4.5.3 Extract Transform Load

Extract is the process to obtain data from an external sources. The source may be a relational table, a file system and so forth. The transformation step is the process which describes the modification of the input into form where the data can be loaded into the target system. This step may also reject certain data if it cannot be loaded. The load is the final step where the the transformed data is loaded into the target system.

ETL tools have been around since the mid 1990s and have gained acceptance as a data migration and transformation technology. ETL tools serve two very specific and distinct purposes which are (Henry et al., 2005):

1. To provide a development environment that is easy to manage and embedded into the graphical interface.

2. To provide increased throughput, which increases the productivity of the user.

The increased throughput is achieved by separating data management from data access. In addition, used together, ETL tools provide a complete solution to move data from one system to another.

There is also no guarantee that the data source maintains the same format of the data and is required by law to inform the ACC that the structure or format of the data source has changed. In addition, the organization that supplied the data is not required to provide an explanation of what the fields mean or indicate. This can create problems that can introduce error into the data cleansing process.

4.6 'Schema-First' and the Intelligence Life Cycle

Currently, the 'Schema-First' Approach is the intelligence collection and storage framework chosen by most law enforcement agencies. This approach lends itself to relational technologies and products like IBM's Analyst Notebook and Palantir which take advantage of this approach. In addition, most of the popular ETL tools work well with this platform.

The volume and velocity of data is not slowing and the process is described as:

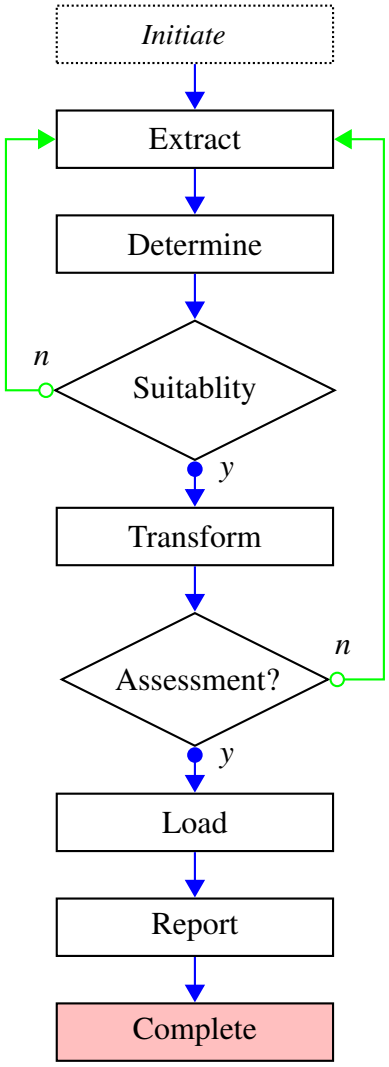


Figure 4.3: Extract Transform Load process

> "...the need to answer questions with data won't go away and access to new data sets won't go way. Instead of worrying about the difficulty of getting clean data, build skills on your team so you can create clean data sets and come up with new insights faster than the competition (Lohr, 2014)."

Timothy Weaver, the chief information officer of Del Monte Foods, calls the predicament of data wrangling data as:

> "**iceberg** issue, meaning attention is focused on the result that is seen rather than all the unseen toil beneath. But it is a problem born of opportunity. Increasingly, there are many more sources of data to tap that can deliver clues about a company's business (Lohr, 2014)."

Data formats are one challenge that the intelligence community must respect and solve. It is not always possible for an organization to manage all the known data formats and there at least a dozen valid **'comma separated value**' varieties. If there is no control over the format of source data and it is left to the discretion of the provider then this adds additional complexity to data collation.

Data is often stored in idiosyncratic formats or designed for human viewing rather than computational processing. The messiness of data which is part of 'Knowledge Discovery' is addressed by the 'Schema-First' approach by removing data attributes that are not captured by the schema. This may in turn lead to misleading conclusions or adversely affect the quality of the intelligence.

4.7 Summary

The 'Schema-First' Approach is the approach taken by many organizations to capture, collate and process intelligence data. This approach utilizes Extract Transform Load (ETL) technology as the foundation of the capture, collection and collation phase of the Intelligence Life-Cycle and plays a significant role in the preparation phase of the CRISP-DM framework. The data cleansing approach can fundamentally change or distort data values which may affect the resultant conclusions made within intelligence based products.

The proposed ‘Schema-Last’ Approach will be shown to be a superior to the ‘Schema-First’ Approach to dealing with data ambiguity and processing. The next chapter will propose the novel approach to the ‘Schema-First’ Approach where the messiness of data is addressed.

Chapter 5

The Proposed 'Schema-Last' Approach

The traditional (largely based on ETL technologies) approaches were no longer, effective within the ACC and a new and novel approach had to be found. In addition, the analyst's demands for data and data availability grew and so did the available data sources. There were a number of attempts made to address the analyst's concern until settling on the 'Schema-Last' Approach (SLA).

This chapter will describe in detail the 'Schema-Last' Approach and this will include:

1. The Data Triage Process.
2. The artefacts contained within SLA Schema.
3. A formal description of these artefacts.
4. A formal notation to represent these artefacts.
5. The relationship between the SLA Schema and artefacts.
6. How RDF can be used to represent Structures and how the RDF List structures can be used to store the data.

5.1 The Data Quality Challenge

The introduction of errors as part of the Collection and Collation phase of the Intelligence Life-Cycle will only result in incorrect analytical results. The application of the schema is important where 'Schema-First' Approach applies the schema at the beginning of the Intelligence Life-Cycle and no change can be made without affecting future collections and collations (see Figure 5.1). The 'Schema-Last' Approach allows the schema to be applied on demand and there is absolutely no necessity to have a *fixed* schema (see Figure 5.2).

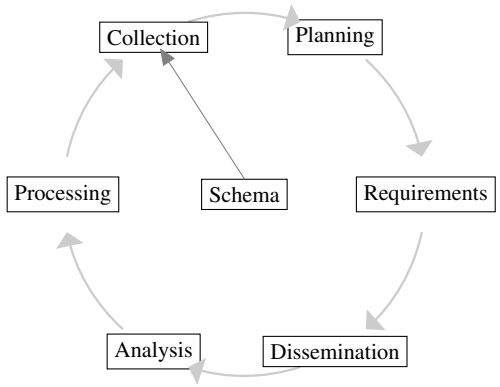


Figure 5.1: 'Schema-First' Approach

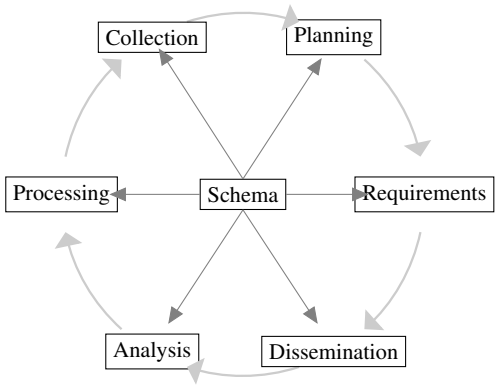


Figure 5.2: 'Schema-Last' Approach

5.1.1 Data Format and Data Cleansing

Data also carries a format or representation which adds value to the data. Data Cleansing will often drop or remove the format and many data miners disregard the format of the data to have any significance. Table 5.1 illustrates the variety of data formats and the significance of the data's format. The 'Schema-First' Approach required that assumptions must be made up front to process data and in turn resolve any ambiguity. If the data did not meet those assumptions then the data was either rejected or modified to meet those expectations.

The 'Schema-Last' Approach deals with ambiguous data differently than 'Schema-First' Approach and does not reject data or impute any missing values if not present within

Data	Cleansed Data	Format	Comment
20 July 2014	20-07-2014	DD Month-YYYY	Standard Australian Format
07-20-2014	20-07-2014	MM-DD-YYYY	US Format
2014-20-07	20-07-2014	YYYY-DD-MM	Arabic Format (right to left)
July 2014	01-07-2014	Month YYYY	Imputed Value - Day

Table 5.1: Date format representation

the data itself. The following chapter will show how applying index strategies will solve many of these ambiguity problems.

5.2 The 'Schema-Last' Approach Specification

A new approach, is to apply the schema only when the data is retrieved. This approach allows for a more dynamic treatment of the data. The relational model can mimic the 'Schema-Last' Approach but is not designed to hold dynamic schema definitions. Generally this approach required each record to be allocated a unique key. The unique key is independent to the actual data. There are two approaches with the allocation of the primary key and these are:

1. Use a time-stamp to identify each record within the data source.

2. Use a Universal Unique Identifier or commonly known as an UUID. The UUID is unique and based on the principle that no other future or past generated UUID can ever be the same. There are currently three commonly used UUID generation algorithms which are time, network, or random number generation based.

The following section will describe the formal process, artefacts, nomenclature and rules that make-up the 'Schema-Last' Approach.

5.2.1 Formal Process Description

SLA provides a formal *framework* to describe the process and the language necessary to describe a data source. The process provides an inherent feedback loop and is an iterative

process. It is always possible to 'start over again' if the SLA schema definition is proved to be incorrect.

The SLA process consists of the following phases which are described in Figure 5.3:

1. **Data source triage**: The coercion of data into a form where it can be uploaded into the data repository. No data is lost or changed during the **triage** process.
2. **Specification**:
   (a) The labelling of all fields contained within a data source.
   (b) The association of each field with a specific domain.
   (c) The creation of models and associated field memberships.
   (d) Schema storage and version management.
3. **Application**: Create indexes based on the schema defined in step 2.
4. **Verification**: Ensure the application of the schema and that erroneous index entries are not created and that the models defined within the schema correspond with the data contained within the data source.
5. **Fuse**: Identify entities which are common within the data source.
6. **Resolve**: The construction of a *'single source of truth*' to represent an entity (generally a person or organization) within the entire data repository.

5.2.2 The Triage Process

Data Cleansing is the transformation of data from a non-canonical to a canonical state. ETL involves the transformation of data into a state suitable for ingestion into a database. This usually requires the standardization of data fields from the original source to a new target format (see Figure 5.4 and Figure 5.5).

For example dates may be transformed from ***mm/dd/yyyy*** to **dd/mm/yyyy** or addresses may be required to have the postcode field removed. Extraneous attributes within a field

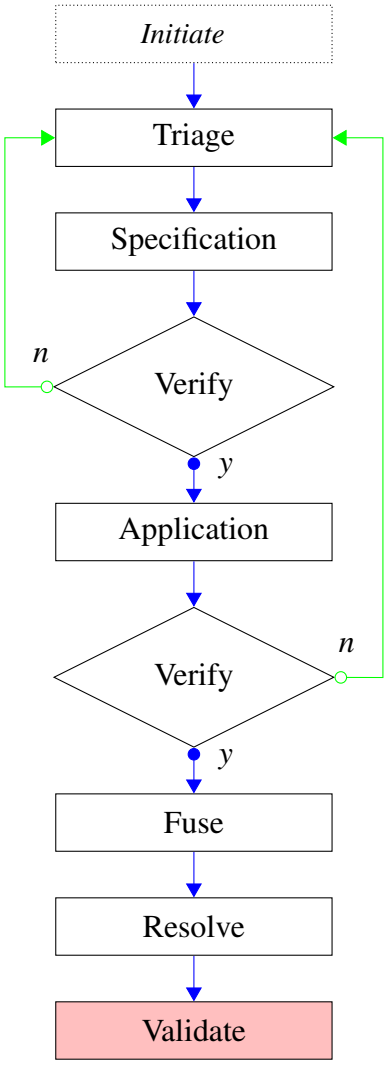


Figure 5.3: 'Schema-Last' Approach - processing

or the entire field are lost due to the cleansing process. For example a person's salutation (MR, MRS, DR, *et cetera*) if contained within a field may be required to be removed to comply to the name field specification. Often the ETL process can lose data or perhaps pervert the original data in some way and may in turn reduce overall quality of the data (Rajaraman, 2014). As argued by N. Brierley, T. Tippetts and P. Cawley:

> "Formal data cleansing can easily overwhelm any human or perhaps the computing capacity of an organization (Brierley et al., 2014.)."

This problem was also identified by Vincent Burner in 2007:

> "that the data volume may overwhelm the Extract Transform Load process and that *data cleansing* may introduce unintentional errors (McBurney, 2007)."

Data Triage is different to ETL in that the raw value of the data is always maintained throughout the transformation process. The data is loaded into the data stores **verbatim** unless there are structural transformation issues with the original data source.

To summaries the differences between the Data Triage and ETL:

- Data Triage does not alter the original data value or format whilst ETL may alter a field's content.
- Data Triage will not eliminate any data field contained within the original data source whilst the result of an ETL load process will eliminate fields that do not comply to a fixed schema.

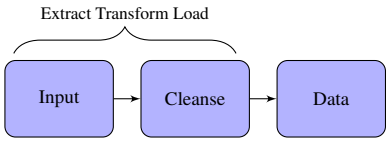


Figure 5.4: Data cleansing

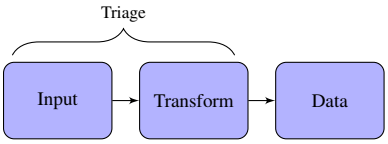


Figure 5.5: Data triage

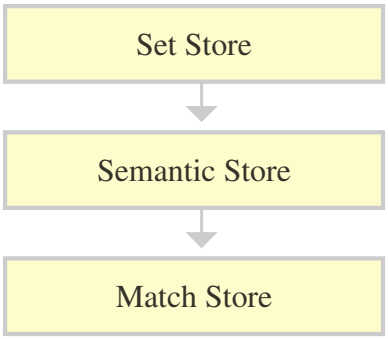


Figure 5.6: The structure hierarchy

5.3 Representational Artefacts

The 'Schema-Last' Approach contains two distinct artefacts. ***Structural*** artefacts define physical structure of the data and how the data will be stored (see 5.6). Whilst ***Representational*** artefacts describe the characteristics of those artefacts. Essentially, ***Representational*** artefacts are meta-data which define the structure of the SLA repository. A ***Conceptual*** artefact is a concept that can be used to represent a notional relationship or object.

The actual physical storage is not mandated and could be any big data implementation that supports both the representational and logical structures.

There are three distinct store types even though the store may reside within the same storage repository. The three stores types are:

Set-Store The Set-Store is the initial container of any data storage which include snapshot, feeds or a complete data store.

Semantic-Store The Semantic-Store builds on the set-store and identifies entities (an entity are the attributes within a set pertaining to an identifiable person or organisation) contained within the set-store. The Semantic-Store also describes relationships between sets contained within a store.

Match-Store The match store establishes link between entities contained within the entity store. In addition to establishing the link, entities are also resolved if it can be established that an entity is the same contained within two independent sets then this relationship is saved within the **match-store**.

5.4 The *Set-Store*

The *set-store* is a first order store of data that can be expressed at a sparse array. The *set-store* is where the data first arrives, is indexed and made available for general use. Where no modification is required, the data is considered *raw* and *data integrity* is maintained. The data within the set store is never modified. Data may be appended to the end of the store via the *bag* mechanism or entire store and bags may be dropped from the repository. It is important that data provenance is maintained if sets with the store are referred to by either the semantic or match-store. If this is the case then this data must never be dropped otherwise the data lineage is broken.

Whilst the underlying data cannot change, additional knowledge about the data may change. This *additional* knowledge can be reflected in changes to the Schema.

5.4.1 Physical Artefacts within the *Set Store*

Physical artefacts are artefacts that *are* a physical item or data element and will consume storage within a set store. There is no restriction on the number of repositories, stores, sets, bags and cells.

Repository: The repository is a collection of stores. There is no restriction to the number of stores contained within a repository. There may be one or more repositories.

Store: The Store is a container of bags. The set order is preserved so that the sets may be retrieved in that order. A store is analogous to a relation table. A store may have only one associated schema. This schema may change over time, however the content of the store cannot change.

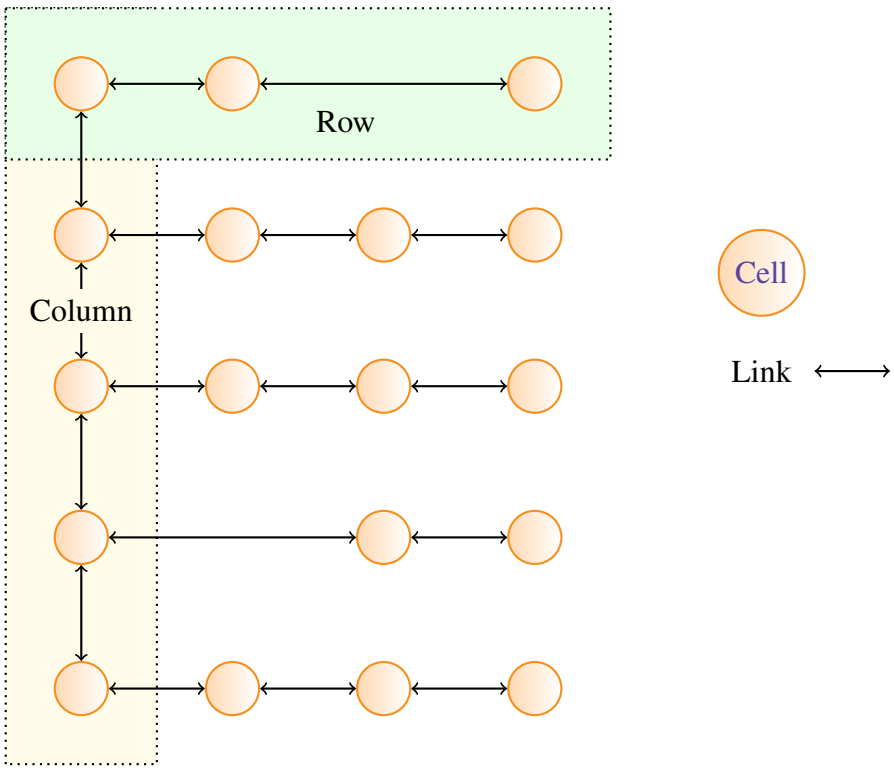


Figure 5.7: Set store ‘list’ structure

Set: The Set is a container for cells. The cell order is preserved so that the cells may be retrieved in that order. A set is analogous to a row within a relational table. Unlike a row within a relational table the set order within the store is important. Sets are uniquely identified by a Set Identifier.

Bag: The store is a container for one or more bags. A bag represents the addition of a logical group of sets that naturally belong to each other. Generally, sets are not individually removed from a store. The premise behind the bag is to capture *snap-shot data* as a single instance.

Cell: The cell is the *atomic* data item. A cell is identified by its position within a set. A cell can either be identified by a label or by its position.

5.4.2 Representational artefacts within the *Set Store*

Representational Artefacts are artefacts that *represent* a physical item or data element within a set store. All representational artefacts are optional, however there are inter-dependencies between some of the artefacts for example a model cell must have an associated domain.

Label: A label is a title for a specific field usually assigned by the person or organization that initially created the data source. This would be the column name if the data was originally from a spreadsheet and as such does not have to be unique within a store.

Domain: A domain is an optional range of values - domain - to all the cells within a nominated position within a set. The domain is the logical set of values for a specific cell at that position.

Map: A map contains the relative location of each label contained within the store. Each entry in a map must contain a label. The position of the cell within the map determines the cell's domain and relative model position.

Model: A model is a logical group of fields (see Figure 5.8) . A model may hold fields contained within another model. Models that do not hold fields from other models are said to be **distinct** (Figure 5.9). Models containing all the fields held within another model are said to be encapsulated (Figure 5.10). There is no restriction on the number of encapsulated or distinct models contained within a SLA schema. Models may share fields within the same schema definition (Figure 5.11).

The same model definition may reappear in another data source schema. It is also possible that two or more data sources have identical SLA schema definitions. The model definition is not fixed, at anytime new knowledge about the data source is known, this may affect the model definitions contained within a schema.

5.4.3 Conceptual Artefacts

There are artefacts that do not have a physical representation within the *schema* but become more important when further processing is required for entity extraction and data matching. Conceptual artefacts are used as input to the Indexing Strategies and used to define the schemas used by the indexing engines. Conceptual artefacts can change over-time and are not set in stone at inception.

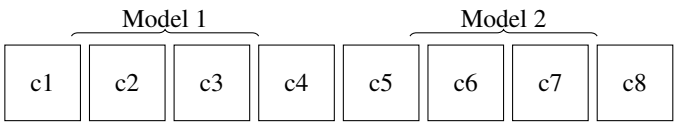


Figure 5.8: 'Schema-Last' models

Domain Group: There are domains that *naturally* belong together and may be used by itself but do not make sense when used in conjunction with other domains. An example of a domain that falls into a domain group would be **month-of-birth** would require both a **Day-of-Birth** and a **Year-of-Birth** to make a complete **Birth-Date**. In some cases especially with message data that is associated with intelligence data that the data source may only contain the **Year-of-Birth** without the Day and Month. All three domains then make up the domain group **Birth Date**. As another example, a model may contain four domains the

Group Domain	Domains	Description
Date-of-Birth	Birth-Day Birth-Month Birth-Year	A person's date of birth
Address	Unit/Flat Number Street Number Street Name Suburb City State Postcode Country	An established address

Table 5.2: Group domains

street number, the street name, the postcode and the country, and if combined these four domains form, the complete address (see Table 5.2).

If the data set contains Geo-spatial coordinates then this domain would logically be associated with an address domain.

Domain Array: The domain array is where a model contains multiple occurrences of a single domain. An example of this would a telephone number where it is not uncommon for a person to have both a mobile and home telephone number. A domain group cannot also be a domain array.

In the case where multiple domains are required to identify a group for example, an address, can be achieved through the use of a **model group**.

Model Group: The model group is a logical collection of domains that may exist one or more times within a model. There is no representation of a model group but it typically applies to *multi-domains* for example an address. A model group is a logical collection of cells that relate to a particular entity. If the model is of a person then the model would contain the person's name, date-of-birth and address. However, if the model represents an organization then this would contain the organization's name, address and Australian Business Number (ABN). Models may only contain one model group of that group type and

if the group is different, such as a person's name and their address then this is permissible. However, if the person has multiple address then a separate model must be created for each address but contain the person's name and **date-of-birth** if present.

Store Column: The Store Column is the set of cells at a specific position within all sets. A column has assigned a mandatory label and optional domain. The column's label and domain applies to all cells at the column's position. In addition, the column position is used to identify the cells and any associated domain within a model (see Figure 5.13).

$$C(S) = \{c_n | c \in s_c\} \tag{5.1}$$

Store Identifier: All set stores must have an unique identifier to identify and be able to locate the store within the repository. The Universal Unique Identifier or UUID is an accepted approach to generate unique identifiers and is used as the set identifier within all reference implementations. As long as the store identifier is unique there is no restriction on the format or value of the store identifier. It is not uncommon for datasets received by the agency not to contain a primary key. The store identifier can be used for this purpose and if the data source does have an identifiable primary key then *meta-data* can be used to indicate that an external primary key is available.

store	C_1	C_2	C_3	C_4	C_5
s_1	c_1	c_2	c_3		c_5
s_2	c_1	c_2	c_3	c_4	c_5
s_3	c_1	c_2	c_3		c_5

Figure 5.13: Row/column representation

Store Folders: Folders are created to logically group stores together (see Figure 5.14). The folder structure could include any number of stores or another store folder. Therefore, folders provide a hierarchical conceptual representation of the stores and provide the capability to group stores together.

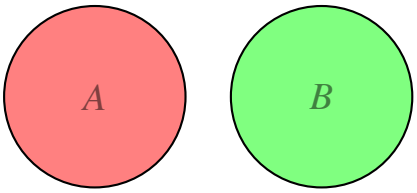


Figure 5.9: Distinct models

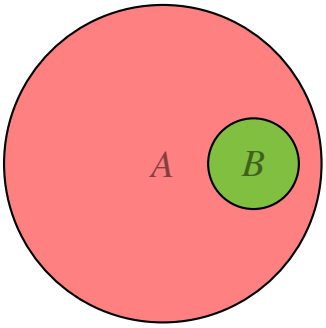


Figure 5.10: Encapsulated models

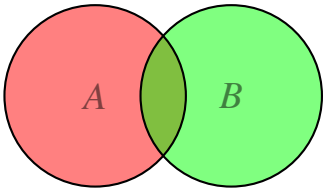


Figure 5.11: Overlapping models

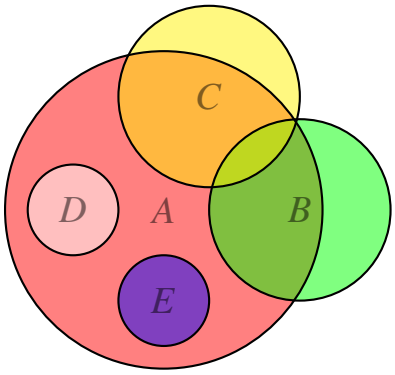


Figure 5.12: Complex model specification

5.4.4 Formal Definition

It is important to formally define the interaction and structure of the artefacts that makeup the SLA framework. Table 5.4 defines the symbolic representation to describe the 'Schema-Last' Approach. The Store is a tabular representation of partitioned data which is described by a single schema definition. The schema definition can be modified at any time. Even though a store has many similarities with a relational table there are also significant differences. The Schema is not fixed and can be modified at any time without affecting the composition of the data. The data within a store is partitioned into bags. Bags are a sparse matrix of *sets* and *cells*. Sets maintain a position within a *bag* (starting from 0) and this is referred to as a positional set and identified by $s_n$where n is the position within the *bag*. The position reflects the relative row position within the data source if the data source is a spread-sheet or comma separated value. Sets contain cells which also contain a position and are identified by $c_n$where n is the position within the *set. Sets* and *cells together m*ake up the sparse two dimensional array. The value within a bag can be expressed where:

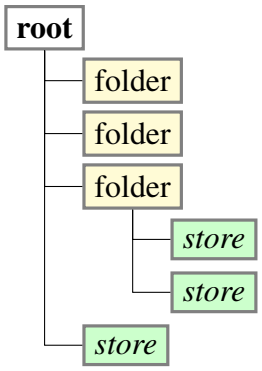


Figure 5.14: Folder Structure

$$b[s_n, c_n] \tag{5.2}$$

Therefore value ***fred*** at *set* position 4 (s_4) and *cell* position 3(c_3) would be expressed as:

$$b[s_4, c_3] = fred \tag{5.3}$$

Or as an array specification:

$$b[4,3] = fred \tag{5.4}$$

Symbol	Name/Function	Description
$\mathbf{R}$	Repository	The repository of stores.
b	Bag	A bag is a container of sets.
$S_{\mathbf{b}}$	Store Bags	All the Bags contained within a store .
c	Cell	A cell is a container for a single value.
c_n	Positional Cell	A cell position within a set.
C_n	Positional Column	A column at position $\boldsymbol{n}$ within a store.
S	Store	A store with all its contents.
S_{id}	Store Identifier	Uniquely identifies a store.
s or $\{\}$	Set	A set is a container of cells.
$\{\varnothing\}$	Empty set	A set with no cells.
s_n	Positional Set	A set at position $\boldsymbol{n}$ within a bag.
S_s	Store sets	All the sets contained within a store .
s_c	Set cells	All the cells contained within a set.
b_n	Positional Bag	A set at position $\boldsymbol{n}$ within a bag.
l	Label	A label is a short description/moniker associated with a cell.
l_n	Positional Label	A label at position $\boldsymbol{n}$ within a set.
$\mathbf{D}$	All Domains	All defined domains.
d	Domain	A domain describes the type of data that may be contained within a cell.
S_d	Store Domains	All the Domains contained within a store.
d_n	Positional Set Domain	A domain at position $\boldsymbol{n}$ within a set.
S_l	Store Map	All the labels contained within a store
m	Model	A model is a sequenced set of domains.
m_n	Positional Model	A model at position $\boldsymbol{n}$**.**
S_m	All Models	All the Models contained within a store.
t	Tag	Tag represents a Name/Value pair.
c	Undefined cell	Cell that has **no** domain and not contained in a model.
c_n	Undefined store cell	All cells that are within a store but **not** contained in a model or have **no** associated domain.
$c^{\dagger}$	Defined cells	A cell that has a domain.
$c_n^{\dagger}$	Defined store cell	All cells within a store that have an associated domain (not those **not** contained in a model).
$c^{\ddagger}$	Assigned cell	A cell that has a domain and contained within one or more models.
$c_n^{\ddagger}$	Assigned store cell	A cell that has a domain and contained within one or more models.

Table 5.4: Symbols and nomenclature

Because a *bag* is a *sparse array* it is possible that there is no value contained within a bag at a particular position. This represented by the$\varnothing$symbol or nothing. Bags also have a relative position within a Store therefore a specific value within a store can be represented as:

$$S[b_{0,}s_4, c_3] \tag{5.5}$$

Therefore value ***fred*** at bag position 0 (b_0) *set* position 4 (s_4) and *cell* position 3(c_3) would be expressed as:

$$S[b_0, s_4, c_3] = fred \tag{5.6}$$

Or as an array specification:

$$S[0,4,3] = fred \tag{5.7}$$

5.4.4.1 Set Definition

A set can be expressed as:

$$s = \{c|c \in s_c, s \in S\} \tag{5.8}$$

Where: s is the set of 10 cells and $c_n$where n is the relative position of each cell within the set.

5.4.4.2 Store Equivalence

The store's domains determine whether a store share domains.

$$S^{'} \equiv S^{''} \Rightarrow \{S^{'}_d|d \in S^{''}_d, d \in D\} \tag{5.9}$$

Rule -S1 : Sets from two different sources are equivalent if the stores are equivalent:

$$s^{'} \equiv s^{''} \Rightarrow S^{'}_s \equiv S^{''}_s \tag{5.10}$$

5.4.4.3 Store Map

The store map is defined as:

$$S_l = \{l_1 ... l_n\} \tag{5.11}$$

Rule-M1 : The labels (store map) within store $S_l^{'}$ and $S_l^{''}$ are not equivalent:

$$S_l^{'} \not\equiv S_l^{''} \tag{5.12}$$

5.4.4.4 Label Definition

Each Field - c_n can have an associated label - l_n :

$$c_n \rightarrow l_n \tag{5.13}$$

Rule-L1 : A label at any position from different sets are not equivalent:

$$S^{'} \rightarrow l_n \not\equiv S^{''} \rightarrow l_n \tag{5.14}$$

5.4.4.5 Model Definition

Which contain two models m_1 and m_2

$$S^{'} = \{m_1, m_2\} \tag{5.15}$$

m_1 contains:

$$m_1 = \{c_1, c_2, c_3, c_4, c_5\} \tag{5.16}$$

$m_2$contains:

$$m_2 = \{c_1, c_2, c_6, c_7, c_8\} \tag{5.17}$$

The union of $m^1$and m^2can be defined as:

$$m_1 \cup m_2 = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\} \tag{5.18}$$

The intersection of $m^1$and m^2can be defined as:

$$m_1 \cap m_2 = \{c_1, c_2\} \tag{5.19}$$

The unassigned cells in $S^{'}$are as follows:

$$S_c = \{c_9, c_{10}\} \tag{5.20}$$

5.4.4.6 Domain Definition

A domain is related to field in that a field can only ever have assigned a single domain:

$$d^n \rightarrow c^n \tag{5.21}$$

Therefore for S:

$$S_d = \{c_d | c \in S_c, d \in D\} \tag{5.22}$$

A set may can contain one or more domains:

$$\{S_d | d \in D\} \tag{5.23}$$

Rule-D2: The same domain contained within two different sets are equivalent:

$$S^{'} \rightarrow d_n \equiv S^{''} \rightarrow d_n \tag{5.24}$$

5.4.4.7 Domain-Model Definition

The same applies if two models have the same domains then the equivalent:

$$m_1 = \{d_1, d_2, d_3\} \tag{5.25}$$

$$m_2 = \{d_1, d_2, d_3\} \tag{5.26}$$

Rule-M1: The same models contained within two different sets are equivalent:

$$m_1 \equiv m_2 \tag{5.27}$$

5.4.4.8 Cell-Model Membership

For a cell to be a member of a model this must be true where P is the cell-model membership function:

$$\forall c \in S_c, \exists c \in S_{c^\dagger} : P(c) \tag{5.28}$$

5.4.4.9 Domain-Model Fusion

Two models in different stores $S_m^{'}$and $S_m^{''}$are said to be equivalent if and only if:

$$\{d : d \in S_m^{'}\} \cap \{d : d \in S_m^{''}\} \neq \{\varnothing\} \tag{5.29}$$

5.4.4.10 Distinct Models

Two models in stores $S_m^{'}$and $S_m^{''}$ and P is the cell-model membership property function are said to be distinct if:

$$\forall c \in S_c^{'}, \exists c \notin S_c^{''} : P(c) \tag{5.30}$$

5.4.4.11 Overlapping Models

Two models property in stores $S_m^{'}$and $S_m^{''}$and P is the cell-model membership function are said to be overlapping if:

$$\forall c \in S_c^{'}, \exists d \in S_c^{''} : P(c) \tag{5.31}$$

5.4.4.12 Store Fusion

Rule-F1: Store $S^{'}$ and $S^{''}$ and P is the domain membership function can only be fused if the stores have at least one domain in common:

$$\exists d \in S^{'}_{d}, \exists d \in S^{''}_{d} : P(d) \tag{5.32}$$

5.4.5 Label Allocation

The label is assigned by the data source provider to identify the cell at a specified position. Usually this is a short description such as as *supplier name*, *registered-owner*, *date of incident* and so on. As stated in ***Rule-L1***, labels from different data sources are not equivalent even if the labels share the same value. Labels should never be modified by the receiving *uploader* no matter how cryptic the label's value. It is important that the data provider assign meaningful labels to describe the positional cells contained within the data. If the data provider chooses not to provide the label then the domain for the *positional cell* may suffice.

5.4.6 Domain Classification

The domain identifies a cell content. Unlike 'Schema-First' the domains are not fixed but suit the ontological requirements. For example, a domain may be a first-name, date-of-birth, address-line and post-code. The classification of domains excludes primitive data types which are: string, integer, byte, char, *et cetera* (see Equation *5.33*). Therefore a domain represents the set of values for a particular cell. Furthermore, the domain is used to relate fields from different data sources. Correct usage of domain classifications provides a capability to fuse data sources together. Domains are compatible across stores and it is important that domains are chosen carefully because once assigned domain usage cannot be changed.

Unlike the *label*, domains are allocated by the provider of the data source it is up to the analyst or person to determine the domain of a positional cell. This may be achieved by

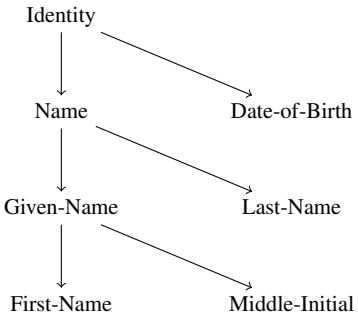


Figure 5.15: Typical ontology support

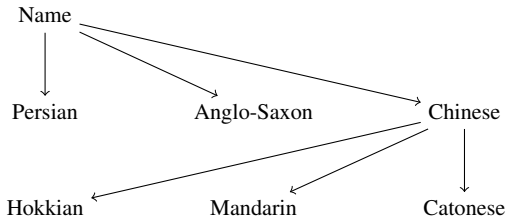


Figure 5.16: Cultural ontology

random data samples of the positional cell's value and making a determination. The cell's *label* may also provide a clue of the positional cell's domain.

The positional cell's domain is not fixed, and can be changed later if it is proven to be incorrectas as long as domain d the domain is member of all known domains D :

$$d \quad \in \quad \mathbf{D} : P(d) \tag{5.33}$$

5.4.7 Domain Ontological Structure

Not many data modelling products have Ontological support, however W3C has specified Ontology Web Language (OWL) which defines a comprehensive ontological language (W3C, 2012). OWL allows the definition of hierarchical ontological structures based on the RDF specification. SLA can also assist in the ontological description of the data in regards to domain classification as shown in Figure 5.15 and OWL could be used to describe the ontological structure of the domain. The ontological structure should be kept simple unless there is a need to refine a domain if somehow it does not make sense in the context it will be used. In turn, the domains will drive the indexing strategies.

5.4.8 Cultural Ontology Classification

Apart from *normal* ontological structures, ontological definitions may be sensitive to cultural variances. The ACC has identified various groups, some based on a person's race, club affiliation, religious institution or any other group or organization (see Figure 5.16). Any information pertaining is useful in determination of the index strategy to help determine a person's identity and cultural characteristics (Price, 2007).

5.4.9 The Logical Schema

To choose a language for the **Schema** to represent the artefacts it is important that it utilizes standard descriptive languages. XML is one such option in that XML has been used to express many standards including National Information Exchange Model (**NIEM**) and the Health Standard for information exchange **HL7v3** and there are many other examples. However, the Resource Definition Framework (RDF) provides a platform to represent any data structure including SLA - Schema Definitions.

RDF is a specification that represents knowledge and with the addition of graphs or context allows data to be classified. In addition, the blank node allows for the seamless grouping of schema artefacts which include the models contained within the map and the map itself. The RDF **'Sequence'** (see Figure 3.3.1) can represent a store map where the RDF item's sequence number is the relative position of the label. The same applies for the model which can also take advantage of the RDF **sequence** structure so that the domain position can also be represented by the RDF sequence number (see Figure 5.17).

In addition, the schema RDF representation can be easily shared or even managed by source management systems such as CVS, Subversion, GIT, *et cetera*.

The schema definition is formally represented in RDF N3 form (W3C Working W3C, 2014). RDF blank nodes are used to group the fields that are contained within a model. Figure 5.17 is a simple schema definition that contains three fields and two models.

```
@prefix rdf: <http://www.w3.org/...#>
@prefix rdfs: <http://www.w3.org/...#>
@prefix schema: <http://www.sla.org.au/...#>

<file:/schema.ttl> schema#:schema
 [
 <schema#://model>
   [
    rdf:#_1 _:b1 ;
    rdf:#_2 _:b2 ;
    rdf:_#3 _:b0
  ];
 <schema#://model>
  [ rdf#:_0
   _:b0 , _:b2 , _:b1 ;
    rdf#:_1
   _:b0 , _:b2
  ]
 ] .
_:b0 rdfs#:domain "document-id" ;
   rdfs#:label "document-name" .
_:b1 rdfs#:domain "first-name" ;
 rdfs#:label "given-name" .
_:b2 rdfs#:domain "last-name" ;
 rdfs#:label "surname" .
```

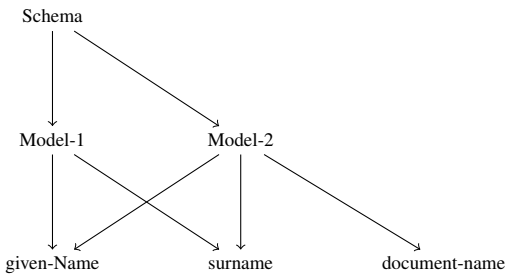


Figure 5.17: Schema definition and visual representation

5.4.10 Meta-Data

Good meta-data conforms to community standards in a way that is appropriate to the collection (*store*) and current and potential future uses of the collection. It is essential to conform to, or at the very least map to, known local and international standards for meta-data, rather than using proprietary or homegrown schemes. However, simply because a particular meta-data scheme is considered a standard does not necessarily mean that it is the appropriate standard for any given collection. It is a well-established standard for describing intact archival (see Figure5.18) collections with a common provenance, but it is not the best scheme for describing heterogeneous object compositions that have different data sources.

Good meta-data should be coherent, meaningful, and useful in both organizational and global contexts beyond those in which it was created. This means that it must include all pertinent information about the object, since assumptions about the context in which it is accessed locally may no longer be valid in the wider networked environment. For example, a collection of asset ownership may not indicate the year or why or how the asset

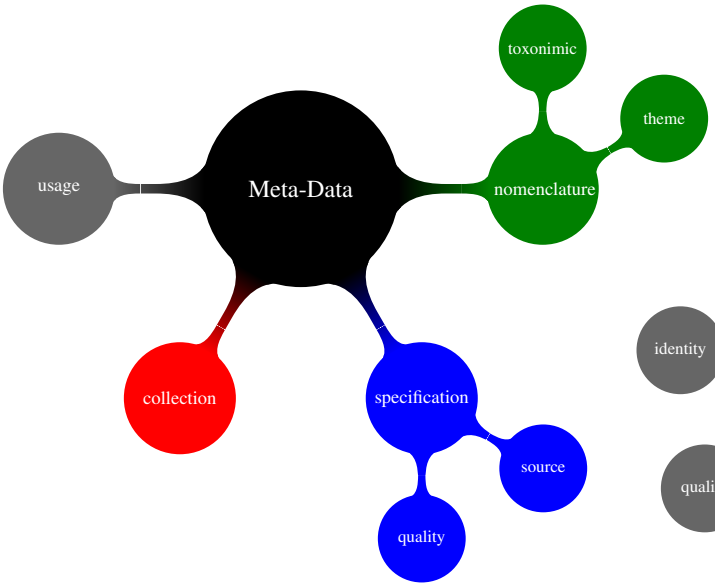


Figure 5.18: Meta-Data descriptions

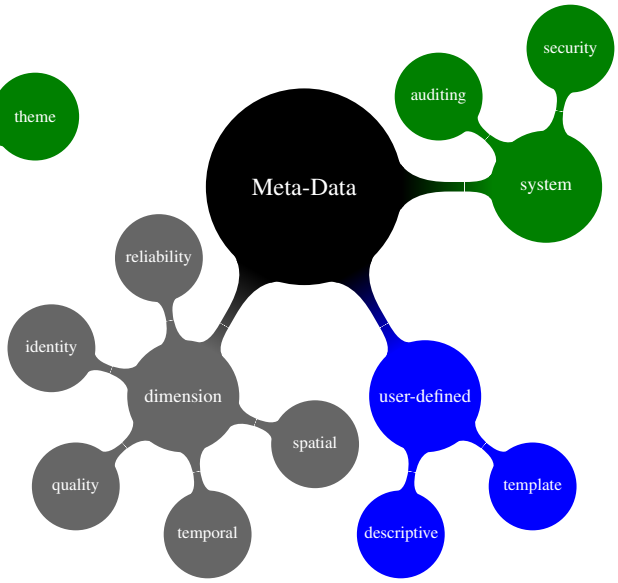


Figure 5.19: Meta-Data domains

was first purchased. However, in the wider network context, form and quality information becomes important. Digital collections with a topical focus are notorious for creating non-interoperable meta-data when they assume that users know the main topic of the collection. When this meta-data is shared in larger aggregations, descriptions that made sense in the context of the original collection can be mystifying. This has been dubbed the 'on a horse' problem, from the description of a photograph in Harvard's Teddy Roosevelt collection, where the title assigned to the photograph did not indicate who was sitting on the horse, since all the materials in the collection related to Roosevelt.

The creation of accessible, meaningful shared collections implies responsibilities on the part of both the data providers (organizations that create meta-data records and contribute them to federated collections) and service providers (aggregates that provide access to federated collections or union catalogues). Data providers should strive to create consistent, standards-based meta-data, to use appropriate controlled vocabularies and thesauri, and to follow appropriate data content. Service providers must implement meta-data normalization, remediation, and enhancement, and should, as their name implies, provide additional *value-added* services such as vocabulary-assisted searching, subject clustering,

terminology mapping, and other enhancements. Adherence to appropriate standards and collaboration between data providers and service providers are crucial elements of effective aggregated digital collections.

5.4.11 ISO Standard 11179-1

ISO Standard 11179-1 describes meta-data as a kind of data that describes one or more objects. Objects can be of any type, such as books (e.g., resource meta-data), databases and data itself (e.g., data semantics meta-data), documents (e.g., records management meta-data), equipment (e.g., device meta-data), and data governance itself (e.g., registration meta-data). Good meta-data practices (consistent observation, consistent meaning, and useful cataloging taxonomies) can provide valuable benefits to the enterprise, such as:

- Better search and discovery of desired objects (e.g., documents, data, services, etc.) and their related objects (e.g., administrative services).
- Automated and semi-automated processing of on-demand data assets, e.g., using data assets in meaningful ways without prior knowledge.
- Better re-use of one's objects (e.g., data assets) via proper cataloging (applying and tagging meta-data) and discovery by other users.

Any meta-data encapsulated within the set store must comply to ISO Standard 11179-1.

5.4.12 Meta-Data and Provenance

The life-cycle of meta-data, like data, is created, processed, stored, and communicated. The life-cycle of meta-data has common features, not all of which are shared by general data. There are key stages in the meta-data life-cycle and these stages do not necessarily imply a particular business, information, or technology process. Meta-data is created when a descriptive relationship is revealed between a datum (now to become meta-data) and an object (the datum describes the object). Usually creation meta-data is never changed once created and typically, descriptive relationships are not created accidentally but are created deliberately through a business, information, or technology processes. In other words, it

is rare that meta-data is created free-form without any guidance or constraints of what can/should be recorded. Typically, meta-data can take the form of:

- keywords,
- names or titles,
- owners or authors (their system user identifier),
- descriptions,
- contextual information, for example into a record that is embedded within, adjacent to, or associated with a target object.

5.4.13 *Container* Level Meta-Data

In some cases it may be important to restrict the tag name to the related or group stores. For example, if certain stores belong to a particular investigation the tag name and value can be used to define this relationship (see Figure 5.19). In addition to restricting tag names, tags can be broadly broken into the following classifications:

User: These are tags allocated by the person who initially loaded the data to annotate the data source.

Template: A template is used to format the ***set*** data contained within a store for reporting purposes. An example would be XSLT template or an **R** program that processes the set data.

Descriptive: Tags that describe the users of this data or business processes. Descriptive tags are short textual narratives.

Identity: To be able to locate a store within a repository if there are hundreds or even thousands contained within it can be difficult. This group of tags allows the quick location of store within a repository.

Quality: A sub-group of descriptive tags is data quality which can be used as input to the data-matching algorithms.

Symbol	Name/Function	Description
T	Tags	The set of tags name.
t	Tag	An assigned or unassigned tag.
t_n	Store Tag	A tag assigned to a store.
S_t	Store Tags	All the tags assigned to a store.
B_t	Bag Tag	A tag assigned to a bag.

Table 5.6: Tags, symbols and nomenclature

Security: These tags are used to restrict access to the store data.

Dimension: These tags are generally used to define **where** data came from and where the data reflects a geo-spatial location which could be in the form of a geo-spatial polygon.

System: These are tags automatically created by the system for auditing purposes. For example the date and time the data was uploaded.

5.4.14 Meta-Data Tags Formal Definition

Descriptive meta-data can form part of the schema definition. However it is important to formally define the relationship between tags to stores and bags. Tags can only be assigned to container artefacts which are stores and bags.

Rule-T1: The same tag contained within two different stores ($S^{'}$ and $S^{''}$) are equivalent:

$$S^{'} \rightarrow t^{'} \equiv S^{''} \rightarrow t^{'} \tag{5.34}$$

This rule applies to all stores with that tag. In addition, the tag name and value may be copied from one store to another.

Rule-T2: The same tag contained within two different bags ($B^{'}$ and $B^{''}$) are equivalent:

$$B^{'} \rightarrow t^{'} \equiv B^{''} \rightarrow t^{'} \tag{5.35}$$

5.4.15 Storage Considerations

There are several big data implementations each of which can support the 'Schema-Last' Approach to data modelling. The data sources obtained by the ACC can be subject to interpretation and it is important that the interpretation is represented by the schema. In addition, it is important that the schema can be modified without affecting the underlying data. The schema must be able to capture ambiguity within the data. It is not always possible to classify a data item as person or organization name. A contact address may be a phone number or email address. This classification problem can be overcome in a number of ways, either through a flexible schema approach or through the utilization Extract Transform Load (ETL) technologies.

The SLA demands suitable storage platform to store both the store's schema and data. Furthermore, the data (the store's sets could number in the millions) must be also be able to be retrieved in a specified order or the order on ingestion. Therefore a sequence number must be incorporated into each set to allow for the retrieval of a set in a pre-determined order.

5.4.16 Provenance and Storage

The ACC requires that the store's data be returned in the original form as it was acquired. This is to be able to identify the row within a spread sheet or reproduce the spread sheet assuming the data was *collected* that way. Therefore the set order must maintain:

R - is the retrieval function
S - is the Store

$$R(S) = \{s_1 ... s_n\} \tag{5.36}$$

The meta-tags would also contain information to retain the provenance of the data and this would include:

- The date and time the data was collected.
- The name or an identifier of the source.

artefact	RDF Structure	RDF Example
Store	**base**://store/<id>	sla://store/189090748374
Bag	**base**://bag/<id>	sla://bag/189090748374
Set	**base**://set/_<pos>	sla://set/_000100
Cell	**base**://cell/_<pos>	sla://cell/_<pos>

Table 5.7: RDF representation of SLA artefacts

- How the data was obtained. For example was the ACC required to invoke coercive powers to obtain the data?
- Any caveats associated with the data. An example of a caveat would be restrictions on how the data can be used and who has access to the data.
- The owner or owners of the data.

5.5 RDF Representation

RDF is a flexible standard notation and can represent any data structure. As such, it is ideal to represent the *set store* artefacts. RDF resource is expressed as a Universal Resource Identifier (URI). The URI consists of three components, the first being the base or protocol. This is user defined, however **http** and **rdf** have special significance. The second part which is followed by the '**://**' is the artefact type. The final part of the resource is an identifier or a position. RDF uses the underscore character '_' to identify a numeric value otherwise the URI is invalid since numbers cannot follow a back slash.

5.5.1 RDF List Structure

The RDF triple store allows the storage and retrieval of any data structure and well suited to store the 'Schema-Last' artefacts see Figure (5.3). Therefore, the triple store can store both the data and schema structure. Ideally, a triple store *graph* would only contain a specific store's data and schema structure. The RDF list structure would contain the store's sets where a set would be an item within the list. In addition, the RDF list itself would retain the set order.

```
@prefix ex: <http://example.org/>.
@prefix rdf: <http://www.w3.org/...#>
ex:list1 rdf:first [
     rdf:_1 "Basement Jaxx".
     rdf:_2 "2004".
     rdf:_3 "Good Luck".
   ].
ex:list1 rdf:next _:a01.
_:a01 rdf:first [
     rdf:_1 "Groove Armada".
     rdf:_2 "1996".
     rdf:_3 "Superstylin".
   ].
_:a01 rdf:next _:a02.
_:a02 rdf:first [
     rdf:_1 "Fat Boy Slim".
     rdf:_2 "2002".
     rdf:_3 "Weapon of Choice".
   ].
_:a02 rdf:next _:a03.
_:a03 rdf:first [
     ex:artist "Daft Punk".
     ex:year "2013".
     ex:song "Get Lucky".
   ].
_:a03 rdf:next _:a04.
_:a04 rdf:first [
     ex:artist "Pharrell Williams".
     ex:year "2013".
     ex:song "Happy".
   ].
_:a04 rdf:next rdf:nil.
```

Figure 5.20: RDF list represented in N3 form

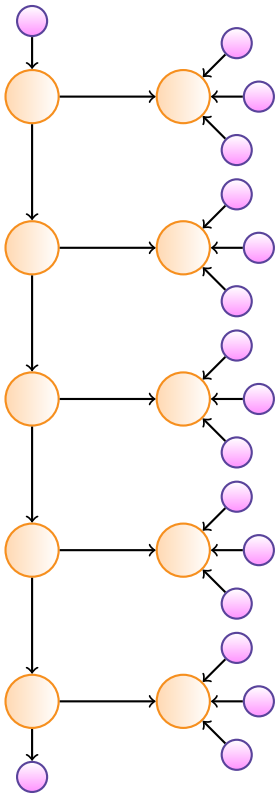


Figure 5.21: A visual representation of Figure 5.20

Syntax Form	Matches
uri	URI or a prefixed name. A path of length one.
^elt	Inverse path (object to subject).
(elt)	A group path *elt*, brackets control precedence.
elt1 / elt2	A sequence path of *elt1*, followed by *elt2*
elt1 ^ elt2	Shorthand for *elt1 / ^elt2*, that is elt1 followed by the inverse of *elt2*.
elt1 | elt2	A alternative path of elt1, or elt2 (all possibilities are tried).
elt*	A path of zero or more occurrences of *elt*.
elt+	A path of one or more occurrences of *elt*.
elt?	A path of zero or one *elt*.
elt{n,m}	A path between *n* and *m* occurrences of *elt*.
elt{n}	Exactly *n* occurrences of *elt*. A fixed length path.
elt{n,}	*n* or more occurrences of *elt*.
elt{,n}	Between *0* and *n* occurrences of *elt*.

Table 5.8: Property path expressions

5.5.1.1 SPARQL 1.1 Property Path Expression

SPARQL 1.1 introduced the *property path expression* which simplified the navigation of the RDF list structure. The *property path expression* can be used to search any directed graph and is particularly useful to extract items from the RDF list structure. The rationale for the introduction of the *property path expression* as stated by the W3C SPARQL working group is as follows:

> "Property paths allow for more concise expression of some SPARQL basic graph patterns and also add the ability to match arbitrary length paths (W3C, 2010)."

Table 5.8 summarizes the extension and how the Property Path extension can be used to navigate an RDF Graph Structure. It is important to note that property path expression can travel forward or backward along a path within a graph.

The property path allows for the traversal of the RDF linked list structure and therefore able to retrieve the *sets* in a specified order. If the data was ingested via a spread-sheet then this spread-sheet could be reproduced. In addition to the set store data meta-data can also be stored alongside the data in RDF form.

5.6 Additional Processing Requirements

The set store can be used to store a variety of data types which include snap-shot, feed, Geo-spatial and so forth. Certain artefacts within the set store were created for a particular storage type. In addition, the set store should also accommodate data destruction or partial data destruction of a set or sets.

5.6.1 *Feed* Management

Not all data is collected at the same time and may in fact come in parts or as a *feed*. To accommodate this requirement the *bag* artefact allows a subset or entire feed to be appended to store and can be treated as an independent collection. Bags will also adopt the model, domains, meta-data contained with in a store. However, bags may contain additional meta-data to describe characteristics about the bag.

5.6.2 Data Disposal

The bag represents a single instance of a captured data. There are times where all instances of the data must be removed as if the data never existed. The *bag* provides the capability to remove all the sets from one captured data instance.

5.7 The Semantic Store

The set store contains the *map, domains* and the raw data. The maps and models give the data context before any further processing begins, but do not contain the relationships between entities within the same set or entities contained within the same store. The set store is in effect the *first port of call* to collect and analyze all intelligence data. The **semantic store** identifies relationships between entities within the same **set store** or between sets **contained** within the same **set store**. In addition, a probability or certainty must be *attached* to each relationship since intelligence or data captured though covert means is not always certain. The models contained within set store identify an entity within the set or

other entities contained within the set store and the relationship of both is contained within the set. There may be other attributes within the set to identify the type of relationship.

$$\{s^{'} \rightarrow s^{''} | s^{'} \in S_s, s^{''} \in S\} \tag{5.37}$$

Sets that share a common identifier can be considered to share some relationship. The semantic model is applied to represent relationships between sets within a store. The semantic store defines set relationships within the same store. It is assumed that this relationship is true and confirmed. This relationship is used as input to further analysis and depending on the *nature* of the store allows for a semantic representation of relationships of sets or entities within the store.

Semantic relationships may be stored with or independently to the store. However, the **set** identifiers are used to identify the relationships between sets and their respective entities.

5.7.1 The RDF Schema Specification

RDF schema is a semantic extension of RDF. It provides mechanisms for describing groups of related resources and the relationships between these resources. RDF schema can be expressed in RDF Turtle, N3 or JSON formats. These resources are used to determine characteristics of other resources, such as the domains and ranges of properties.

The RDF schema class and property system is similar in nature to that of object-oriented programming languages such as Java and C#. RDF schema share many of the attributes associated with the schema defined within the 'Schema-Last' Approach. As with the RDF Schema there are also domains, range values and labels. The specification does not prescribe what constitutes a legitimate domain. The specification does however define a simple ontology specification that describe properties associated with a domain. In addition, domains can be a member of a *greater* domain group.

A significant benefit of the RDF property-centric approach is that it allows anyone to extend the description of existing resources, one of the architectural principles of the Web.

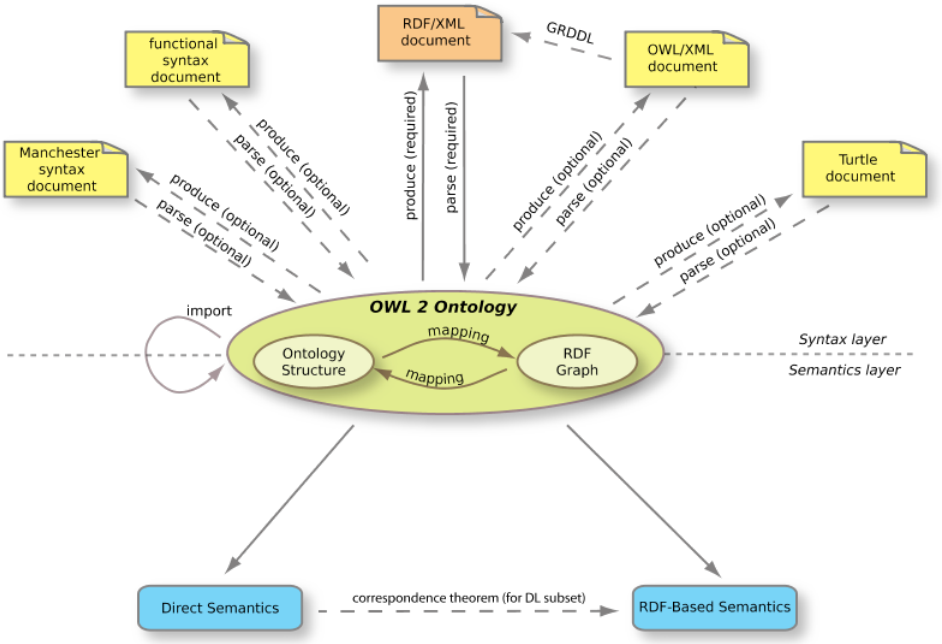


Figure 5.22: Owl ontology specification

5.7.2 The OWL Ontology Specification

Semantic definitions go beyond the set store schema with the models, domains, and the groups (see Figure 5.7). The semantic store applies independent ontological structures to the data contained within the set store. The ontological structures may contain relationships to other cells within other sets contained within *set store*. An example would the OWL ontology specification (see Figure 5.22).

This OWL specification is generally more accepted as a standard, as these testimonials confirm (W3C, 2004):

> "Boeing is a member of the W3C and is an early adopter of RDF, OWL and related Semantic Web technologies. Boeing has a number of projects exploring semantics-based applications in various areas including information and application integration and interoperability, publish/subscribe, knowledge management and network centric operations. These technologies are expected to have

> a strong impact on future Boeing programs. Ontological structures have become fairly widespread in their use and automated reasoning tools are becoming mature. The time is ripe for standards in this area, and for widespread tool support from vendors." *James L. Phillips, Director, Mathematics and Computing Technology, Boeing Phantom Works* (W3C, 2004)."
>
> "As the leading provider of content creation tools to help people communicate better, adding intelligence to media via meta-data was integral to our strategy. We developed Adobe XMP (Extensible Metadata Platform) based on RDF, because it provided a flexible and interoperable framework for fostering the capture, preservation, and interchange of meta-data across digital media and work-flows. The Adobe Creative Suite provides a design platform that enables creative professional to create information rich assets powered by XMP that can be more effectively re-purposed and consumed across multiple media and diverse domains." *David Burkett, Director of Product Management, Adobe Systems* (W3C, 2004)."
>
> "The University of Bristol is delighted to see the publication of the W3C RDF and OWL Recommendations. The University is a strong supporter of open standards and a long-term participant in the RDF work and considers the Semantic Web as important in developing advanced learning and research technologies for education. Successful RDF-based projects at the University include representing meta-data schemas, describing thesauri, events and calendaring, syndicating news, web site annotation and trust and smarter web searching for digital libraries. The University intends to continue developing projects, software and services based on this work." *Alison Allden, Director, Institute for Learning and Research Technology (ILRT), University of Bristol* (W3C, 2004)."

The OWL ontology language provides a comprehensive specification to represent the structure of set store data in an ontological structure. The structure is independent of data and as new knowledge becomes available this may result in a change to the ontological definition. OWL has become *de facto* standard to represent ontological structures set stores and

provides the nomenclature to represent semantic relationships between entities within a set or within the store.

5.7.3 The Palantir Ontology Specification

The Palantir Ontology specification is a comprehensive ontology specification that has been designed to allow for the ingestion of ontological structures into the Palantir product. The Palantir Ontology shares many similarities with the OWL specification. Currently, the Palantir ontology is a proprietary implementation and based on their own XML specification. Their XML specification is available for download from their web-site.

5.7.4 The Role of the Semantic Store

The semantic store provides a structured view to each set within the set store. Whatever ontological specification is decided on, the semantic store allows for the interchange of entities contained within the Set Store with other applications. The resultant entity does not require to reside with or include the set store data but the resultant structure must maintain the lineage back to the raw data by referencing the set identifier. Therefore the role of the semantic store is as follows:

- Provide a mechanism to extract entities from the set store.
- Allow for the interchange of entities between intelligence products.‘

5.8 The Match Store

The set store is the *first port of call* to collect and analyze any intelligence data. The **semantic store** identifies relationships between entities within the same **set store** or between sets **contained** within the same **set store**. In addition, relationships may have an assigned probability or the reliability of assigned relationship. The models contained within the set store may identify an entity or entities within the set. There may be other attributes within the set to identify the type of relationship.

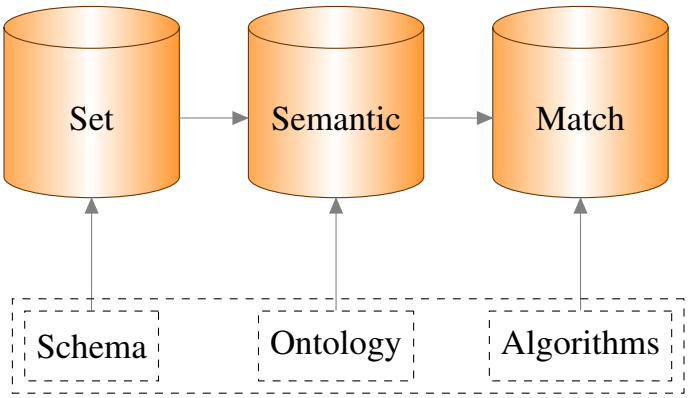


Figure 5.23: The relationships between store classifications

Sets that share a common identifier can be considered to share some relationship. The semantic model is applied to represent relationships between sets within a store. The match store captures the results from the data matching (see Chapter 7) and stores these potential matches as links within the **match store**. These results are maintained as links similar to that which the semantic store represents in relationships described in the previous section.

5.9 Summary

This chapter defined the 'Schema-Last' Approach framework, the artefacts that make up the framework and the relationship between the data and the schema definition. This approach is specially targeted to assist in the collation and process of data gathered as part of the Intelligence Life-Cycle. The three stores provide the basis to analyze and process the data received by the analyst. Figure 5.23 shows the relationship between three store types. The following chapters will describe how these stores are used in data exploration, matching and fusion. The **set** store is the container of all raw data and provides the foundation for further analytical processing and how this can best be exploited by the application of set store models to the indexing strategies. Furthermore, the Schema-Last's artefacts can be applied to both data matching and fusion strategies.

Chapter 6

The ‘Schema-Last’ Approach and Data Exploration

The ‘Schema-Last’ schema provides a framework to build the index structures necessary to locate and exploit data. No longer does the index stand alongside the data or is bound by the storage technology of the data storage systems. Data exploration is an informative search used by data consumers to form true analysis from the information gathered. Often, data is gathered in a non-rigid or uncontrolled manner in large bulks. For true analysis, this unorganized bulk of data needs to be narrowed down. This is where data exploration is used to analyze the data and information from the data to form further analysis.

Data often converges in a central warehouse called a data warehouse. This data can come from various sources using various formats. Relevant data is needed for tasks such as statistical reporting, trend spotting and pattern spotting. Data exploration is the process of gathering such relevant data.The ‘Schema-Last’ Approach provides the framework to build targeted index structures to better explore and exploit data.

This chapter will examine how ‘Schema-Last’ models can provide the structures and artefacts necessary to define the index algorithms and field definitions. In addition, this chapter will show how the Apache Solr index can incorporate the ‘Schema-Last’ artefacts.

6.1 'Big Data' Indexing

For 'Big Data' to have any value to an organization it must be able to be exploited. The best way to achieve this goal is to build indexes to allow end-users and automated processes to query the data and have the result returned to them.

The result can also be sorted by relevance, alphabetical, data source or any other criteria. 'Big Data' can be classified structured, semi-structured or unstructured data. Structured data is the easiest to index in that the format is clearly defined and there is no ambiguity in the field definitions, semi-structured is where there could be some ambiguity in both the structure and format of the data. Examples of this type of data include XML, JSON documents and sometimes CSV (comma separated value) files and spreadsheets may also fall into this category. Unstructured data includes text, images, video or any data that can be defined by schema.

For textual documents much has been written (Moens, 2002) and it will be shown how advances in Textual Indexing can discover knowledge that had not been previously known. A number of strategies can be employed to enable a greater opportunity to discover the unknown entities. For example: Figure 6.1 shows that multiple index strategies enhance the search capabilities and the potential to discover the *unknowns*. The index includes and is not restricted to phonetic, n-gram or absolute equality match.

6.2 Elastic Search

An Elastic Search is a form of search that utilises fuzzy index strategies for example phonetic, n-gram or any strategy that yields inexact search results. The largest single unit of data in an elastic search is an index. Indexes are logical and physical partitions of documents within elastic search. Documents and document types are unique per-index. Indexes have no knowledge of data contained in other indexes. From an operational standpoint, many performance and durability related options are set only at the per-index level. From a query perspective, while elastic search supports cross-index searches, in practice it usually makes more organizational sense to design for searches against individual indexes.

Elastic search indexes are most similar to the ‘database’ abstraction in the relational world. An elastic search index is a fully partitioned universe within a single running server instance. Documents and type mappings are scoped per index, making it safe to re-use names and identifiers across indexes. Indexes also have their own settings for cluster replication, *sharding*, custom text analysis, and many other concerns.

An implementation that utilises Elastic Search is **Apache Lucene** which is a popular document indexing implementation see (Apache Lucene, 2015). Indexes in elastic search are not one to one mappings to indexes, they are in fact sharded across a configurable number of indexes, five by default, with one replica per shard. A single machine may have a greater or lesser number of shards for a given index than other machines in the cluster. Elastic search tries to keep the total data across all indexes about equal on all machines, even if that means that certain indexes may be disproportionately represented on a given machine. Each shard has a configurable number of full replicas, which are always stored on unique instances. If the cluster is not big enough to support the specified number of replicas the cluster’s health will be reported as a degraded ‘yellow’ state and being a clustered database, many data guarantees hinge on multiple nodes being available.

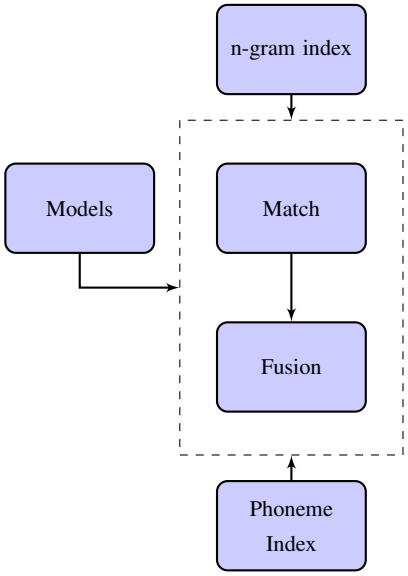


Figure 6.1: Index strategies and the ‘Schema-Last’ Approach

6.3 Index normalization

‘Big Data’ repositories must be in **first normal form** in that:

- each record must a have a unique primary key.

- all occurrences of a record type must contain the fields in the same position for index purposes.

It is not true with big data repositories only that all data must have the same number of fields but all records must have a primary key. The principle is to capture as much information in a single record as possible and it is this record that is indexed. The record may be identified by multiple index entries. Unlike data stored in a **'true'** third normal form, tables are *joined* to develop a similar view of the data as with the 'Big Data' view; the problem with this approach is that it must be known in advance how to best join the tables and ensure that the join operations are optimized. There are benefits to this approach, additional joins can be used to expand the data view and *views* can be used to capture this information.

It may not be possible to construct a comprehensive index with fully normalized data or the associated indexes with a record that may be spread across several tables. For example if the name of the individual is kept in the person table and their address is stored within the location table. The index entry (if the store is indexed within **Solr** repository) may include the name and address points in two distinct rows in different tables.

6.3.1 Phonetic Index Encoding

There are numerous functions to store index entries that represent an index value of a raw data item. These functions are usually intended to find index entries that are *possible* matches of the selected criteria. Phonetic algorithms are useful to provide a *fuzzy* search capability and allow for moderate spelling mistakes providing a comprehensive result set as opposed to an exact search. It is important that indexes are case insensitive and ignore the apostrophe that occurs within names like **O'Brien** and **O'Neale**. SLA through the use of domains can indicate that a soudex or phoentic algorithms can be used to index the data. In addition, tags can capture the type of algorithm that should be used to best query the data.

6.3.1.1 Soundex

Soundex is a phonetic algorithm for indexing names by their sound when pronounced in English. The basic aim is for names with the same pronunciation to be encoded to the same

string so that matching can occur despite minor differences in spelling. Soundex is the most widely known of all phonetic algorithms and is often used (incorrectly) as a synonym for "phonetic algorithm".

Soundex was developed by Robert Russell and Margaret Odell and patented in 1918 and 1922. The Soundex code came to prominence in the 1960s when it was the subject of several articles in the **Communications Journal of the Association for Computing Machinery** (CACM and JACM), especially when described in Donald Knuth's magnum opus, The Art of Computer Programming.

The Soundex code for a name consists of a letter followed by three numbers: the letter is the first letter of the name, and the numbers encode the remaining consonants. Similar sounding consonants share the same number so, for example, the label B, F, P and V are all encoded as 1. Vowels can affect the coding, but are never coded directly unless they appear at the start of the name.

The exact algorithm is as follows:

1. Retain the first letter of the string
2. Remove all occurrences of the following letters, unless it is the first letter: a, e, h, i, o, u, w, y
3. Assign numbers to the remaining letters (after the first) see figure: 6.1
4. If two or more letters with the same number were adjacent in the original name (before step 1), or adjacent except for any intervening h and w, then omit all but the first.
5. Return the first four bytes padded with 0.

6.3.1.2 Meta-phone

Meta-phone is a most popular alternative to Soundex. It was described by Lawrence Philips when he published in the December, 1990 issue of 'Computer Language' magazine. Meta-phone is an advanced version of Soundex which avoids the gross analysis of words. Meta-phone works more exactly than Soundex and is more sensitive for changes in the sequence

Name	Meta-phone	Double Metaphone	Soundex	Cologne Phonetic	Caverphone 1	Caverphone 2	Refined Soundex	Nysiis	Daitch Mokotoff	Phonet 1	Phonet 2
peter	PTR	PTR	P360	127	PT1111	PTA1111111	S360108	PATAR	739000	PETA	PETA
paul	PL	PL	P400	15	P11111	PA11111111	P107	PAL	780000	PAUL	PAUL
mary	MR	MR	M600	67	MR1111	MRA1111111	M8090	MARY	690000	MARI	NARI
mariee	MR	MR	M600	67	MR1111	MRA1111111	M600	MARY	690000	MARIE	NARIE
stephen	STFN	STFN	S315	8236	STFN11	STFN111111	S360108	STAFAN	276000	STEWN	ZTEFN
steven	STFN	STFN	S315	8236	STFN11	STFN111111	S360208	STAFAN	276000	STFN	STFN
billy	BL	PL	B400	15	PL1111	PLA1111111	B1070	BALY	780000	BILI	BILI
bill	BL	PL	B400	15	P11111	PA11111111	B107	BAL	780000	BIL	BIL
joey	J	J	J000	0	Y11111	YA11111111	J40	JY	100000	IOEI	IUEI
joe	J	J	J000	0	Y11111	YA11111111	J40	J	100000	IÖ	IÖ

Table 6.2: Various name encoding algorithms

of the letters and for such combinations as "th". It is based on a method that reduces the words to 16 consonants. The precision is preserved and the variable areas are greatly reduced.

6.4 Lucene and Apache Solr

number	letter
1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M, N
6	R

Table 6.1: Soundex algorithm

Apache Lucene is a free open source information retrieval software library, originally written in Java by Doug Cutting. Lucene is essentially a text based indexing engine.The Apache Solr project was created to provide independent indexing capability. In addition, **Solr** provides powerful full-text search, hit highlighting, faceted search, near real-time indexing, dynamic clustering, database integration, rich document handling, and geospatial search. Solr is highly reliable, scalable and fault tolerant, providing distributed indexing, replication and load-balanced querying, automated failover and recovery and centralized configuration (Smiley and Pugh, 2011).

Letter	Transformation
B	B, unless at the end of word after "m"
C	X (sh), if "-cia-" or "-ch-"
	S if "-ci-", "-ce-", or "-cy-"
	Silent if "-sci-", "-sce-", or "-scy-"
	otherwise, including in "-sch-"
D	J if in "-dge-", "-dgy-", or "-dgi-"
	T otherwise
F	F
G	Silent if in "-gh-" and not at end or before a vowel in "-gn" or "-gned" in "-dge-"
	J if before "i", or "e", or "y" if not double "gg""
	K otherwise
H	Silent if after vowel and no vowel follows or after "-ch-", "-sh-", "-ph-", "-th-", "-gh-"
	H otherwise
J	J
K	Silent if atfer “c”
	K otherwise
L	L
M	M
N	N
P	F if berfore “h”
	P otherwise
Q	K
R	R
S	X (sh) if before “h” or in “-sio-” or “-sia-”
	S otherwise
T	X (sh) if "-tia-" or "-tio-" before "h"
	0 (th) if before "h"
	silent if in "-tch-"
	T otherwise
V	F
W	Silent if not followed by a vowel
	W if followed by a vowel
X	KS
Y	SILENT if not followed by a vowel
	Y if followed by a vowel
Z	S

Figure 6.2: Double Meta-phone algorithm

Solr powers the search and navigation features of many of the world's largest internet sites. Solr is written in Java and runs as a standalone full-text search server within a **servlet** container such as Apache Tomcat or Jetty. Solr uses the Lucene Java search library at its core for full-text indexing and search, and has REST-like HTTP/XML and JSON APIs that make it easy to use from virtually any programming language. Solr's powerful external configuration allows it to be tailored to almost any type of application without Java coding, and it has an extensive plug-in architecture when more advanced customization is required.

Solr has a'plug-in' architecture which comes standard with phonetic based tokenizers which include *soundex* and *double-metaphone* algorithms.

6.4.1 Document Inverse Frequency

Lucene utilizes a technique referred to as document inverse frequency which is the scoring model adopted by Lucene and therefor Solr. At first glance, the Lucene scoring model may seem confusing in that different searches will yield different term scores. Therefore, the Lucene scoring model can appear intimidating to users of Solr.

Raw term frequency as above suffers from a critical problem: all terms are considered equally important when it comes to assessing relevancy on a query. In fact certain terms have little or no discriminating power in determining relevance. For instance, a collection of documents on the auto industry is likely to have the term auto in almost every document. To this end, a mechanism for attenuating the effect of terms that occur too often in the collection to be meaningful for relevance determination. An immediate idea is to scale down the term weights with high collection frequency should be introduced, defined to be the total number of occurrences of a term in the collection. This would be to reduce the **term frequency weight** of a term by a factor that grows with its collection frequency.

Consider a set, D , consisting of n documents:

$$D = \{d^1, d^2 ... d^n\} \tag{6.1}$$

and a set, T , consisting of m unique terms extracted from D:

$$T = \{t^1, t^2 ... t^n\} \tag{6.2}$$

Therefore:

1. The probability of randomly selecting any document from D is:

$$P(d) = \frac{1}{n} \tag{6.3}$$

and the probability that this document mentions a term i from Y is:

$$P(di) = \frac{di}{n} \tag{6.4}$$

2. The probability of randomly selecting any term from T is:

$$P(t) = \frac{1}{m} \tag{6.5}$$

3. The probability that this term is present in a document j from$D$ is

$$P(tj) = \frac{tj}{m} \tag{6.6}$$

4. Inverting these quantities and taking logs we obtain the following weight measures:

$$log(n/di) \tag{6.7}$$

$$log(m/tj) \tag{6.8}$$

Where the above two formulas are referred to as Inverse Document Frequency (**IDF**) and Inverse Term Frequency (**ITF**) respectively. Both formulas can be written down as:

$$log((n-id)/di) \tag{6.9}$$

$$log((m-mj)/tj) \tag{6.10}$$

Both formulas are referred to as: IDF-Probabilistic (**IDFP**) and ITF-Probabilistic (**ITFP**) and have a value of zero when $n = di$ or $m = tj$. Therefore, when $di > \frac{n}{2}$ or $tj > \frac{m}{2}$

Negative weights are notorious for introducing retrieval complexities in **IR** and some search systems that use IDFP and ITFP. A workaround for dealing with these *negative terms* consists in rewriting these as having 0 weight values. To avoid negative terms, special tags associated with the data source might identify stop words so that terms may be ignored by the indexer.

6.4.2 The Solr Schema and Domain Mapping

The Solr schema and the **Schema-Last** *schema* complement each other. The domains identify the relationship between the fields contained within the Solr schema and required indexing strategy. The domains defined in each model must have an associated matching Solr field.

In some cases a domain group is a more appropriate match for a specific Solr field. An example of this would be if the model contained the first-name, middle-name, last-name and the Solr schema only had a full-name. Then domain fields would need to be concatenated to form a new term which becomes the token indexed by Solr. In addition, multiple index strategies can be specified which include: *exact*, *phonetic* and *n-gram* which expand the possibilities of finding the token within the Solr index.

6.4.3 The Solr Schema and Artefact Representation

Apart from the domains, the *schema last* artefacts need to be represented within the **Solr** schema (see Figure 6.4 as an example). The major items that can be represented are the store, bag and set. It is a requirement of **Solr** that each document must have an unique identifier. This can be manufactured by using a **UUID** generator. The store identifier, the bag position, the set position and the model which describes the domain layout if combined can also be used to produce the unique identifier. Ultimately it is the **UUID** that identifies the Solr entry.

$$Solr(id) = S_{id} + b_n + s_n + m_n \tag{6.11}$$

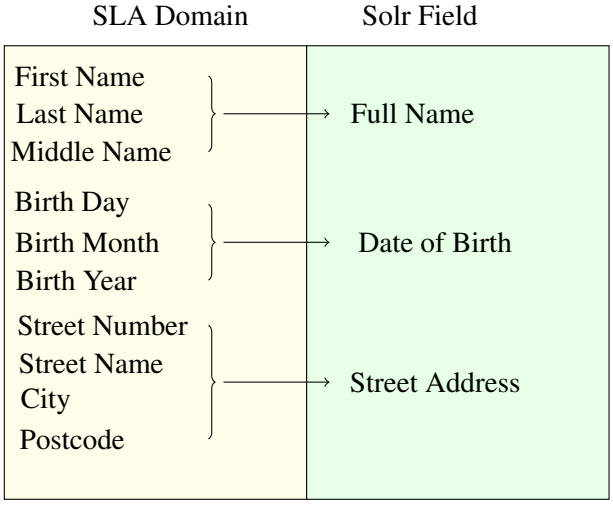


Figure 6.3: SLA domain - Solr field mapping

6.4.4 The Solr Schema and Models

The models contained within a store - S_m- represent the indexing strategies that can be applied to the store to locate individual store *sets*. Each model translates to a Solr document where each domain - *D* - represents a *field-name* within the Solr schema (see Figure 6.5 as an example of a Solr document). The Solr schema provides a *catch-all* where fields that are not explicitly defined within the Solr schema obtain a *default* field definition.

Any cleansing is performed when the index is created and the underlying *raw* data remains untouched. Solr allows for the addition of *tokenizers* which can be used to resolve any form of data ambiguity (see Table 6.3). Tokenizers are fundamental to Solr in that it is the tokenizer that generates the indexable tokens. The domain is used to identify the appropriate tokenizer to generate the indexable tokens and address any ambiguity. The analysts when the be presented with both the index values and raw data in a ranked order determined by the Document Inverse Frequency score.

6.4.5 Search Chaining

The 'Schema-Last' Approach introduces the technique of Search chaining. The introduction of multiple models for a single store allows for a rich query capability where the

```
.
.
.
<field name="identifier" type="string" indexed="true" stored="true" required="true" /
    >
<field name="store" type="string" indexed="true" stored="true" required="true" />
<field name="set" type="string" indexed="true" stored="true" required="true" />
<field name="bag" type="string" indexed="true" stored="true" required="true" />
<field name="name" type="phonetic" indexed="true" stored="true" multiValued="true" />
<field name="given-name" type="phonetic" indexed="true" stored="true" multiValued="
    true" />
<field name="last-name" type="phonetic" indexed="true" stored="true" multiValued="
    true" />
<field name="middle-name" type="phonetic" indexed="true" stored="true" multiValued="
    true" />
<field name="full-name" type="phonetic" indexed="true" stored="true" multiValued="
    true" />
<field name="organisation-name" type="phonetic-organisation" indexed="true" stored="
    true"
                    multiValued="true" />
<field name="birth-date" type="string" indexed="true" stored="true" />
<field name="unit-number" type="text_general" indexed="true" stored="true"
    multiValued="true" />
<field name="street-number" type="text_general" indexed="true" stored="true"
    multiValued="true" />
<field name="street-name" type="text_street" indexed="true" stored="true" multiValued
    ="true" />
<field name="street-complete" type="text_street" indexed="true" stored="true"
    multiValued="true" />
<field name="country" type="text_general" indexed="true" stored="true" multiValued="
    true" />
<field name="address-complete" type="text_address" indexed="true" stored="true"
    multiValued="true" />
<field name="reference-id" type="text_general" indexed="true" stored="true" />
<field name="_version_" type="text_general" indexed="true" stored="true" multiValued=
    "false" />
<dynamicField name="*" type="text_general" indexed="true" stored="true" multiValued="
    true" />
.
.
.
```

Figure 6.4: Snippet of a Solr schema

```
<add>
     <doc>
      <field name="id">11111</field>
      <field name="base">fusion</field>
      <field name="store">af158230-3630-11e4-8c21-0800200c9a66</field>
      <field name="bag">dffcbc10-3630-11e4-8c21-0800200c9a66</field>
      <field name="set">c9570150-3630-11e4-8c21-0800200c9a66</field>
      <field name="full-name">Aldus James Huxley</field>
      <field name="first-name">Aldus</field>
      <field name="middle-name">James</field>
      <field name="last-name">Huxley</field>
      <field name="date-of-birth">20-Jun-1954</field>
   </doc>
   <doc>
      <field name="id">11111</field>
      <field name="base">fusion</field>
      <field name="store">af158230-3630-11e4-8c21-0800200c9a66</field>
      <field name="bag">dffcbc10-3630-11e4-8c21-0800200c9a66</field>
      <field name="set">f9f04240-3630-11e4-8c21-0800200c9a66</field>
      <field name="full-name">Isaac Fredrich Asimov</field>
      <field name="first-name">Isaac</field>
      <field name="middle-name">Fredrich</field>
      <field name="last-name">Asimov</field>
      <field name="date-of-birth">21-Apr-1926</field>
   </doc>
</add>
```

Figure 6.5: Sample Solr document

Tokenizer	Domain	Sample	Tokens	Comment
Date of Birth	Birth-Date	10/01/1962	10-01-1962 01-10-1962 10-01-02 10-01-02	Ambiguity with day or month, could be the first of October or the the 10th of January
Date of Birth	Birth-Date	10/01/02	01-10-02 02-10-01 02-01-10 01-02-10 01-10-02	Ambiguity with day or month an year.

Table 6.3: Tokenizer Ambiguity

analyst can utilise the results from a search result and use this as input to another search (see Figure 6.6). Geospatial search capability enables search chaining through address and locality attributes. Search Chaining is recursive and will eventually return all index entries if not carefully managed.

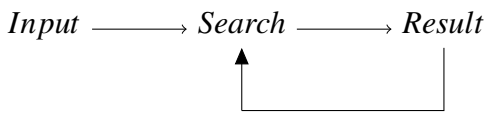


Figure 6.6: Search chaining

Bulk Matching has been identified as a requirement of the agency in order that lists of persons or company names, address or contact details could be used as bulk input to the various search engines. The nature of the search was also taken into consideration in the type of search where a low signal or high signal search (see Sub-section 2.6.2) was required. The results of the search were then returned and used as an **intelligence product** through the dissemination process.

6.4.6 Search Federation

The Informatica Product - IIR - was purchased to support the ACC search capability. IIR is a search engine not unlike Solr but has been tailored to populations. IIR was originally developed in Canberra at the Department of Social Security in the early 80's. Solr has many of the features, however, it lacks the name synonyms that comes supplied with IIR. These synonyms pertain to person names where contractions are often used by persons. For example Steve is a contraction of Steven or Stephen is often used by a person as his nick-name name or the names by which he is known.

Because there are now multiple search engines then the results can be federated into a single view. The search federation takes advantage of 'Schema Last' Approach where results are combined via the **Set Identifier** (see Section 5.4.1). The models define both the Solr and IIR index construction and therefore enable the indexes to conform to a standard *schema* representation.

6.5 Summary

The 'Schema-Last' Approach provides the frame-work to produce a consistent approach to index creation. Solr provides the infrastructure to generate a query of the large data stores as part of the collation phase of the Intelligence Life-Cycle. A fundamental part of knowledge discovery is the ability to explore the large data repositories and not be restricted to a single search engine. This index creation and search capability is fundamental to populate the match store as shown in the Section 5.8. Futhermore, the results from the various search engines are then refined by algorithms.

The models defined by the analyst as part of the 'Schema-Last' Approach allow for a consistent search experience for users of the system. The indexing capability is critical to allow analysts to exploit the data sources. The next chapters will exploit the index strategies defined in this chapter in that the search capabilities form the basis to *fuse* or *reduce* the data for analytical purposes.

Chapter 7

The 'Schema-Last' Approach and Data Matching

Integral to the Intelligence Life-Cycle is the ability to match entities such as a person, organization or even an address from one or more data sources. This process is referred to as **entity resolution**. Entity resolution begins to link the entities within a data source and with entities contained within another separate data source. This process can be automated or manual. In addition, if the entity resolution process was deemed to be incorrect then the process must be reversed. In the case of the ACC, an ontology is used to identify the entity characteristics and potential relationships. Ontologies are excellent in the way entities can be classified and annotated. An ontology can be defined as: "An explicit formal specification of how to represent the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold among them."

This chapter will examine data matching techniques that utilise the 'Schema-Last' Approach. The models reflect the structure of a store and its is input to the various matching strategies. Furthermore, the chapter will introduce various strategies that could be used to combine the various matching results together to formulate a single match score.

7.1 Entity Matching

To identify or classify an entity is important to build an ontological structure and form an organizational *consensus*. Essentially, the analysts within the ACC were asked to identify entities in intelligence documents, data sources. This was achieved through an informal discussion. This was helped by a new strategic initiative to monitor Outlaw Motor Cycle Gangs or OMCG. In addition, a list correlated by the various law enforcement agencies identified persons that pose the greatest threat or risk to the Australian community. Lists of individuals were then given to the ACC to determine whether the list contained the following:

1. People: who are identified by a name and date of birth. A name alone may not be able to identify a person.

2. Organizations: may have multiple names, the trading name and the registered name. Commercial organizations may have a company name or Australian Business Number (ABN) or Australian Company Number (ACN).

3. Groups are similar to organizations and can have a structure and hierarchies.

7.2 Entity Resolution

Entity resolution is essentially the application of various algorithms to determine if two entities in different data sources are identical. The simplest algorithm is to perform a comparison based where the first, last or complete name exactly matches a name in another data source. Analysts have identified that there needs to be some other attribute like a person's date of birth, passport number or other personal identifier. In addition, sometimes the date of birth is incomplete, for example, the Australian Criminal Intelligence Database can record an incomplete data of birth with question marks to indicate a missing day, month or year.

Entity resolution only indicates a possible or potential match. Unless the computed entity resolution can be confirmed a probability or rating is associated with the actual resolution. The 'Schema-Last' approach utilises known identifiers, for example the ABN, to

guarantees that the entity resolution is correct if both entities contain that particular identifier. The *match store* is specifically designed to capture entity resolution and provides the capability to *un-resolve* entities if the resolution is incorrect.

7.3 Data Matching

Fundamental to the intelligence process is the ability to match **'like'** entities contained within records. The 'Schema-Last' Approach provides the framework to identify and model entities, however to get the most out of intelligence data, therefore data matching techniques must be employed to determine potential match candidates. The matching algorithms presented in this chapter have been adapted to take advantage of 'Schema-Last' artefacts and meta-data.

7.3.1 The Data Matching Process

The data matching process relies extensively on field *mapping* utilizing the 'Schema-Last' Approach: *indexing*, candidate *record pair identification* and finally *evaluation.* The first step in the process as described in Chapter 3 aims at defining the data structure, the models and field classifications. The second step is to take the information from the previous step and apply the structure to build indexes. The models from the previous step would have identified attributes that relate to the data source such as a name field, date-of-birth, address and possibly the cultural aspects about the data.

The indexes can then be used to *record pair comparisons* to determine potential matches. If for example the match is certain then there may be no need for further human intervention. However, there will be cases where it is necessary for a human to determine the match status. It is possible that a human can be overwhelmed with potential matches and it is important that most of *record pair identification* is performed by automated processes. Results return from the data matching process are ranked from the most likely match to the least likely. The analyst will still must make a determination which are the potential matches.

There may also be duplicates within the results. It is important to establish the reasons for the duplication. In some cases duplication could be the result of the same record reoccurring within multiple snapshot data sets (See Section 2.7.2).

7.3.2 Data Ambiguity

The heterogeneous nature of data and the nature of variety comes at a price because it is necessary to establish a process to eliminate or at least mitigate data ambiguity. For example, consider two data sources which both contain a field called **name**. Data source one contains the name where a person's surname precedes their first name whilst data source two contains the person's first name followed by their surname. If both data sources are combined this creates ambiguity within the name field. The judicious use of tags allows an analysts to communicate the data matching process that there may ambiguous data present. This can be factored into the scoring model as a part of the overall quality measurement.

7.3.2.1 Name Ambiguity

This is a significant problem within the ACC; some of the data can be ambiguous and lack clarity. For example, often the data can represent a person or an individual. The data contains for example 'David Jones': is that *David Jones* the organization or *David Jones* the person. In addition, in some names it impossible to distinguish the first name from the surname. For instance, a name appears in the data set: *Anthony David* is *Anthony* the first name and *David* the surname or is it equally as plausible that *David* is the first name and *Anthony* the surname (see Table 7.1). Another possible source of confusion in this problem is name contractions. This is where *David* is commonly known as *Dave*, *Stephen* is also known as *Steve* and so on. This type of ambiguity must also be taken into account. A common practice is to build a list or database of known nick-names or name contractions to determine 'like' names.

7.3.2.2 Date Ambiguity

Date is a field that can be problematic and pose special and unique problems. A date on its own can be ambiguous and special attention must be paid to ensure that the date is

Name	Possible Name	Format
Anthony David	Anthony David	First Name/ Surname
	David Anthony	Surname/ First Name

Figure 7.1: Name ambiguity

Date	Possible Date(s)	Format
12/09/10	12 of September 2010	DD/MM/YY
	10 of September 2012	YY/MM/DD
	12 of November 2009	DD/YY/MM
	9 of December 2010	MM/DD/YY
	9 of September 2012	YY/MM/DD

Figure 7.2: Date ambiguity

interpreted correctly. It may be possible that if the data set is sufficiently large that the date can be determined by the other dates within the data source (see Table 7.2). This assumption may still not be valid as date fields may contain errors. Unless there is absolute certainty then ambiguous dates must somehow be resolved.

7.3.2.3 Indexing Strategies and Data Ambiguity

Data pre-processing may prevent some of the ambiguity if the format of the field is known. In some cases this is not possible and even when stated the nature of the data entry may result in ambiguous data. If identified early enough the data may be modified to suit the processing rules that relate to the domain of the field. However, the act of modifying the data's content may have possibly introduced error.

The greatest challenge with 'Big Data' is the creation of indexes to allow the exploitation of the 'Big Data' repository. It is important to choose an index strategy that locates the rows in the 'Big Data' repository can be found quickly and efficiently. There is no requirement that the indexes reside alongside the records within the 'Big Data' repository. The 'Schema Last' Approach dictates that a schema only be used when it is required.

The index strategies can employ one or more fuzzy strategies. However, not all records in a 'Big Data' repository need to be indexed. The big data can include archived records or

records that have limited value. This is a very different approach to relational implementations where every record is indexed. There are many reasons for this as certain records may contain identification information where a *name* sensitive index is more appropriate. Certain data sets may contain name fields where it is unclear which is the first and last name. In that case the index entries need to reflect this and both names must be treated as the first and last names. This also applies to ambiguous dates as multiple date index entries ensure that ambiguous dates are searchable.

7.4 Data Matching Techniques

To perform match comparisons it is important to take into consideration subtle variations to the name spellings. It is no longer sufficient to perform a direct comparison where two names, addresses, or strings are compared character by character, only if there is an exact match are they considered the same. To be able to fuse data it is important to recognize there are a number of algorithms to assist in the match strings. These algorithms are comparing the content of the string with another of a similar type. Some algorithms, particularly the *phonetic* work better if the string contains a **name** which could be a person or organization name or it makes sense to perform a phonetic comparison. In broad terms, the algorithms can be subdivided into three main classifications (Cohen, 2001):

Phonetic: This class of algorithms translates the string (usually a name) to a phonetic code. The phonetic code can then be compared with another and if both are identical there is a good chance that the strings can be considered an approximate match.

Transformation: *Hamming and Levenshtein distance* algorithm identify similar strings by determining what needs to changed for one string to become another. Therefore, the more number of changes required the less likely the strings are to be same.

Gram: Bigram, n-gram and trigram algorithms are designed to identify minor misspellings in names. The gram set of algorithms decompose both strings into either bi-grams (two characters) or tri-grams (three characters). A number of algorithms can be applied to n-grams as shown in Table 7.1.

Level	Pete - $g^{'}$	Peta - $g^{''}$	$g^{'} \cup g^{''}$	$g^{'} \cap g^{''}$	$g^{'} + g^{''}$	Overlap	Jaccard	Dice
Level 1	pe,et,te	pe,et,ta	4	2	6	$\frac{2}{4} = 0.5$	$1 - \frac{2}{4} = 0.5$	$\frac{2\times2}{10} = 0.4$
+Level 2	pt,ee	pt,ea	7	3	10	$\frac{3}{7} = 0.42$	$1 - \frac{7-3}{7} = 0.42$	$\frac{2\times3}{10} = 0.6$
+Level 3	pe	pa	9	3	12	$\frac{3}{9} = 0.33$	$1 - \frac{9-3}{9} = 0.33$	$\frac{2\times3}{12} = 0.5$

Table 7.1: n-gram comparison calculation

To satisfy the above approach a system must be in place to enable collected intelligence to be collated and processed with no interference to the data itself. If the process does pervert the data then the intelligence is unreliable.

7.4.1 *N-gram* Ratio Comparison

N-gram sometimes referred to as *gram* is an algorithm to compare two strings together. The n-gram process is to split the string into short two letter sub-strings or *q-gram* and does this for each letter pair with the string. A further refinement to the algorithm is to progress the algorithm by skipping each subsequent letter until all letters are exhausted.

There are three formulas that can be used to measure the similarity between two words:

1. Overlapping coefficient
$$\frac{g^{'} \cap g^{''}}{g^{'} \cup g^{''}} \tag{7.1}$$
2. Jaccard Distance
$$1 - \left(\frac{(g^{'} \cup g^{''}) - (g^{'} \cap g^{''})}{(g^{'} \cup g^{''})} \right) \tag{7.2}$$
3. Dice Coefficient
$$\frac{2 \times (g^{'} \cap g^{''})}{(g^{'} + g^{''})} \tag{7.3}$$

All three methods, as the score approaches one, indicates a stronger string similarity.

7.4.2 *Monge-Elkan* String Comparison

Monge and Elkan (Monge, 1996) proposed a simple but effective method to measure the similarity between two text strings that contain several tokens, using the internal similarity function $sim(a,b)$ to measure the similarity between two individual tokens a and b . Given two texts A , B , with $|A|$ and $|B|$ being their respective number of tokens, and an external inter-token similarity measure $sim^{'}$.

$$sim_{MongeElkan}(A,B) = \frac{1}{|A|}\sum_{i=1}^{|A|} max\left\{sim^{'}\left(a_i,b_j\right)\right\}_{j=1}^{|B|}$$

7.4.3 *Levenshtein* String Comparision

The edit distance was originally proposed by Levenshtein. It is equal to the minimum number of editing operations required to transform one sequence into the other. The three basic editing operations are insertion, deletion, and substitution. Several modifications to the original edit distance have been proposed, varying cost schemes and adding more edit operations such as transpositions, opening, and extending gaps. The solution for computing the edit distance is a dynamic programming algorithm that stores in a matrix the counts of edit operations for all possible prefixes of both strings. This algorithm computes the edit distance between two strings a and b of length |a| and |b| with a time complexity of $O(|a| \times |b|)$ and space complexity of $O(min(|a||b))$. The edit distance measure can be normalized in the range $[0,1]$ by dividing the total number of operations by the number of characters in the longer string. Once normalized, the edit distance can be converted to similarity by subtracting the distance value to the number 1.

7.5 Stochastic Considerations

Almost any score should consider stochastic attributes within the score formulation. Stochastic attributes would include the quality of the data, the time between the time difference when both data stores were collected which in turn affects the overall score. The 'Schema-Last' Approach utilizes meta-data to contain those values and allows for those values to be used as input as part of the match determination.

7.5.1 Data Quality

The stores meta-data may have indicators that should be applied to the score calculation. For example, if a store contains data that may be considered of poor quality and the compared store has data that is of good quality then this can also be reflected in the overall score. It must be noted that all quality indicators must both be scaled on the same axis, for example 0 to 10.

The quality function is:

- a is the first quality score
- b is the second quality score
- Qd is the quality function to measure data

$$Qd(a,b) = \sqrt{(a-b)^2} \tag{7.4}$$

An example of quality function in use (both: a and b; maximum value is 10 where 0 means no quality and 10 means high quality)

- $a = 5$ out of 10
- $b = 4$ is the second quality score
- Q is the quality function

$$Qd(a,b) = \sqrt{(5-4)^2} = 1 \tag{7.5}$$

7.5.2 Time of Collection

As data ages the value of the data diminishes over time. The principle of ROT (Redundant Obsolete Trivial) can apply to intelligence data. Therefore if two data sets were collected at different time periods then this can be used to calculate the score. This novel algorithm devised as part of the 'Schema-Last' Approach is an enhancement to current data matching algorithms where the time of the collection is recorded within the meta-data associated with the *store*. The Equation 7.6 returns an estimate of the quality and the larger the number, the

further apart the collection periods were taken. At the ACC the time difference is measured in days and the accuracy of a day is determined to be accurate enough.

$$Qt(a,b) = log(1+\sqrt{(a-b)^2}) \tag{7.6}$$

- $a = 12/Jun/2014$
- $b = 1/Jun/2014$
- 376 is the number of days between the above two dates

$$Qd(a,b) = log(1+\sqrt{(-376)^2}) = 2.57 \tag{7.7}$$

The higher the number, the less likely the relevance of the match. Usually any score greater than **2.50** should be considered for rejection.

7.5.2.1 Time of Observation

A further extension to the above model is if a set has an associated temporal dimension within the set. Instead of using the *collection date* associated with the store the time associated with the set can be used in conjunction or instead. The same formula can be used but instead of using the *collection date* this date can be used. The name of the column of the observation time can be identified by a special domain.

7.5.3 Intelligence Rating

A further stochastic criteria to be considered is how reliable is the data in the first place. If the data is from an unreliable data source then this can also be factored into the final score. Unlike quality where the difference is important, reliability is a multiplication of both indicators. The Admiralty System or NATO System (US-Army, 2006) is a method for evaluating collected items of intelligence and this provides a scheme to measure data reliability (see Figure 7.3). It can be used to determine the quality of match in that it can be used to contain an intelligence rating value. The intelligence rating would be applied to

Symbol	Value
A	Completely reliable
B	Usually reliable
C	Fairly Reliable
D	Not usually reliable
E	Unreliable
F	Reliability cannot be judged

Symbol	Value
1	Confirmed by other sources
2	Probably true
3	Possibly true
4	Doubtful
5	Improbable
6	Truth cannot be judged

Figure 7.3: Admiralty system

the entire **store** and represented as meta-data. If both compared sources are truthful, then the intelligence rating needs to be applied to the final match result.

- a is the first quality score (using only reliability)
- b is the second quality score (using only reliability)
- h is the lowest qualitative score (6) for both reliability and accuracy
- Qr is the reliability function to measure data
- Qa is the accuracy of the data

$$Qr(a,b) = 1 - \frac{a \times b}{h^2} \tag{7.8}$$

An example of reliability function in use (both :a and b maximum value is 6 where 1 means highly reliable and 6 is not reliable)

- $a = 3$ out of 6 (a = 1..6)
- $b = 4$ out of 6 (b = 1..6)
- Q is the quality function

$$Qr(a,b) = 1 - \frac{3 \times 4}{6^2} = 0.66 \tag{7.9}$$

Name1	Name2	Hamming Distance	Levenshtein Distance	Sgram	Trigram	Soundex similarity	Monge Elkan
smyth	smith	1	1	0.429	0.250	4	0.68
peter	peta	2	2	0.421	0.286	3	0.9
stephen	steven	4	2	0.345	0.300	4	0.76
peter	paul	4	4	0.038	0.0	2	0.45
billy	bill	1	1	0.600	0.500	4	1
joey	joey	0	0	1	1	4	1

Table 7.2: Various name matching test results

7.5.4 Rarity of Name

If an organization is fortunate enough to have a complete set of names then the *rarity* of the name can be factored into the score. For example, John Smith is a common name in Anglo centric communities, however Patrick Dempsey is not so common and the rarity of the name can be factored into the score. Therefore, if the name in two independent sets from two stores contain the name Patrick Dempsey it is probable that they are the same name. If however, the name is John Smith, by virtue that the name is very common, there is a high probability that these name are not a match. The rarity of name can be stored as meta-data with the associated *set*.

7.6 Multiple-criteria decision analysis

Individual matching algorithms have been discussed in the previous section, however multiple algorithms will lead to a more accurate match result. Matching two names and associated addresses can be interpreted in different ways. The match could correspond to choosing the *best* alternative from a set of available alternatives. Another interpretation of 'solving' could be choosing a small set of good alternatives, or grouping alternatives into different preference sets. An extreme interpretation could be to find all *efficient* or *non-dominated* alternatives. For example in Table 7.2 each algorithm can be used to compare two names and the best score is chosen to indicate the match result.

7.6.1 Decision Trees

Decision Trees can be defined as (Tom, 1997):

1. Define the problem.
2. Identify decision alternatives.
3. List the possible outcomes of each decision alternative.
4. Represent the sequence of events using chance nodes (events determined by chance) or decision nodes (events determined by a decision).

Decision Trees provide the structure to determine a single match score by selecting scores (see Figure 7.4). Weights assoicated with the score can be used to select the appropriate path within the decision tree. An extension to this model is the Markov Chain where the determination of alternate paths can be explored to formulate the match result.

7.6.2 Markov Chains

The Markov model treats a sequence of events over time as a series of state transitions. The model assumes that there is only a finite number of Markov states. Events are modelled as transitions between states or transitions within a state, meaning that after one period, a single Markov cycle.

The progressive state would either strengthen or weaken the hypothesis that the entities are the same along the Markov chain as represented in Figure 7.5. Unlike the decision tree approach it is possible to revisit states and accumulate each score until the end state has been reached. To manage **Transitive Closure** there may be a point when there is no need to continue along the finite state path if the accumulated score exceeds a certain threshold. This has the advantage that the entire path is not required to be traversed to obtain an acceptable result without the necessity to traverse the entire graph. Both the PROLOG and LISP languages provide the capability to terminate via the ‘Cut’ construct where the algorithm has determined that there are no more valid alternate paths.

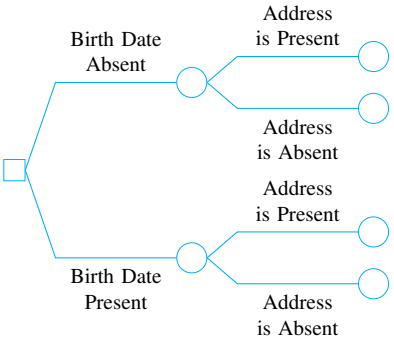


Figure 7.4: A Simple Decision Tree

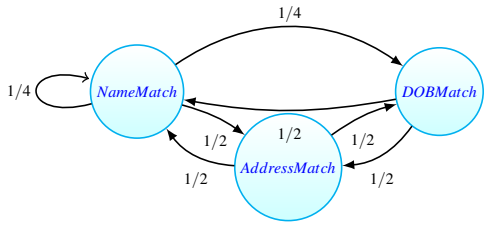


Figure 7.5: Markov chain example

7.6.3 The Weighted Sum Model

The weighted sum model (WSM) is a model that can accumulate one or more scores and weight each score individually and return a single result where 1 is a perfect score (Triantaphyllou, 2002):

The model can be defined as:

- s_n is the score out of a $h_n$where n is the specific test
- m is the number of tests performed
- w_n is the weight associated with the test

$$P(S,H) = \sum_{j=1}^{n} \left(\frac{s_j \times w_j}{h_j \times w} \right) \tag{7.10}$$

Example of a test with two match indicators :

- $s_1 = 80$ is a phonetic name score out of a possible 100 ($h_1 = 100$)
- $s_1 = 70$ is a geospatial score of 70 out of a possible 100 ($h_1 = 100$)
- $w_1 = 8$ the weight of a name
- $w_2 = 2$ weight for geospatial match which is 4 times less significant than a phonetic name match
- $m = 2$ the number of match score results

$$\frac{(80*8)+(70*2)}{(100*8)+(100*2)} = 0.78 \tag{7.11}$$

7.6.4 The Weighted Product Model

The weighted product model (WPM) is similar to the weighted sum model except all tests must yield a $result > 0$ otherwise there is no result at all (Triantaphyllou, 2002).

The weighted product model can be defined as:

- s_n is the score out of a standardized ($h_n = 100$)
- m is the number of tests performed
- w_n is the weight associated with the test

$$P(S,H) = \prod_{j=1}^{n} \left(\frac{s_j}{h_j}\right)^{w_j} \tag{7.12}$$

Example of a test with two match indicators :

- $s_1 = 80$ is a phonetic name score out of a possible 100 ($h_1 = 100$)
- $s_2 = 70$ is a geospatial score of 70 out of a possible 100 ($h_2 = 100$)
- $w_1 = 0.8$ the weight of a name
- $w_2 = 0.2$ weight for geospatial match which is 4 times less significant than a phonetic name match
- $m = 2$ the number of match score results

$$(\frac{80}{100})^{0.8} \times (\frac{70}{100})^{0.2} = 0.7 \tag{7.13}$$

7.6.5 Stochastic Weighted Average Score

If a quality measure is introduced it could be kept within a user-specified tag within the store (see 5.4.13). An example would be a data quality dimension which would be a value from 0 to 10 for example where 10 is high quality and zero is no quality. Other examples would be the timeliness of the data as the data ages the value may diminish.

- s_n is the score out of a h_n
- m is the number of tests performed
- w_n is the weight associated with the test
- Sq_n is a quality measurement assigned to data source - a meta-data value associated with the store.
- v_n is maximum quality measurement
- n is the number of tests performed

$$P(S,H,Q) = \sum_{j=1}^{n} \left(\frac{s_j \times w_j}{h_j \times w} \right) \times \sum_{k=1}^{n} \left(\frac{Sq_k}{v_k} \right) \tag{7.14}$$

Example of a test with two match indicators :

- $s_1 = 80$ is a phonetic name score out of a possible 100 ($h_1 = 100$)
- $s_2 = 70$ is a geospatial score of 70 out of a possible 100 ($h_2 = 100$)
- $w_1 = 8$ the weight of a name
- $w_2 = 2$ weight for geospatial match which is 4 times less significant than a phonetic name match
- $m = 2$ the number of match score results
- $q = 8$ the quality measurement out of a possible 10.

$$\frac{(80*8)+(70*2)}{(100*8)+(100*2)} \times \frac{8}{10} = 0.64 \tag{7.15}$$

7.6.6 The ACC 'Aries' Score

The ACC took a unique approach to determine a score from utilizing multiple criteria score and called this the *'Aries score*'. The score is an accumulation of all the scores into a single number where each position within that number represents the score for a specific test. Each test must be scaled from zero to a upper limit number (usually 9) where this scale represents the test score. An example of this approach is as follows:

- $s_1 = 8$ is a phonetic name score out of a possible 9 ($h_1 = 9$) possible scores are $\{0,1,2,3,4,5,6,7,8,9\}$
- $s_2 = 7$ is a geospatial score of 7 out of a possible 9 ($h_2 = 9$) possible scores are $\{0,1,2,3,4,5,6,7,8,9\}$
- $s_3 = 5$ is a date-of-birth score of 5 out of a possible 9 ($h_3 = 9$) possible scores are $\{0,1,2,3,4,5,6,7,8,9\}$
- $s_4 = 6$ is a *monge-elkan* score of 6 out of a possible 9[1] ($h_4 = 9$) possible scores are $\{0,1,2,3,4,5,6,7,8,9\}$
- $m = 4$ the number of match score results

$$'Aries - Score' = [8756] \tag{7.16}$$

Certain score combinations werezb considered to indicate a closer match and unless the score yielded results where name score was greater than 8 then this was deemed *'not a match*'. For example: the Analysts at the ACC considered that scores greater than 7 was a close match for example:

The Name *Thomas Jones* matched with *Tom Jones* both born on the 15 February 1971 then the calculated Aries score would be:

- $s_1 = 7$ is a phonetic name score out of a possible 9 ($h_1 = 9$) for the match Tom to Thomas
- $s_2 = 0$ is a geospatial score out of a possible 9 ($h_2 = 9$)
- $s_3 = 9$ is a date-of-birth out of a possible 9 ($h_3 = 9$)
- $s_4 = 8$ is a *monge-elkan* scoreout of a possible 9 ($h_4 = 9$)

$$'Aries - Score' = [7098] \tag{7.17}$$

The analysts used as a rule of thumb any *'Aries score'* any score with 3 componets greater or equal to '7' a good match.

[1]The *monge-elkan* score has been scaled

7.7 Data Matching Techniques in Practice

Whatever matching technique is chosen to match one set of records with another then the number of records can pose a significant computational problem. The approach taken at the ACC was to utilise the elastic indexes as much as possible and utilise the data exploration approach as described in Chapter 6. Furthermore, the data exploration will return a set of results which when further refined can be made subject to the various matching techniques as described in this chapter. These matching techniques could act as an additional *filter* to a set of results returned from a search query.

The Aries score was discarded because of confusion amongst the data analysts; it was found to be difficult to compare one score with another. This was replaced by the **Weighted Sum Score** as described in this chapter and was found to be an effective way to share the match results with other analysts. The ACC intelligence analysts have also requested that any score that is an amalgamation of multiple scores that each individual score be presented within the report.

7.8 Summary

This chapter discussed various algorithms and how data matching can be applied to sets. The next chapter will discuss how data can be fused into a single view by utilizing the matching algorithms described in this chapter and thereby allowing the analyst to have a single and consistent view of the data. The next chapter will take the algorithms presented in this chapter and apply these algorithms to *fuse* and *reduce* data sets.

Chapter 8

The 'Schema-Last Approach' and Data Fusion

Data fusion is the adoption of a single logical view across all data sources. This can be expressed as the sum of all the assessable data sources. The assets include all the models, meta-data and domain specification and utilize the data exploitation and matching process described in the previous chapters. Data fusion can only be achieved if the index engines are built and tuned (see Chapter 6).

The data source does not have to be local to the organization but contained within a cloud environment or a public network. The section will demonstrate how the 'Schema-Last' Approach provides the platform to enable a consistent approach to data fusion. Data fusion is a multidisciplinary area that involves several fields, and it is difficult to establish a clear and strict classification.

Apache Hadoop provides the means to produce the fused data. SLA can assist Apache Hadoop to provide the *maps* as the input to the **'map phase'** in regard to the map reduction process.

This chapter will demonstrate how both data fusion and data reduction can utilize the 'Schema-Last' Approach.

Intelligence processing involves both information processing and information fusion. Gathering intelligence is generally provided at a *high* level. In the world of intelligence the

data is often presented in the form of an intelligence report which is already at a high level of abstraction – either free form text or in a predefined report.

- Collation – associated intelligence reports are correlated and brought together. Some combination or compression may occur at this stage. Collated reports, however, may simply be packaged together ready for fusion at the next stage.
- Evaluation – the collated intelligence reports are fused and analyzed. Historically, highly skilled human intelligence analysts have undertaken this process. The analysis may identify significant gaps in the intelligence collection. In this case, the analyst may be able to task a collection asset directly. More usually, however, this requirement is included in the disseminated information.
- Dissemination – the fused intelligence is distributed to the users (usually military commanders) who use the information to make decisions regarding their own actions and the required deployment of further collection assets.

This chapter is organized in the following manner:

- examine existing data fusion models
- where data fusion can benefit from the 'Schema-Last' Approach
- the role 'Schema-Last' Approach has with data munging
- the role 'Schema-Last' Approach has with data reduction

8.1 Data Fusion and Data Reduction

There are well defined data fusion classifications. These classifications will be discussed in turn and will show how each model can benefit by the 'Schema Last' Approach. Data fusion is combining data sources into a consolidated view whilst data reduction is the opposite where the data is summarized for further data analysis.

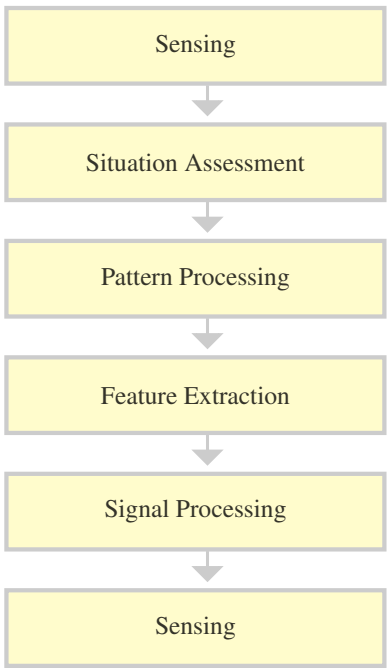


Figure 8.1: The Waterfall model

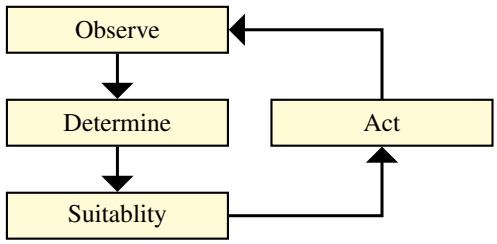


Figure 8.2: The Boyd loop

8.1.1 The Waterfall Model

The Waterfall Model (Bedworth and O'Brien, 2000) places its main emphasis equivalence on approximates as its name suggests is **not** an iterative process. Even though not widely used the technique forms the basis for other models. For example the JDL and Dasarathy models.

8.1.2 Boyd Loop

The Boyd Loop has been widely used for data fusion even though the process is based on a military strategy identified by John Boyd in the early sixties (Boyd, 1987). The Boyd (or OODA) Loop is an improvement on the Waterfall Model because the process is iterative. The inherent feed-back loop enables the process of data fusion to continuously improve.

8.1.3 The JDL Model

In the JDL model, proposed by the US Joint Directors of Laboratories Data Fusion Sub-Group in 1985 (Bedworth and O'Brien, 2000) and recently updated, the processing is divided into five levels as shown in Figure 8.3.

Level 0 – sub-object data assessment, is associated with *pre-detection* activities such as pixel or signal processing, spatial or temporal registration.

Level 1 – object refinement, is concerned with the estimation and prediction of continuous (location or kinematic) or discrete (behavior or identity) states of objects.

Level 2 – situation refinement, introduces context by examining the relations among entities such as force structure and communication roles. By aggregating objects into meta-objects an interpretation may be placed on the situation.

Level 3 – implication refinement, delineates sets of possible courses of action and the effect they would have in the current situation. This level also introduces the concept that the data fusion system may be operating in an adversarial domain.

Level 4 – process refinement, is an element of resource management is used to close the loop by re-tasking resources (e.g. sensors, communications and processing) in order to support the objectives of the mission.

This model has been widely used by the US data fusion community and can now be regarded as the *de facto* standard for defense data fusion systems, at least in the US. Partly because of its popularity it is applied in a variety of ways and not always used appropriately. The JDL model was never intended to prescribe a strict ordering on the data fusion levels. This was indicated diagrammatically by the use of an information bus rather than a flow structure. Nevertheless, data fusion system designers have consistently assumed this ordering. Clearly there is a need for users to have an ordering whilst the authors of the JDL model rightly defend the need for a model which admits systems with different hierarchies at different levels.

8.1.4 Durrant-Whyte Classification

The employed methods and techniques can be divided according to the following criteria: attending to the relations between the input data sources, as proposed by Durrant-Whyte. These relations can be defined as:

- complementary;
- redundant, or cooperative data;
- according to the input/output data types and their nature, as proposed by Dasarathy;
- following an abstraction level of the employed data:
- raw measurement;
- signals;
- characteristics or decisions;
- based on the different data fusion levels defined by the JDL;
- Depending on the architecture type:
  - centralized,
  - decentralized, or
  - distributed.

8.1.5 Dasarathy's Classification

One of the most well-known data fusion classification systems was provided by Dasarathy (Dasarathy, 1997) and is composed of the following five categories (see Figure 8.4):

1. Data in-data out (DAI-DAO): this type is the most basic or elementary data fusion method that is considered in classification. This type of data fusion process inputs and outputs raw data; the results are typically more reliable or accurate. Data fusion at this level is conducted immediately after the data are gathered from the sensors.

The algorithms employed at this level are based on signal and image processing algorithms data in-feature out.

2. (DAI-FEO): data is combined to derive some form of a feature of the object in the environment or a descriptor of the phenomenon under observation. Fusion in this mode, depending on one's view point, input-fusion of data or output-fusion resulting in features, has been looked upon either as data fusion or feature fusion. Feature fusion has also been variously referred to as symbolic fusion, information fusion, fusion at the intermediate level, and so on. The manner in which depth perception is achieved in humans, by combining the visual information acquired from the two eyes, can be looked upon as a classical paradigm of this feature or information fusion. Indeed techniques based on this kind of paradigm have been investigated for machine perception of depth in robotic systems. The traditional approach to the computation of object surface temperatures using the intensities from two infrared (IR) bands of a multispectral scanner is another good example of data in-feature out mode of fusion processing. In some cases, this fusion mode may be the first step with the previous mode being totally absent.

3. Feature in–feature out (FEI-FEO): at this level, both the input and output of the data fusion process are features. Thus, the data fusion process addresses a set of features to improve, refine or obtain new features. This process is also known as feature fusion, symbolic fusion, information fusion or intermediate-level fusion;

4. Feature in-decision out (FEI-DEO): this level obtains a set of features as input and provides a set of decisions as output. Most of the classification systems that perform a decision based on a sensor's inputs fall into this category of classification.

5. Decision in-decision out (DEI-DEO): This type of classification is also known as decision fusion. It fuses input decisions to obtain better or new decisions.

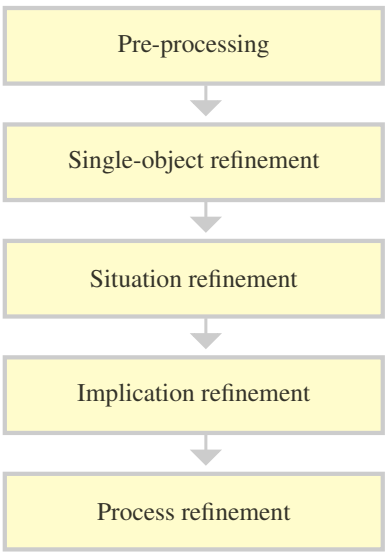


Figure 8.3: JDL model

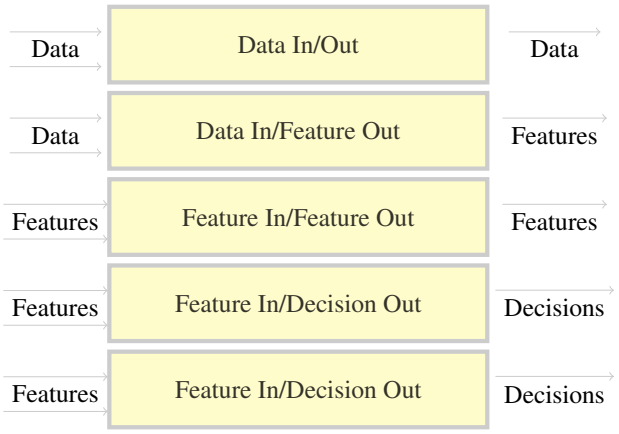


Figure 8.4: Dasarathy's classification

8.1.6 Thomopoulos Classification

Thomopoulos (Thomopoulos, 1989) posed an architecture for data fusion consisting of three modules, each integrating data at different levels or modules to integrate the data, namely:

- **Signal level fusion**, where data correlation takes place through learning due to the lack of a mathematical model describing the phenomenon being measured.
- **Evidence level fusion**, where data is combined at different levels of inference based on a statistical model and the assessment required by the user (e.g. decision making or hypothesis testing).
- **Dynamics level fusion**, where the fusion of data is done with the aid of an existing mathematical model.

Thomopoulos describes the fusion process as:

> "A fusion system can be a very complicated system. It is composed of sources of information, of means of acquisition of this information, of communications

> for the exchange of information, of intelligence to process the information and to issue information of higher content. The issues involved may be separated in topological and processing issues. Despite the interconnection between both issues in an integrated fusion system design, they can be decoupled from each other in order to facilitate the development of a systematic methodology of analysis and synthesis of a fusion system (Thomopoulos, 1991)."

8.1.7 Fusion Models and Intelligence Life-Cycle

Dasarathy, JLA Thomopoulos classifications align closely with the Intelligence Life-Cycle. The collection and collation phase must be cognizant of the fusion approach and apply the fusion classification accordingly. The Thomopoulos classification demands a hypothesis or a rationale to build the fused data view. The importance of construction of a fused view of the data cannot be under-estimated, however tools are required or built for analysts to exploit the fused data. Fused data is prepared as part of the collation or processing phase of the Intelligence Life-Cycle.

Data reduction technologies, for example **map/reduce** allow the analysts to develop algorithms which extract data to be used for statistical analysis or reporting purposes. Data reduction can utilize the matching algorithms from the previous section. It may be necessary to employ and develop data munging techniques to effectively 'fuse' the data.

8.2 Data Munging and Data Fusion

Data munging or data wrangling is loosely the process of manually converting or mapping data from one 'raw' form into another format that allows for more convenient consumption of the data with the help of semi-automated tools (Raymond, 1996). This may include further munging, data visualization, data aggregation, training a statistical model, as well as many other potential uses. Data munging as a process typically follows a set of general steps which begin with extracting the data in a raw form from the data source, 'munging' the raw data using algorithms (for example sorting) or parsing the data into predefined data structures, and finally depositing the resulting content into a data sink for storage and

future use. Given the rapid growth of the internet such techniques will become increasingly important in the organization of the growing amounts of data available.

8.3 Data Fusion Quality

'Big Data' is an asset and like any asset can lose value over time. The life cycle of data can be described as a never ending-cycle whereby data enters the system and ultimately is stored and becomes part of the'Big Data' repository. Finally data may be retired once the value of data diminishes. Another characteristic of data is that data can be copied, manipulated and analyzed. Data annotation is also important to enhance the data's usefulness. For example, the data source, the update frequency and the data's credibility are all important.

8.3.1 Data Collection and Data Fusion

The cost to input a collection into a database can be substantial (Armstrong, 1992) but is only a fraction of the cost of checking and correcting the data at a later date. It is better to prevent errors than to cure them later (Redman, 2001) and it is by far the cheaper option. Making corrections retrospectively can also mean that the incorrect data may have already been used in a number of analyses before being corrected, causing downstream costs of decisions made on poor data, or of re-conducting the analyses (Chapman, 2005).

8.3.1.1 Prevention is better than cure

The primary responsibility for the management of data quality rests with the collector of the data. It is their responsibility to make sure that:

- label information is correct,
- label information is accurately recorded and documented,
- allocate the domain correctly to each data item,
- locality information is as accurate as possible, and both accuracy and precision are documented,

- collection methodologies are fully documented and
- label or field notes are clear and unambiguous.

The importance of data collection cannot be underestimated.

8.3.2 Incomplete or Missing Data

Incomplete data values are a considerable problem in any data fusion process and will require some kind of action either programmatic or human intervention. The most common approach to missing data is to simply omit those cases and to run analyses on what remains. Thus if five subjects in a group one do not show up to be tested, that group is five observations short. Or if five individuals have missing addresses on one or more variables, those rows are simply omitted from the analysis or index. This approach is called *list-wise* deletion.

The imputation of missing values can be at best problematic but can be based on previous data within the data source. At least some of the data's attributes may be imputed. For example if the *sets* are modified to include imputed values then this has perverted the original data.

The *null* value is another solution to the incomplete data in that null values indicate that the value is unknown. Meta-data associated with the data source through the use of tags can best be used to denote missing or imputed values. In addition, the quality of the data source would deteriorate and this could also be reflected in the meta-data.

8.4 Data Reduction

The sheer size and complexity of the data set sometimes makes the analysis daunting, but a large data set may also yield richer and more useful information. Data reduction strategies combine qualitative and quantitative analysis techniques for the analysis of large, qualitative data sets. In broad terms data reduction is the application of algorithms to reduce a large data set to a summarized form. The benefits of the data reduction techniques proposed increase as the data sets themselves grow in size and complexity. The application of

data mining techniques or general map reduction strategies (Namey et al., 2007), provides the capability to develop a deeper understanding of the data, relationships, or any associations without de-emphasizing the importance of the context and richness of the original data. Data reductions bring a perspective and focus to the multiple interpretive lenses that a group of researchers brings to team-based analysis.

Data reduction provides the platform for a deep understanding of the data without affecting the underlying raw data set.

8.4.1 Map Reduction

Map-Reduce is the heart of Hadoop®. It is this programming paradigm that allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster. The map job, takes a set of data and converts it into another set of data where individual elements are broken down into tuples (key/value pairs). The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As the sequence of the name Map-Reduce implies, the reduce job is always performed after the map job. The result may not always be a reduction, but equally be a data source fusion. SLA complements the Map-Reduction because it provides the models and structures to identify the label, domains and structure to the reduction phase (see Figure 8.6).

Fusing or *munging* two or more data sets can be achieved utilising the store models and domains described in Section 5.4.4.9. If this is true for any two data sources then these two data sources can be *fused.* The fused result may or may not include unassigned cells (cells without an assigned domain).

Languages as Apache Pig (see Figure 8.5 a very simple code sample) and Apache Hive are designed to simplify the Hadoop platform and provide a platform to munge or fuse data. Both products are open source products and Apache Pig was developed by Facebook as a map-reduction platform and was later made available to the open-source community.

8.4.2 Data Reduction and Hadoop

As ‘Big Data’ and Apache Hadoop go hand-in-hand. Apache Hadoop is an open source software project that enables the distributed processing of large data sets across clusters of

```
register "eland-pig-1.10-SNAPSHOT.jar"
register "file:/usr/lib/pig/piggybank.jar"
register "peppapig-1.0-SNAPSHOT.jar"
dump_set = LOAD "fusion://localhost:9160/identifier" USING
  org.eland.pig.storage.ElandReader("id");

extract = FOREACH dump_set $0, $1, $2;

STORE extract INTO 'temp.pig' using PigStorage("~");
```

Figure 8.5: Apache Pig example

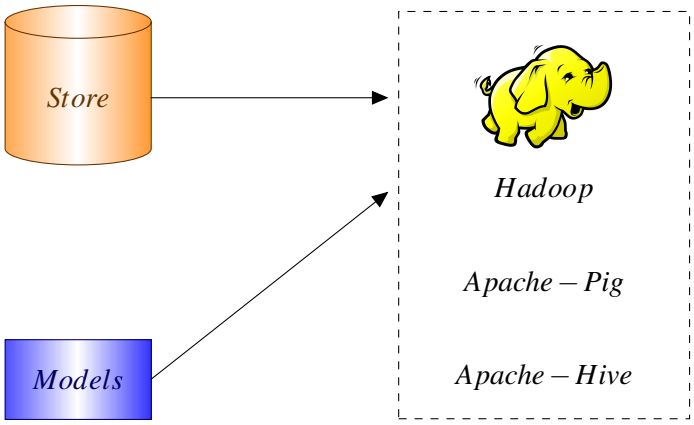


Figure 8.6: Hadoop and the 'Schema-Last' Approach

commodity servers. It is designed to scale up from a single server to thousands of machines, with a very high degree of fault tolerance. Rather than relying on high-end hardware, the resiliency of these clusters comes from the software's ability to detect and handle failures at the application layer Hadoop is not used for data storage, but is best served for data processing or fusing. Apache Pig provides an interface for programmers or vendors to write their own extraction and storage routines. This provides a user-friendly development interface to the *set stores* to provide a mechanism to bulk extract and process sets. In addition, the store models and meta-data can be used as input to the map reduction as part of the Apache Hadoop processing.

8.5 Summary

The ‘Schema-Last’ Approach is an essential part of data fusion. The data can be fused and presents a consolidated view to any analytical processing. The new tools Apache Hadoop, Apache Flume just to name a few can all be used to fuse data sets into a consolidated view. Data fusion and reduction are essential for data analysis whether the fusion is semantically linking **lik**e entities or formulating the ‘single source of truth’. Informatica has developed products that address data fusion and state:

> “Fusing intelligence-driven security data from multiple sources and processing it with big data analytics has the potential to solve this problem. But first, analysts need to identify relevant information and recognize its potential relationships to other data points. The sheer volume of data makes it very difficult to detect, analyze, and act on threats from any single source. ‘Big data’ analytics rises to this challenge and presents an opportunity to garner intelligence collectively from all of the pieces.(Informatica, 2014)”

The next chapter will present a case study which demonstrates the ‘Schema-Last’ Approach and how this approach could present the data as a consolidated view.

Chapter 9

The 'Schema-Last' Approach: A Case Study

The ACC determined that the existing process to ingest data as part of the collation phase of the Intelligence Life-Cycle was unable to cope with the influx of existing data sources (see Figure 9.1)[1]. This meant a novel solution was required to cope with collation ingestion demands. The concept Fusion Data Holding (FDH) was created as a repository to hold the raw data for further analysis. It soon become obvious to ACC management that cleansing source data was contributing to processing delays.

The intention with the case study was to deliberately **not** cleanse any data and leave the data in its *raw* state. Any data cleansing would be done by the various Solr tokenizers or the indexers and each index record would contain a reference back to the original record. The ACC management accepted the new proposed design, however there was much debate over product and technology selection within ACC's analysts. The 'Schema-Last' Approach was a novel strategy and part of the case study was to select the appropriate technology and product set. If no product could support the approach then a **bespoke** development of a product would be required. However, the intention is to minimize any **bespoke** development and that the case study would form the basis for a production system that would support the Fusion Data Holding.

[1]The figure summarizes the total volume of data received (in orange) against the volume of data processed (in purple)

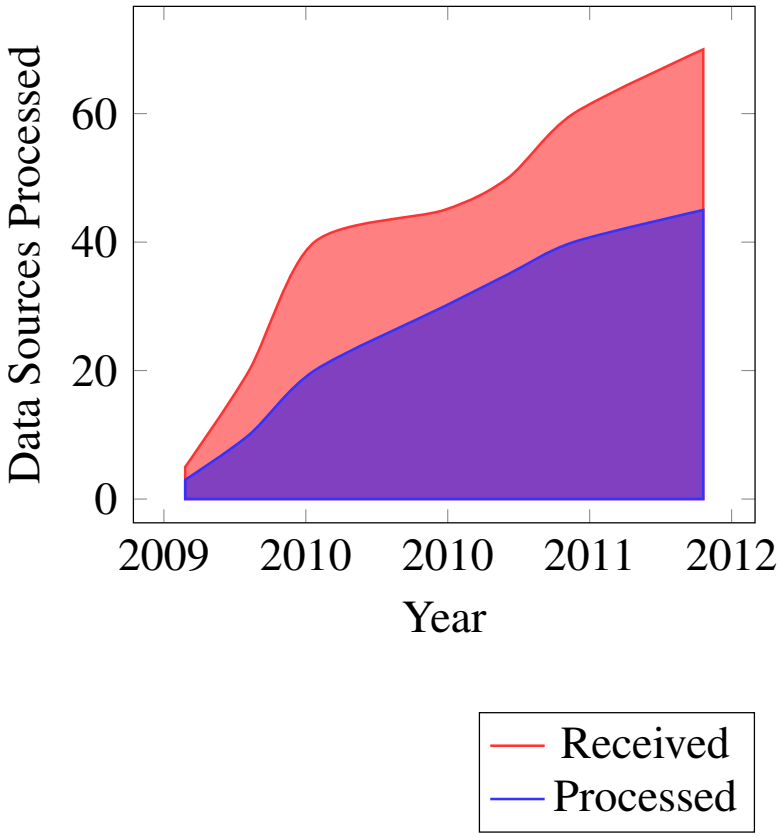


Figure 9.1: Backlog demand: 2009 - 2011

9.1 Fusion Data Holding

The Fusion Data Holding or **FDH** is a single repository created at the ACC to house the 'Big Data' repository. There are a number of mandatory requirements that must be met for any design to succeed, which are:

1. Data must not be modified.
2. If the data was ordered - then the order could matter and must be retained.
3. The provenance of data is important and must be maintained and not lost through the data life-cycle.
4. The data must be able to be annotated which also must not be lost.
5. Data must be extracted in bulk quickly and efficiently.

Prior to a 'Big Data' solution this research will describe the four iterations to attempt to resolve the problem. Each design progressed through a number of iterations and each significant change was indicated by a assigned name for each system developed by the ACC (see Figure 9.2):

1. **Aries** was the original already in use and largely developed by the analysts.
2. The **Shiloh** project was an attempt to develop a big data repository utilizing triple store technologies.
3. The **Eland** project was the iteration which built upon a columnar data base.
4. The **Minerva** project the final iteration that built upon the **Eland** project.

9.2 The Architecture

It is not clear yet how an optimal architecture of an analytical system should be to deal with historic data and with real-time data at the same time. An interesting proposal is the Lambda architecture of Nathan Marz (Marz and Warren, 2015). The Lambda architecture

solves the problem of computing arbitrary functions on arbitrary data in real-time by decomposing the problem into three layers: the batch layer, the serving layer, and the speed layer. As described by Nathan Marz:

'All data entering the system is dispatched to both the batch layer and the speed layer for processing.

1. The batch layer has two functions:

   (a) managing the master dataset (an immutable, append-only set of raw data), and

   (b) to pre-compute the batch views.

2. The serving layer indexes the batch views so that they can be queried in low-latency, ad-hoc way.

3. The speed layer compensates for the high latency of updates to the serving layer and deals with recent data only.

4. Any incoming query can be answered by merging results from batch views and real-time views. (Marz and Warren, 2015)'

It combines in the same way as the Hadoop system for the batch layer, and Indexes and Big Data system for the speed layer. The properties of the system are: robust and fault tolerant, scalable, general, extensible, allows ad-hock queries, minimal maintenance, and debuggable. 'Schema-Last' model has been using multiple technologies as foreshadowed in Chapter 1.

There was no clear architecture or obvious product suite that would support the Fusion Data Holding. The initial size of the Fusion Data Holding was unknown.

It is also important that any reports or statistical analysis of such large datasets are not affected by the elimination of data cleansing and as **Bradley Efron (Efron, 2010)** explains in his book about Large Scale Inference, it is easy to go wrong with huge data sets and thousands of questions to answer at once.

The solution would also need to support the following operations:

- **Distributed mining**. Many data mining techniques are not trivial to parallelize. To have distributed versions of some methods, a lot of research is needed with practical and theoretical analysis to provide new methods.
- **Time evolving data**. Data may be evolving over time, so it is important that the 'Big Data' mining techniques should be able to adapt and in some cases to detect change. For example, the **data streaming** is imperative to support data mining techniques.
- **Compression**: Dealing with 'Big Data', the quantity of space needed to store data is relevant.
- **Sampling**: Can be applied if a representative sample can found. This will reduce space required to store data in many orders of magnitude. However, data will be lost if this technique is applied.

The architecture would be required to support the formulation of Core-sets which are small sets that approximate the original data for a given problem. Using merge/map-reduce the small sets can then be used for the development of the following (Fan and Bifet, 2012):

- Early warning: develop fast response in time of crisis, detecting anomalies in the usage of data.
- Real-time awareness: design programs and policies with a more fine-grained representation of reality.
- Real-time feedback: check what policies and programs fail, monitoring it in real time, and using this feedback to make the needed change.

9.3 'Schema-Last' Approach Reference Implementation

An excellent example of a SPARQL implementation is the **Apache Jena** framework. The Apache Jena framework was originally developed by Hewlett Packard in their Bristol laboratories back in 2010. Since then, the Apache Jena is now a top-level product as of April 2012 after leaving incubation status. The framework contains a SPARQL parser (Jena

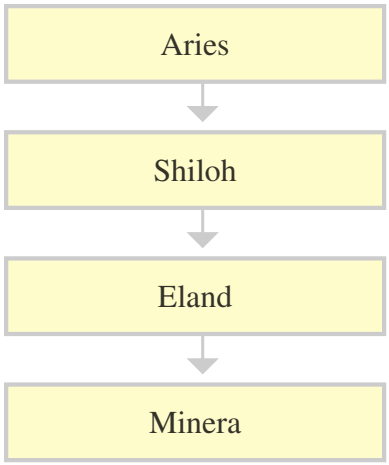


Figure 9.2: System evolution

ARQ) which transforms the SPARQL query into one or more triple patterns see Table 9.6. This provides the abstraction layer between the logical structure represented by the triplet and the physical storage implementation. Jena is not the only library that provides this functionality, Aduna's Sesame library ***openrdf*** also provides an abstraction layer between the physical and logical and supports numerous triple storage implementations. Another example is the **Redland librdf** library which also provides the same capability as Apache Jena and Aduna's Sesame frameworks.

W3C has also defined a REST style protocol to communicate to SPARQL implementations (W3C, 2008b). All three libraries support the W3C SPARQL protocol specification.

9.4 Evaluation of Existing Implementations

The comparisons of the existing implementations are based on the Lehigh University Benchmark (LUBM) software tools (Schmidt et al., 2009). The tests were performed on Sun/Oracle SPARC machine with 128GB and eight 2.2GHz processors. The operating system is Solaris 11 running a Postgres 9.2 database. The Oracle version was 11.2G with the Spatial/RDF option activated. The Sesame version from Aduna was 2.7.8 which utilized the Apache Tomcat 7 *servlet* engine. All the below implementations utilized Aduna's

Sesame as the RDF front end and provider. The Oracle Sail was not compatible with this particular version of Sesame and some modifications were required for the tested Oracle implementation to succeed.

Two main requirements had to be met by a suitable candidate for a Big Data repository. The first requirement was that the RDF triples could be loaded quickly and the memory footprint was small (less than 250 megabytes). The second requirement was that the entire triple store could be extracted or unloaded quickly and again with a small memory footprint (less than 250 megabytes).

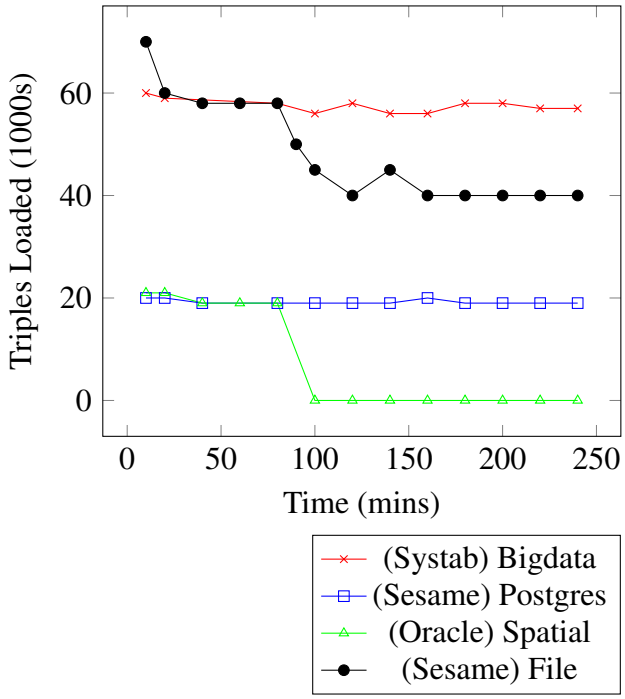


Figure 9.3: Comparison of triple store implementations (Load)

All tested implementations were able to load the RDF triplets except for the Oracle implementation which failed after only 90 minutes on both the load and extraction runs. During the Oracle extraction test, the Oracle's Performance Monitor reported an unusually high number of locks and contention on several tables that contained the triplets and this may have caused the failure. Overall, all implementations were **unable** to extract triplets

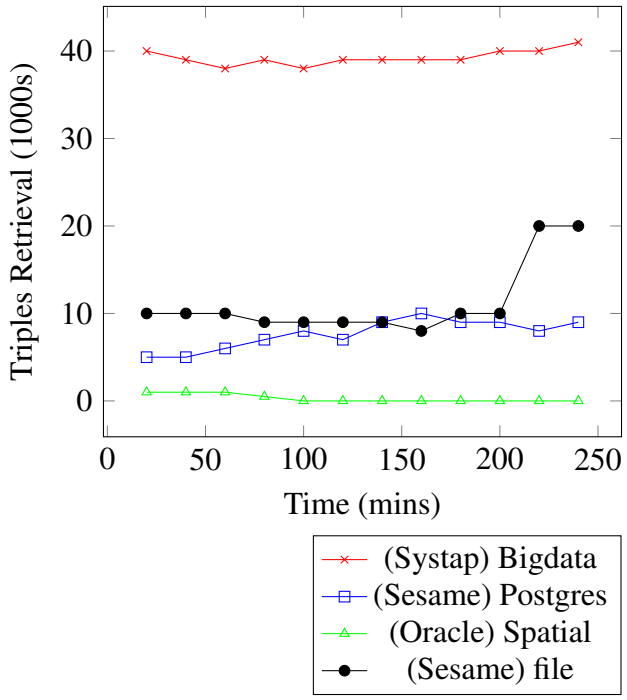


Figure 9.4: Comparison of triple store implementations (Retrieval)

without incurring a non-trivial memory leak. The overall performance of the various implementations declined over time.

The SPARQL property path expression support was also tested and the results are described in Figure 5.8. The only implementation that showed any promise was the Sesame Postgres implementation mainly through the Systap 'Big Data' *sail*. The documentation from both *Systap's Bigdata* and *Sesame's native file storage* did indicate that there is property path support within their implementation but both reported a *syntax error* when the property path clause was specified within the SPARQL select expression.

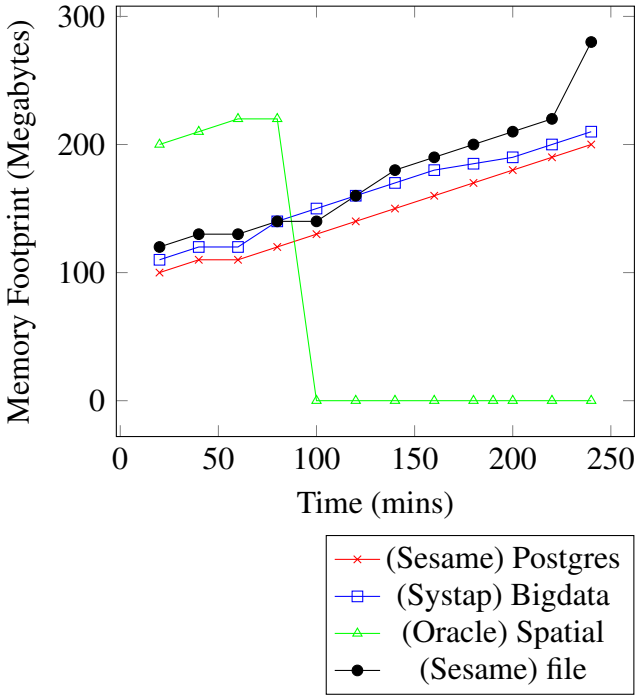


Figure 9.5: Memory footprint on retrieval

9.5 Summary of 'Schema-Last' Approach Implementations

The triple stores are new and this type of technology is not yet main stream. However, companies such as Oracle are just starting the development and implementation of triple stores. In Oracle's case they have two triple store implementations, where one utilizes relational structures to store the triplet, the other utilizes a key/value pair database. One aspect of triplets is that eight notional indexes are required to effectively support the SPARQL query language (see Figure 9.6).

The adoption of triple store technology has been slow, most data base vendors are either reluctant or slow to build triple store products. Palantir has rejected the triple store as a storage mechanism and is unlikely to consider triple stores at least in the near future. This is best summarized by Ian Davis in that he states:

Subject	Predicate	Object	Description
○	Any	Any	Retrieve all the triplets for a given subject.
○	○	Any	Retrieve all the triplets for a given object.
Any	○	Any	Retrieve all the triplets for a given predicate.
Any	Any	○	Retrieve all triplets for a given object.
Any	○	○	Return all subjects for a specific object and predicate combination.
○	Any	○	Return all the predicates for a given subject and object combination.
Any	Any	Any	Return all the triplets contained with in a graph.
○	○	○	Determine if a specific triple pattern exists within a graph.

Figure 9.6: Basic retrieval patterns

"One often overlooked advantage that RDF offers is its deceptively simple data model. This data model trivializes merging of data from multiple sources and does it in such a way that data about the same things gets collated and de-duplicated. In my opinion this is the most important benefit of using RDF over other open data formats.

Paradoxically this excellent feature is also a significant factor in the slow adoption of RDF. The reason is that RDF is a general solution to the problem of merging disparate types and sources of data. If you don't have that problem then RDF will always look inefficient, verbose and obtuse to you. Even if you are merging data today you're most likely only doing it from a few known sources and it'll be easier to write some custom code to do it for you.

I've heard many other arguments for the slow adoption of RDF over the years ranging from perceived deficiencies in the RDF model through obtuse XML formats and all the way up to techies being blamed for being bad at explaining

> RDF. I've been guilty of complaining long and hard about blank nodes and status codes. In reality, none of these things have any impact on the rate adoption of RDF because people won't use it until they have the problem it solves.
>
> This is a typical characteristic of technical paradigm shifts. No-one thought they had the problem of not being able to speak to anyone they liked wherever they were until the cellphone arrived and shifted expectations.
>
> Right now, no-one realises they have the problem of not being able to merge and combine data from thousands of different primary sources. Most people aren't thinking about it and those that do are facing an economic barrier, not a technical one. We know a general technical solution exists but the benefit/cost ratio needs to be high enough to warrant using a general solution over a custom one and today the costs of integrating data at scale are too high for most even given the massive benefits that could be possible (Davis, 2011)."

Oracle have embraced 'Big Data' technology and have introduced a triple store (RDF) front-end as part of their NoSQL (Not Only SQL) technology. Oracle did have a triple store implementation which was part of their geospatial Solution Option. Oracle recently released a RDF triple store implementation utilizing a Berkley Database for storage.

There have been a number of attempts to utilize a HBase backed triple store (Khadilkar et al., 2012). Another RDF/HBase implementation was at Stanford University in 2012 (Haque and Perkins, 2012). Both implementations take advantage of the wide column nature of the HBase columnar implementation and there is no restriction to the number of columns contained within an HBase table. The major problem is that all elements of the triples must be described as a combination of rows and columns. The paper describes 5 layout types which are described in Table 9.1.

9.6 Relational Table and Recursive Structures

Translating SPARQL to SQL can generate complex SQL statements as described in the paper by Eric Prud'hommeaux and Alexandre Bertails from **W3C** describing how this could be achieved. In their paper the authors suggest using the SQL union and group

Layout Type	Storage Schema
Simple	3 tables each indexed by subjects, predicates and objects
Vertically Partitioned (VP)	For every unique predicate, two tables, each indexed by subjects and objects
Indexed	Six tables representing the six possible combinations of a triple namely, SPO, SOP, PSO, POS, OSP and OPS
Hybrid	Simple + VP layouts
Hash	Hybrid layout with hash values for nodes and a separate table containing hash-to-node mappings

Table 9.1: Schema types

clauses but disregard the recursive select clause available in some SQL implementations. Translated SPARQL queries largely rely on the SQL optimizer for any performance gains and as Eric Prud'hommeaux and Alexandre Bertails state:

> "...that such a translation may not benefit from the capabilities of the SQL optimizer (Prudhommeaux and Bertails, 2010)."

This was also confirmed by **IBM Research** which developed a strategy to store and retrieve triplets utilizing a DB2 back-end:

> "SPARQL queries, as we illustrate in this paper. Such queries often have deep, nested sub-queries whose inter-relationships are lost when optimizations are limited by the scope of single triple or individual conjunctive patterns. To address such limitations, we introduce a hybrid two-step approach to query optimization (Bornea et al., 2012)."

In addition, the SPARQL specification has been expanded to include a property path that usually requires a form of recursion to implement. Usually relational databases are not well suited to store graph structures. This is largely due to the inconsistent approach taken by the

MYSQL:

```
WITH RECURSIVE t(n) AS (
        VALUES (1)
UNION ALL
        SELECT n+1 FROM t WHERE n < 100 )
SELECT sum(n) FROM t;
```

ORACLE:

```
SELECT name, SUM(salary) "Total_Salary" FROM (
        SELECT CONNECT_BY_ROOT last_name as name, Salary
                FROM employees
                        WHERE department_id = 110
                CONNECT BY PRIOR employee_id = manager_id)
GROUP BY name;
```

Figure 9.7: SQL recursive queries on various platforms

various data base implementations to handle recursive queries. For example **WITH RECURSIVE** is an extension provided by Postgres to allow the navigation of recursive table structures. Oracle provides an extension to the SQL language which is the **START-FROM** and **CONNECT-BY** that allows the navigation of both graph and hierarchical structures (see figure 9.7). However there are no SPARQL to SQL translators which include both Aduna's Sesame and Apache Jena implementation which takes advantage of the SQL recursive construct.

Similar recursive structures can also be navigated via complex relational joins if the recursive query is not supported. Overall, relational databases are ineffective when processing hierarchical or graph structures. This will be demonstrated by the results obtained in Figure 9.25.

9.6.1 Aries

Aries was the initial attempt to resolve the 'Big Data' issue to store the data within a relational database in a highly structured manner. This was used by the Australian Department of Human Services (DHS) with some success and that the same approach could be applied at the ACC. Aries took advantage of existing Oracle technologies which include SQL Loader and SQLPLUS utilities. The major drawback with Aries was that ETL scripts for each data source were required to be developed to load data into the Aries database. In

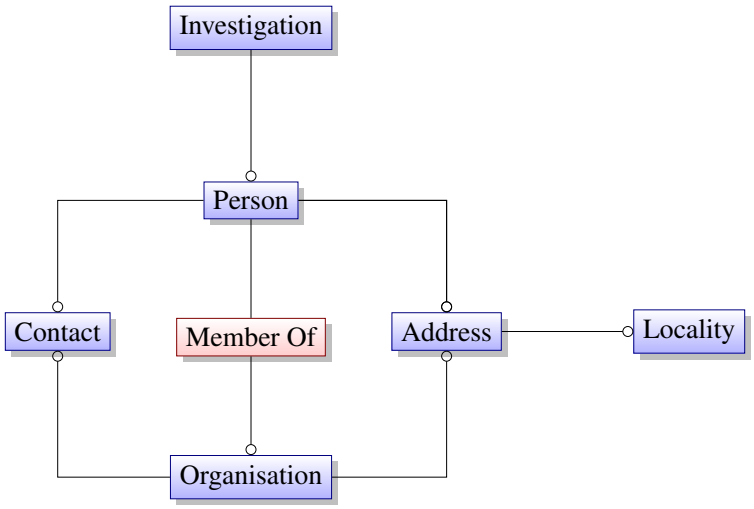


Figure 9.8: Aries schema structure

addition, a team of twelve people were needed to develop the scripts to cleanse and coerce the data into the Aries schema. The analysts then used SQ PLUS to interrogate the data cleansed and loaded by the 'upload' team for any additional analysis.

On average the team could process no more than one to two datasets per day and any data that could not fit neatly into the schema would be *discarded*. This approach also led to an unacceptable back-log of unprocessed data sets.

9.6.2 Shiloh

Shiloh was the first implementation of the Fusion Data Holding and based on Aduna's Sesame triple store implementation. Sesame is cleverly designed to allow third party storage to provide multiple storage implementations. Sesame provides a complete SPARQL 1.1 engine and SPARQL update and delete commands. In addition, Sesame provides a comprehensive well documented interface to allow third party open-source and commercial storage providers. The parsing and processing of SPARQL statements is largely performed by the Sesame kernel. In addition, the kernel is responsible for all optimization in terms of SPARQL performance. Many third party commercial vendors provide storage implementations that have been incorporated into the Sesame product suite which include: Oracle,

Systap and Virtuoso. Systap has developed an implementation called 'Bigdata' which is an optimized triple store. Aduna developed a number of implementations based on open-source technologies which include: an in memory volatile database, a file based triple store and a Postgres and MySQL. Apache Solr provided the fuzzy index capability required by Shiloh independent of the Sesam implementation.

Overall, the Shiloh was a failure largely due to the inability to extract data in bulk. The Shiloh implementation utilized both Postgres database and Systap's 'Big Data' SAIL. As the data increased meant that the Solr index strategies required adjustment which required a complete extraction all the sets contained within the stores. Both **t**he Big Data and Aduna **SAIL** implementations were unable to extract the data in bulk and therefore unable to generate the data to build the Solr indexes.

9.6.3 Eland

Eland can be be viewed as a file system of tabular data. Each file or store contains the content of the uploaded spreadsheet(s) or some tabular data. The store is composed of bags, tags, sets, models and cells. The bag itself represents an instance of a single data ingestion. Bags contain sets which can be viewed as rows in a spreadsheet or table. Each set contains one or more cells. Each cell contains a single value and together form the set. The cell also contains a sequence number which represents the cell's position within the set. The set also contains a sequence number which represents the set's position with the bag.

Sets can be returned in the order in which the rows were uploaded which has an overhead or just extracted in order determined by Cassandra (this is the quickest way to extract sets). Every set has an unique identifier or UUID. The UUID is an universal identifier in that no two sets regardless of the repository and store will have the same set identifier. The store itself is identified by a UUID which is also used to derive the keyspace name. The **base** is used as the prefix followed by the UUID. The **base** is also used to identify the Cassandra cluster. The entire keyspace name can be no longer than 42 bytes in length, this created a challenge because graph URIs were were in excess of 42 bytes. In addition,

```
/**
 * Convert the Context to a valid Keyspace Name
 *
 * @param context
 *            generally the store name but could be anything
 *
 * @return the converted context name
 *
 * @throws Exception
 *             thrown if the context name is invalid
 *
 */
private String toKeyspaceName(String context) throws Exception {
  Pattern pattern = Pattern
    .compile("[\\w]{8}-[\\w]{4}-[\\w]{4}-[\\w]{4}-[\\w]{12}");
  Matcher matcher = pattern.matcher(context.replaceAll("_", "-"));

  matcher.reset();
  if (matcher.find()) {
    return locator.getBase() + "_"
                          + StringUtils.replace(matcher.group(), "-", "_");
  }

  throw new Exception("Context must contain a UID: '" + context + "'");

}
```

Figure 9.9: Graph keyspace translation

Cassandra could only contain Alphanumeric and underscore characters. A URI name mangling algorithm was required to translate a graph URI to a valid Cassandra name-space (see Equation 9.9).

Stores also may contain tags and meta-tags. Tags are user defined and there is no restriction on how many tags a store may contain. Tags must have a name and value and the name may only contain the following national characters. The tag value may contain any value of any length including line-feeds. Meta-tags are generated by the Eland applications and are always attached to a relevant bag.

9.6.3.1 Physical Structure

Eland takes advantage of Cassandra's key features, which are:

- Cassandra can virtually have unlimited columns.

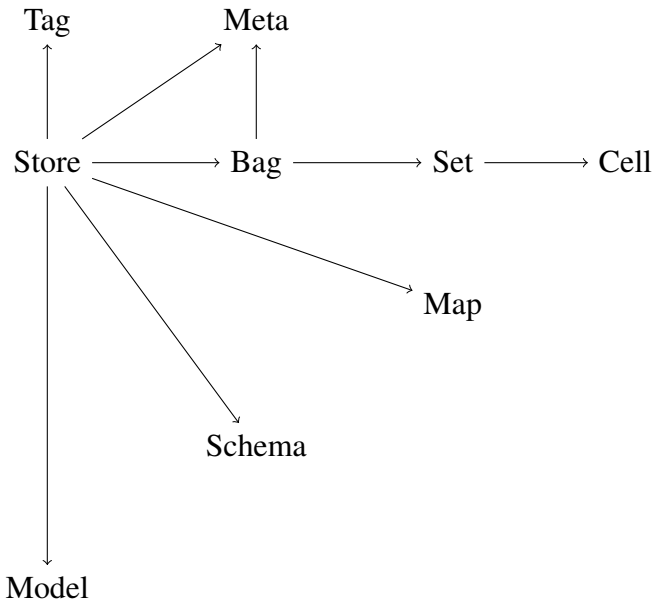


Figure 9.10: Eland logical structure

- The columns can retain an order that can be alphabetical or numerical (Eland stores columns in numerical order for lists and alphabetical order for nodes).
- Columns can be indexed via a secondary index mechanism.

There are three column families which describe the structure of the Eland. These columns families are called:

Link represents a link resource in RDF where the object is an RDF resource.

List represents an RDF list construct, however the object may be a resource or have a literal value. The predicate must comply to the W3C RDF List standard.

Node represents a collection of literal values that belong to the one resource identified by the subject.

The first column family contains a subject, predicate and object where the object is the key to the column family (see Figure 9.2). The subject and object are indexed but unlike the object are not unique (see Figure 9.11). This represents a triple in RDF.

Column Name	Key	Indexed	Description
Object	●		The triple's object
Predicate		●	The triple's predicate
Subject		●	The triple's subject

Figure 9.11: The **Link** column family

Column Name	Key	Indexed	Description
Object	●		The triple's object
Predicate		●	The triple's predicate
Subject			The triple's subject

Figure 9.12: The **List** column family

Column Name	Key	Indexed	Description
Object	●		The triple's object
Predicate		●	The triple's predicate
Subject			The triple's subject

Figure 9.13: The **Node** column family

Column Family	Description	Triple Support
LINK	The LINK column family is used to establish connections between the link and nodes or links to other links.	OSP,SPO,POS,SPO
LIST	The LIST column family is used to model RDF LINK structures. These link structures can contain up to 2 billion links.	SPO
NODE	The NODE column family is used to model an RDF node. An RDF node can only contain literal values. During the creation of the NODE column family the API allows the creation of one or more secondary indexes to identify columns that can be searched.	SPO

Table 9.2: Eland's *keyspace triple* support

The second family is used to capture ordered **RDF** list structures. Unlike the first column family, *wide* columns (see Figure 9.14) are used to represent list entries. This permits entries to be returned in *list* order. The RDF specification represents a list by an underscore '-' character followed by a number for the final part of the *uri*. Therefore the predicate contains the position of an object within a list for a specific subject (see Figure 9.12). The column family takes advantage of Cassandra's *wide* column capability (a list may only contain 2 billion entries). In the unlikely event this is insufficient there is no restriction placed upon the number of **bags** that can be contained within a store. Therefore, each item within a list is represented by a column keyed by the predicate. The object can be a link (**uri**) or a literal value. In the case where the object is a link then this **uri** can be used to identify a node or a link contained within the store.

9.6.4 Physical Artefacts within the *Set Store*

The third family is used to capture ordered **RDF** literal values. The subject is the owner of the resources and the column name is the predicate name (see Figure 9.13).

9.6.4.1 Models and Indexes

The model and predicates are used to create the Solr index entries. Each model will result in a separate Solr index. Therefore, if the store has three models then there will be three

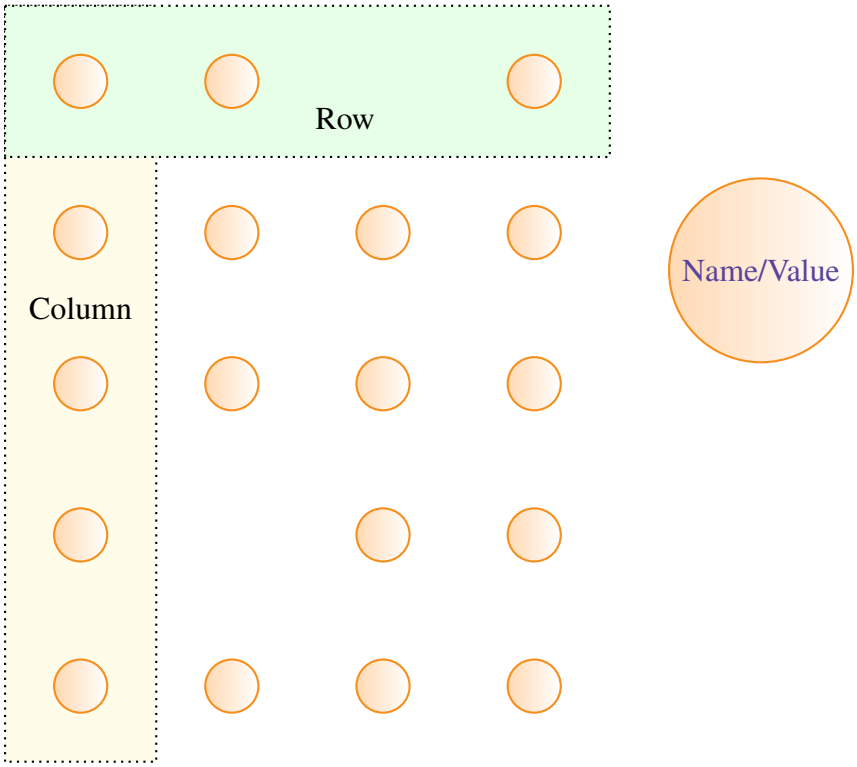


Figure 9.14: Wide Column Structure

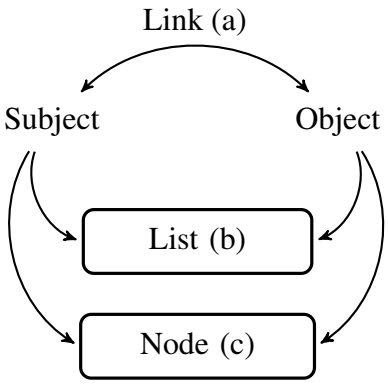


Figure 9.15: Eland structures

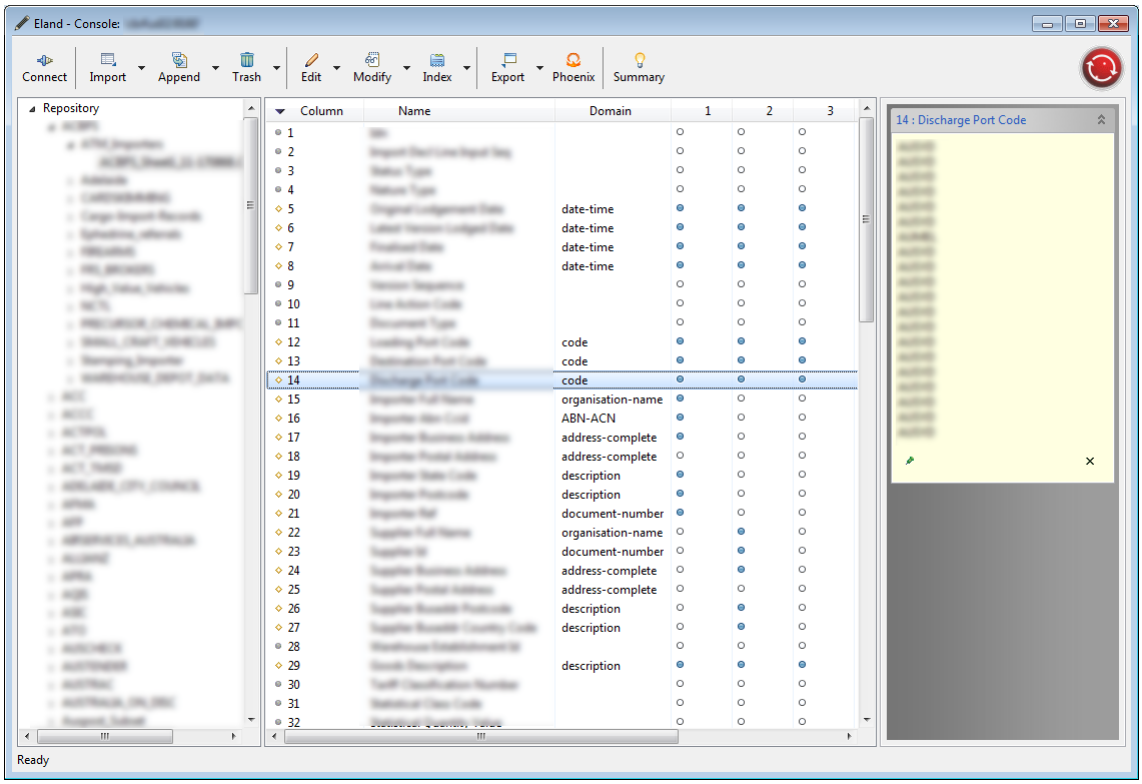


Figure 9.16: Eland console

associated Solr indexes created. There are no **restrictions** on how many models can be contained within a single store. It is permissible for a **store** to contain no models.

The schema that represents a set field entry must comply to the Solr schema. Therefore, for the field to be indexed there must be an entry contained in the Solr schema. At any stage an index may be dropped or modified. Solr does not need to be the only indexer.

Two applications, a fat and thin client (see Figure 9.16 and 9.17)[2] would allow an Eland user to create, modify and delete **Schema** definitions. The stores application would also create, update, append data to stores or bags. Both applications had limited triage capability. In general the data would be required to be represented in **comma separated format**.

[2]Note: these figures have been redacted.

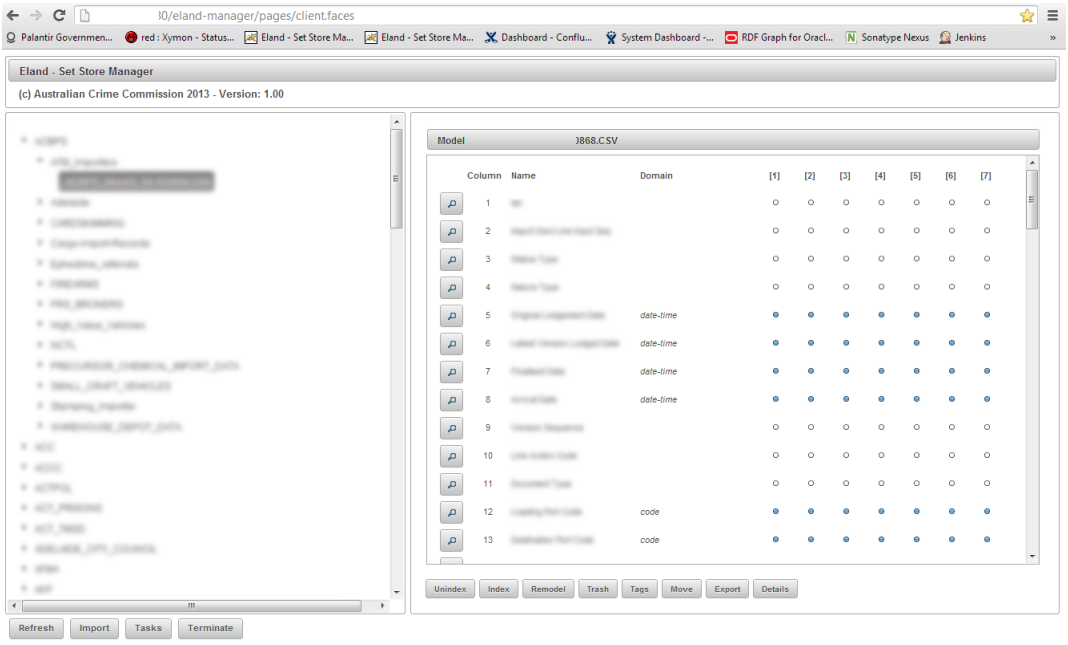


Figure 9.17: Eland thin client modeller

9.7 The Minerva Project

The poor performance of the Aduna Sesame product led to the initiation of the Minerva project which was an attempt to improve upon existing implementations specifically targeting the RDF list ingestion and extraction using the SPARQL property path extension. Taking the experience from the previous tests, this series of implementations was designed to reach the specified memory and storage footprints. This implementation was developed at the ACC as a proof of concept to determine if the triple store supported by the DataStax Cassandra columnar database was feasible The Aduna Sesame product was discarded in favour of Apache Jena which provided property path support. Cassandra was chosen as the back-end database because of its ability to distribute the work load amongst multiple nodes and its general availability on most platforms.

Cassandra consists of the following key components (Datastax, 2014):

- Cluster: a container for one or more *keyspaces*. A Cassandra instance may contain only one cluster, however the cluster may be distributed across one or more computers.

- Keyspaces: a container for *column families.* A keyspace is analogous to a schema in a relational database.
- Column families: analogous to a table in a relational database which may contain an unlimited number of rows and up to four billion columns. Each row within a column family must have a single unique key to identify the row.
- Columns: a column is name-value pair. The name can contain 64 kilobytes of data and there is no real limitation on the size of the value.

Cassandra has the unique feature where the columns are stored in a nominated order (alphabetic, numeric, date or self-defined). Cassandra also has the concept of a *super* column which is a column that may contain any number of sub-columns. Support for alternative access to data is provided through *secondary* indexes. There is no restriction on the number of secondary indexes a column family may contain, however it is recommended that the secondary indexes are of low cardinality.

Keyspaces were used to group triplets into graphs. The Jena ARQ parser provided the graph node and this was used to determine the Cassandra keyspace that contained the triplets. There is no documented limit to the number of keyspaces allowed within a Cassandra instance. HBase shares many of the same attributes with Cassandra except that there is no support for keyspaces.

9.7.1 Storage and Processing Strategies

The final design determined there were three possible storage and processing strategies which are:

- Strategy 1 [s1] : A column family for each predicate within a graph. The column family would comprise two columns which contain both the subject and object respectively. Secondary indexes would be used to select particular rows if only one subject or object is known. There is an assumption that there will only be a limited number of predicates within a graph. An MD-5 hash key of both the subject and predicate would be the key to the row. Therefore, a total of five column families are required to store the triplets which are shown in Figure 5.20.

- Strategy 2 [s2] : This strategy comprised three column families which are SPO (Subject, Predicate, Object), OSP (Object, Predicate, Subject) and PSO (Predicate, Subject, Object) to store a triplet's resources. The PSO column family takes advantage of Cassandra's capability to store any number of columns within a row. The MD-5 hash was dispensed with in favour of the target artefact value contained within the triplet. The SPO column family would be the *subject*, the OPS column family would be both the *object* and the *predicate* for the PSO column family. There was an exception for the OPS column family because an MD-5 hash value was used as the key if the *object* represented a literal. There were no secondary indexes used in this implementation (see Figure 9.19).

- Strategy 3 [s3] : A column family to contain all the literal values for a specific subject identified by the predicate (Subject, Predicate, Literal). Each row within this column family is referred to as a node. There are two additional column families for every predicate within the graph and these column families are in the form of SPO (Subject, Predicate, Object) and OSP (Object, Predicate, Subject). For the SPO column families if the object is a literal value then a placeholder - the asterisk character - is used to indicate that the value is stored within a node column family. Otherwise, the subject and object resources are rows within SPO and OPS column families where the predicate value determines the column family name (see 9.20).

9.7.2 Load Performance

All tests were performed on the same hardware as the previous evaluations but were using the latest version of Cassandra which at the time was 2.0.9. The **Thrift** protocol was used to communicate to Cassandra via an **Apache Hector** proxy. The load figures clearly demonstrate that the second strategy [s2] clearly outperformed the third strategy [s3] which outperformed the first strategy [s1] (see Figure 9.21). The LUBM test contained approximately 28 predicate *uri* specifications which meant there was an additional 24 column families and 56 secondary indexes required to support *s1* over *s*2. *s3* also required the same 24 additional column families but did not require the secondary index support.

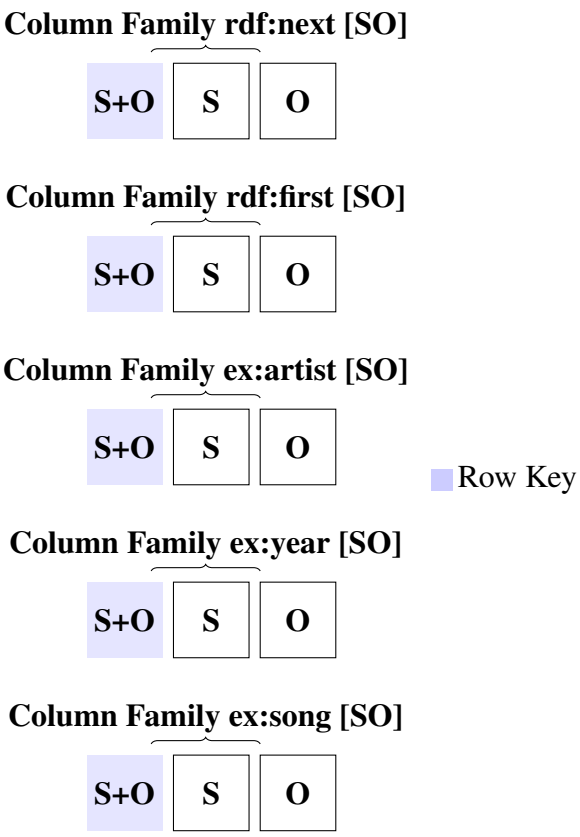


Figure 9.18: Strategy 1: Column family structure

The *s1* ingestion test was redone without the secondary indexes and subsequently the performance improved dramatically. This confirmed it was secondary index construction that caused the performance degradation, however without secondary indexes this makes *s1* unusable.

9.7.3 Extraction Performance

All strategies could support the nine possible triplet search patterns (see Figure 9.6). For *s1,* if the predicate is unknown and a subject or object is specified then all column families have to be searched for any matching triplets. For *s2*, the presence of the **PSO** column family that was keyed on the predicate removed the need to search multiple column families. s3

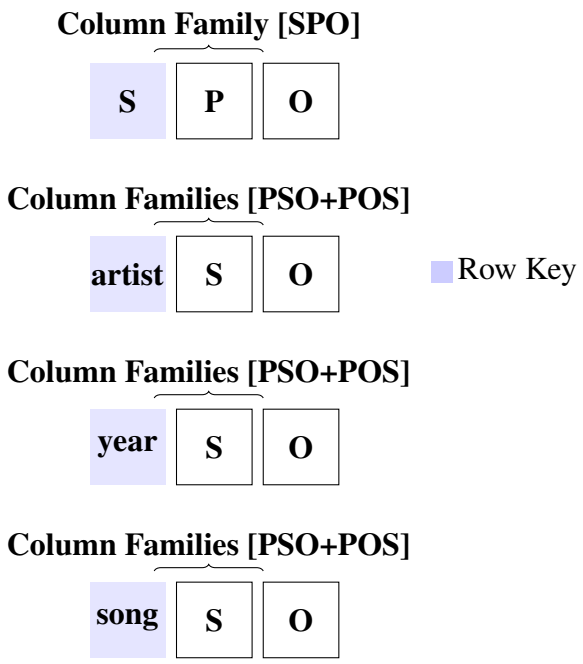


Figure 9.19: Strategy 2: Column family structure

could utilize the column family name which happened to be the predicate *uri* to efficiently traverse the RDF list structure (see Figure 9.22). There were no detectable memory leaks with either the *s1, s2* or *s3* implementations during both the ingestion and extraction tests (see Figure 9.23).

9.7.4 Property Path Support

The unbounded property path expression's performance was problematic for both *s1*, *s2* and not so with *s3* as show in Figure 9.24 where *elt* is a path element to navigate to and the *number* in braces is the degree of separation. To improve performance of *s2* the *link* triplets involved in RDF list structures were placed in a separate *column family* with two rows keyed by the **rdf:first** and **rdf:last**. Each link is assigned an ascending number which is used as the column identifier. This meant that triplets can only ever be appended to an RDF list, therefore if insertions are required anywhere within the list structure other than at the end then the entire list would have to be reconstructed.

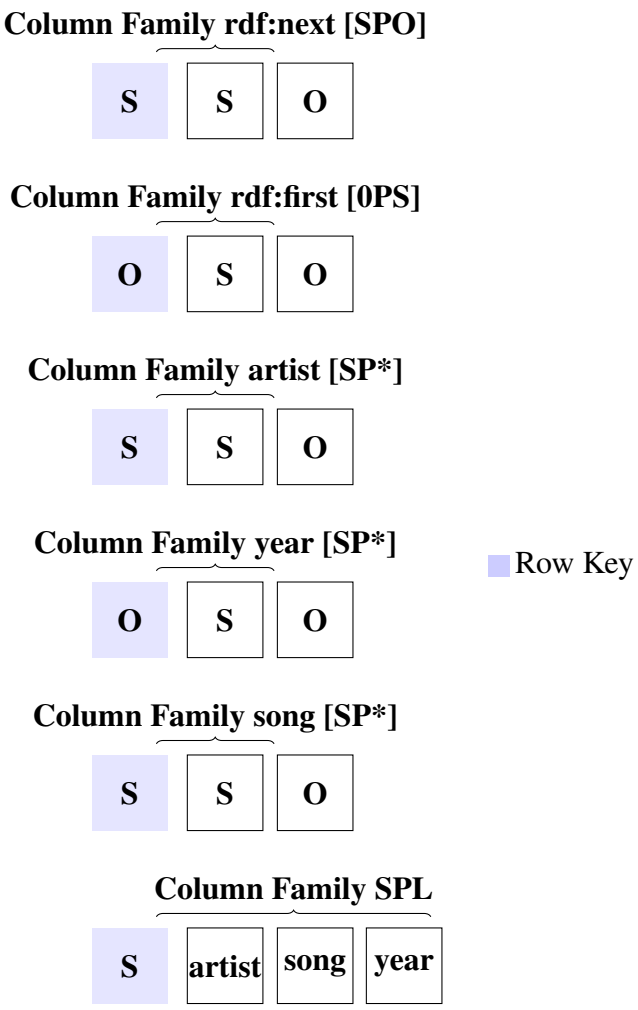


Figure 9.20: Strategy 3: Column family structure

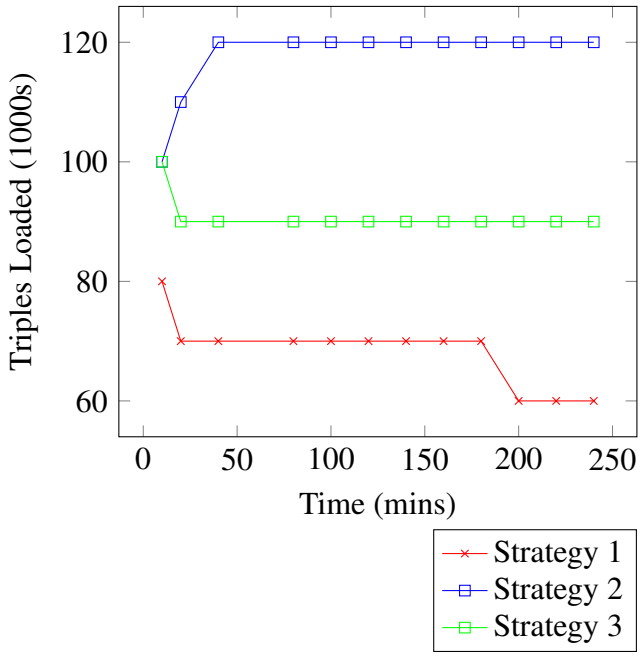


Figure 9.21: Minerva - Load performance

9.7.5 Implementation Acceptance

The decision was made to deploy *s3* over *s1* and *s2*. Utilizing s*3,* tabular data in excess of 60 million rows was loaded in approximately 5 hours and extracted in 6 hours.

The initial reaction to triple store implementation has been positive amongst the test audience. However it took some time to train the users to familiarize themselves with the SPARQL query language syntax and how best to use the property path expression to query directed graph structures.

9.7.6 The Bulk Matcher

The bulk matcher was a tool developed at the ACC to bulk search the Big Data repository. This tool proved to be a great success amongst the analysts and other Australian intelligence agencies which would submit their own lists to the bulk matcher. There are approximately three bulk match requests per week which could contain any number of names. Other Australian government agencies have expressed interest in developing similar capability.

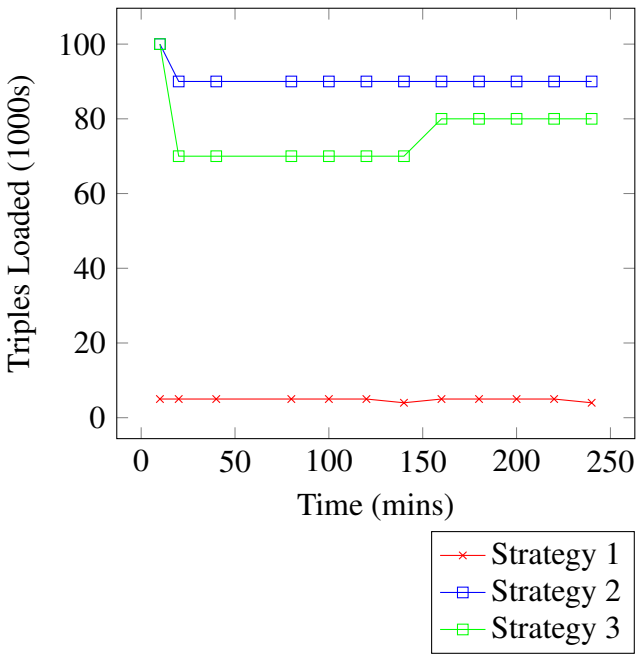


Figure 9.22: Minerva - Extraction performance

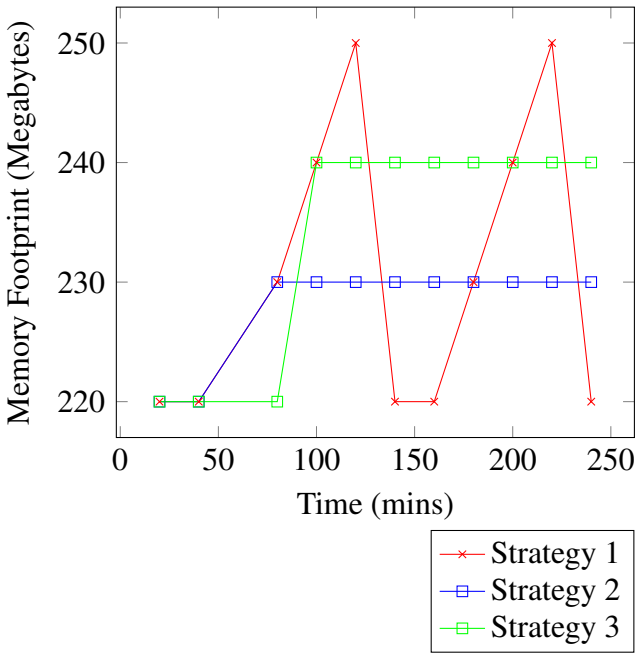


Figure 9.23: Minerva - Memory footprint on extraction

Property Path	elt*	elt{,5}	elt{,10}
s1	✔	✔	✔
s2	✔	✔	✔
s3	✔	✔	✔

Figure 9.24: Minerva - Tested property path expressions

As an example, over the last month (September 2014) there have been over 50 bulk match requests.

9.8 Other Implementations

The CORES (Heery and Johnston, 2005) meta-data schema registry is designed to enable users to discover and navigate meta-data element sets. The paper reflects on some of the experiences of implementing the registry, and examines some of the issues of promoting such services in the context of a "partially Semantic Web" where meta-data applications are evolving and many have not yet adopted the RDF model. The CORES project has explored the potential for supporting the creation and reuse of meta-data schema definitions using Semantic Web technology.

The Optique project (Pinkel, 2013) is an attempt to join the Semantic Web at developing an end-to-end system for semantic data access to 'Big Data' in industries such as Statoil ASA and Siemens AG. The first version of the Optique system was customized for the Norwegian Petroleum Directorate's Fact Pages, a publicly available dataset relevant for engineers at Statoil ASA. The system provides different options, including visual aids to formulate queries over ontologies and to display query answers. Optique offers installation wizards that allow the analyst to extract ontologies from relational schemas, extract and define mappings, connecting ontologies and schemas, and align and approximate ontologies. Moreover, the system offers highly optimized techniques for query answering utilizing the SPARQL language and OWL ontologies.

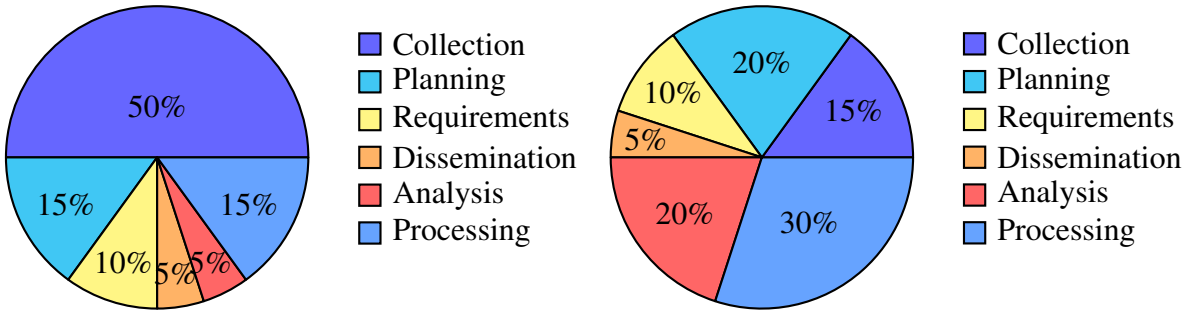


Figure 9.25: Comparison between the 'Schema-First' and 'Schema-Last' Approach

9.9 Summary

The success of the 'Schema-Last' Approach within the ACC enabled the organization to quickly collate and process data received from external and internal providers. The dramatic reduction in time taken to ingest data allowed analysts to examine and process data and not be required to cleanse and transform data (see Figure 9.25). The usage of Apache Pig and Map reduction enabled analysts to generate criminal and test models that were not possible with the ARIES system. The analysts now had at their disposal a number of tools that were previously unavailable to them.

The use of a triple-store to capture and store the data was also successful. The implementation - Eland (which is still in production and soon to be replaced with Minerva) - allowed the users to store meta-tags and relationships within the stores as RDF triplets.

Palatir was adopted as the intelligence recording tool by the ACC. Eland was integrated into Palantir where a semantic representation could be captured within the store and applied to every set contained within the store. Palantir would use that semantic mapping to represent as PXML which is a Palantir XML representational language. A Palatir plug-in (the search assistant) would enable analysts to search the Eland stores. A screen shot of the **search assistant** is shown in Figure 9.26 It was not unusual for analysts to begin an investigation to search for any record of a person or entity within the Fusion Data Holding.

Overall, Eland and Minerva proved to be a great success amongst analysts within the ACC and an example to other intelligence agencies on how to collect and process data as part of the Intelligence Life-Cycle.

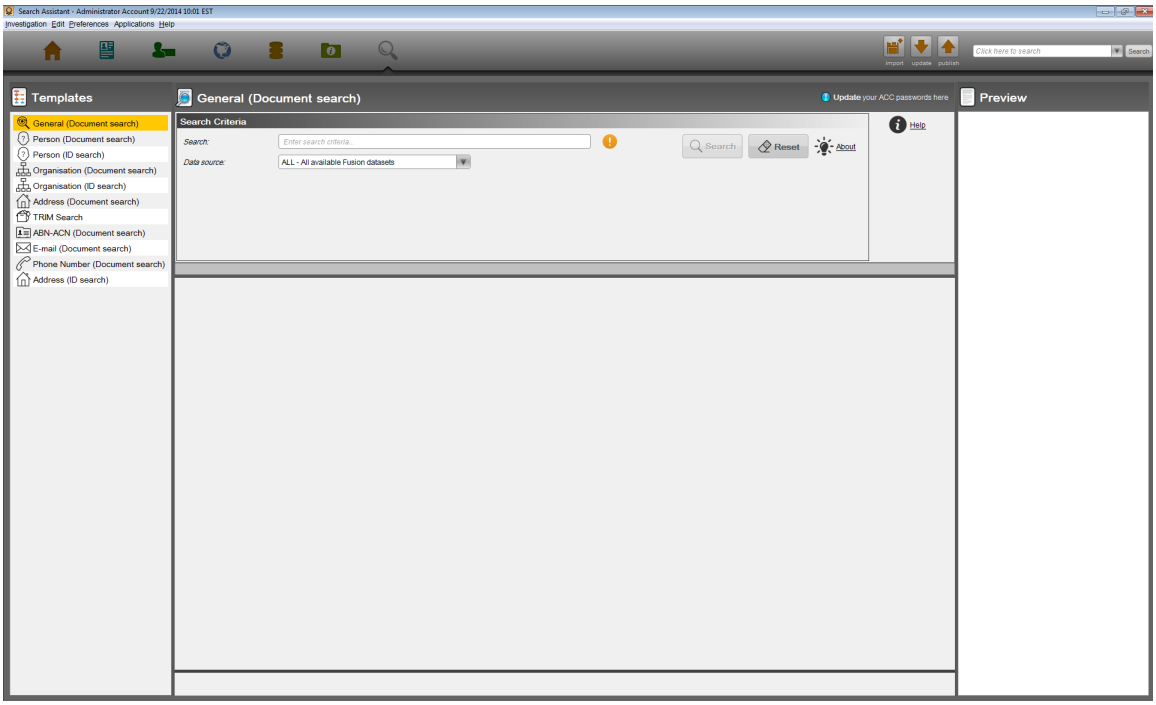


Figure 9.26: The Palantir Search Assistant

The backlog that caused the initial concern by ACC management was brought under control. Eland was introduced around July 2012 and made an immediate impact with the elimination of the data source processing backlog (see Figure 9.27). Every data source was indexed using both Apache Solr and and Informatica's Identity Resolution product. The analysts were then able to search the Fusion Data Holding and generate *core-sets* for further data analysis.

The incorporation of the 'Schema-Last' addresses the problem depicted in Figure 9.1 which utilized the pre-existing 'Schema-First' Approach. Furthermore, the ACC has been able to reduce the staff required to *clean* data.

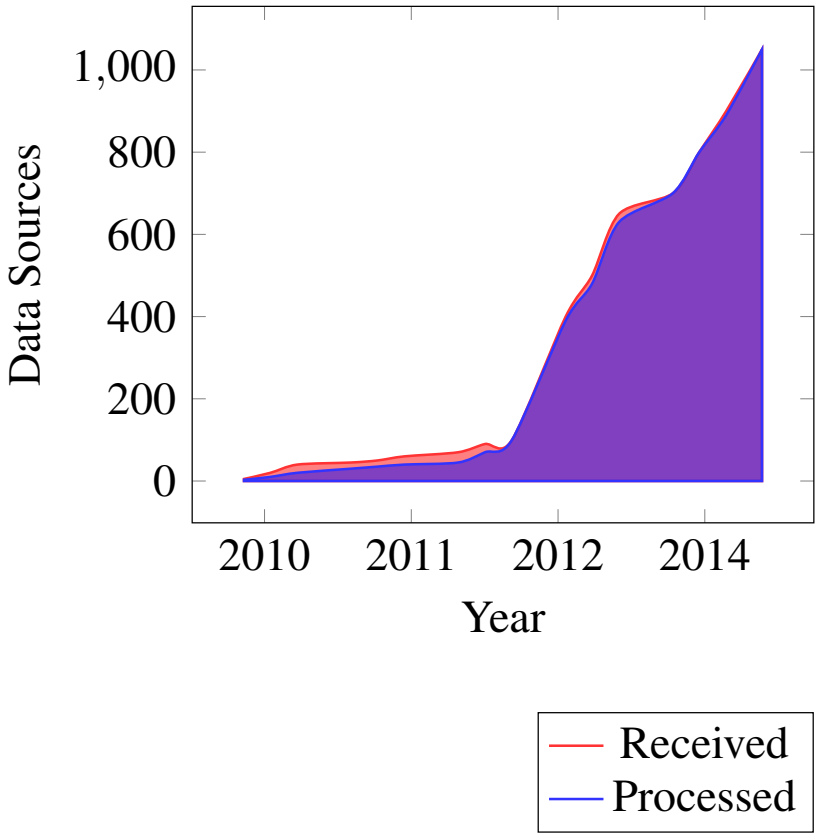


Figure 9.27: Data source processing summary: 2009 - 2014

Chapter 10

Conclusion and Further Work

Most of the questions for research have been investigated. The proposed new 'Schema-Last' Approach demonstrates a strong value add to the analysis of 'Big Data' especially within the Intelligence Life-Cycle.

The ACC now has an improved tool to address the eclectic nature of the data sent to them. The 'Schema-Last' Approach can be used to define index strategies, provide the *map* in **map** reduction and the foundation for data mining and analytical processing. The formal definition of the schema syntax allows for the interchange of models and meta-data amongst organizations, institutions and available to the general community. Overall, the 'Schema-Last' Approach has provided the platform to *fuse* data into a consolidated view and to resolve issues associated with variability and variety of data when obtained from multiple sources.

There were issues pertaining to the amount of data returned from any search of the fusion data holding. Common name searches could yield thousands of results and it was difficult for analysts to process these large result sets. The case study presented in the previous chapter demonstrated that 'Big Data' and semantic technologies can be successfully incorporated into the Intelligence Life-Cycle and because the Intelligence Life-Cycle shares many characteristics with the CRIPS-DM framework then this can be applied to other data mining problems. Furthermore, the ability to effectively represent tabular data within RDF structures enables data mining tools that utilize map-reduce algorithms to take advantage of data stored that way.

The general success of the Eland implementation has demonstrated that it is possible to store RDF list structures in a columnar data base. Both Eland and Minerva utilized the Cassandra as a back-end storage mechanism and it was found to be suitable to store billions of triplets. The case study has clearly demonstrated that there is really no logical restriction to the amount of data that can be held in a columnar data base. Furthermore, having separate indexes for the RDF object was an effective strategy to allow analysts to search the 'Big Data' repository.

The acceptance of the 'Schema-Last' Approach has had a profound effect on the approach taken by the ACC with reference to the Intelligence Life-Cycle. The collation and process phase of the Intelligence Life-Cycle has now been largely automated with the introduction of the Eland System. Eland was the second attempt at addressing the automation of the collation and processing phases of the Intelligence Life-Cycle and the bulk matcher has gone someway to automate the dissemination phase. There are some political and cultural hurdles to overcome to have a fully automated dissemination process in that management feel that dissemination of intelligence products requires a human to *sign-off and confirm* before the product can be released.

Another observation was that the analysts did not append data utilizing the *bag* artifact to existing stores. The analysts preferred to create a new store for a new feed or a new snap-shot and use the folder structure to group store updates together.

Much debate was had amongst the analysts on how to replace or update an existing *set store*. The general consensus was to never drop a store but drop the store's index entries and leave the store's data in place.

Overall, data triage has proven to be an effective replacement over the existing ETL technology stack. Furthermore, schema definitions are used throughout the Intelligence Life-Cycle, not at the beginning as with the traditional 'Schema-First' approach.

The resulting 'Schema-Last' model has demonstrated enormous potential and interest in the intelligence community and its validation against ACC's data holdings and has been received favorably amongst the analysts and senior management.

There is still a place for the 'Schema-First' approach in that if the data is transferred from new to legacy systems and it is unlikely that this process will change then use ETL and predefined schemas. If however, the data source can alter the data's structure at any

Purpose	Schema-First	Schema-Last
Data provenance matters		●
Raw data must be retained		●
Data source never changes the structure	●	
Data is to be used for analytical purposes	●	●
Data is a part of a business process	●	

Table 10.1: Schema-First/Schema-Last Comparison

time then 'Schema-Last'approach should be taken (see Table 10.1). In addition, if the data is part of a business processing system then the 'Schema-First' approach is a reasonable option.

10.1 Response to Thesis Questions

The following sections are the responses to the thesis questions as specified in the introductory chapter.

10.1.1 Data Management and Stewardship

The following is the response to the thesis questions relating to data management and stewardship:

- How best to *ingest* data received from external sources?
  - The 'Schema-Last' Approach has been well received as an alternative to the existing Extract Transform Load or 'Schema-First' Approach that was in place.
- How to process data in a timely manner?

  - The 'Schema-Last' Approach has proven to be an effective method to process data without the need for data cleansing or development of complex data processing scripts. The introduction of the triage process has effectively reduced the time taken to collate and process data sets from months to minutes.

- How to retain data *provenance* to ensure that all data can be traced back to the original source?

  - The introduction of meta-data and the allocation of a *set identifier* ensures that data provenance is retained.

- How to ensure that data is not changed or that the data meaning is not lost through modification and transformation?

  - The 'Schema-Last' Approach does not alter the raw data but transforms the structure if required.

10.1.2 Data Quality

The following is the response to the thesis questions pertaining to data quality:

- How to deal with data sets with messy or noisy data values?

  - The 'Schema-Last' Approach does not impute missing values or alter the original data values to comply to a schema specification.

- How to deal with data sets with no *identifiable* primary key?

  - It is not uncommon for data sets not to contain a primary key, therefore the *set identifier* becomes the data source's primary key. Meta-data is used to indicate if the data source contains an external primary key.

- What if the time and man-power taken to *clean* and *collate* data exceeds the agency's processing capability?

  - The case study clearly demonstrated that the manpower and the time taken to clean and collate data was significantly reduced. This will allow the agency to process any future data collation demands well into the future. The Minerva implementation utilizing RDF and columnar technologies performed well beyond expectations in its ability to collate and process data sources.

- How to deal with data values that have an ambiguous value or meaning?

  - The data values and meaning are always respected. If the data value is ambiguous then this ambiguity is managed by index strategies where multiple tokens are generated to represent any data ambiguity.

10.1.3 Data Fusion

The following is the response to the thesis questions pertaining to data fusion:

- How to provide *consistent* fused view between the data sets contained within the agency?

  - The 'Schema-Last' models and domains are designed to provide a consistent view between the data sets. In addition, these models can be changed or multiple models can be applied to the one data source. A *formal* nomenclature is defined to represent 'Schema-Last' artefacts which include the models and domains.

- How to fuse and analyze data on demand?

  - The 'Schema-Last' has provided the platform to fuse and analyze data on demand. The significant reduction in the time taken to collate and process data has now meant that data can be analyzed in real time. In addition, the 'Schema-Last' Approach supports the CRISP-DM framework and specifically targets the collation and process phases of the Intelligence Life-Cycle.

10.2 Problem Statement Response

The following is the response to the problem statement specified in the introductory chapter:

1. Lack of an agreed ideal end state for the fusion capability.

    - The 'Schema-Last' Approach provided the platform to support the collation and processing phases of the Intelligence Life-Cycle. The reference implementation proved to be a great success in eliminating the the need for data cleansing. The data triage becomes the formal collation process within the ACC.

2. Lack of core data management function and data management regime around bulk data holdings.

    - The 'Schema-Last' Approach becomes the core management regime and defines all the procedures related to the bulk data holdings.

3. Data entry functions are cumbersome and time consuming due to inflexible data structures.

    - The 'Schema-Last' Approach defined the flexible data structures required to promote data ingestion and ultimately led to significant reduction in collation and processing.

4. Search and discovery capabilities are highly ineffective; due to lack of connectedness of data silos across different systems and networks. Early consideration of the key problems in the agency identified an inability to answer "what do we know?" and as a result, an **Advance Search Capability** was developed.

    - This was the greatest challenge and as a result a number of index strategies were defined to allow easy access to the data contained within the **Fusion Bulk Data** holding.

5. Excessive time spent collating data rather than spending time analyzing the data.

- There was a significant improvement in collating data which in turn led to more time dedicated to analyzing data. Both Eland and Minerva demonstrated that raw data could be collated and processed without dedicating significant amounts of time and effort to **cleanse** or **coerce** data into a predefined schema. The time is now spent describing and analyzing the data.

6. Identities are only able to be matched by converting data to a standard format across all data sources. There is an inability to handle *messy* data where the data was either poorly structured or contained a variety of data formats.

   - Unlike the schema definitions used by the other approaches ('Schema-First' Approach) the Schema definitions allow for the collation and analysis of *messy* data. The 'Schema-Last' Approach does not change or alter the raw data; models and domains can be used to describe the the messy data

7. Lack of a single collaborative platform for discovery, collation and analysis of data holdings. The approach taken by many analysts is to use *Analyst Notebook* and *Microsoft Excel.* They have been the primary analysis tools used by analysts. These tools do not have access to all data sources available within the agency.

   - The data fusion and matching chapters enable organizations to share schema definitions. The schema definition can be expanded to include new fields which can be transmitted along known data items.

8. Lack of sufficient basic analysis tools available enterprise wide including social network analysis (SNA), temporal data mining and geospatial analysis.

   - With the correct meta-data applied to the set data the addition of geospatial coordinates provides the basis for any geospatial analysis to be performed on the raw data. In addition, the index strategies as described in Chapter 6 support geospatial searching and Apache Solr. The indexing system within the reference implementation a was able to perform geospatial radius and polygon searches.

9. Detection of previously unknown high risk entities is limited to data matching processes due to a lack of time contiguous data sets.

    - The 'Schema-Last' Approach did not directly address this problem statement, however the dramatic reduction of time taken to process any new data sets in turn led to an overall improvement with data matches.

10. Collected data that is not managed according to an agreed process and security model.

    - The 'Schema-Last' Approach became the agreed process to support the collation and processing phases of the Intelligence Life-Cycle.

11. Detection of previously unknown high risk entities is limited to data matching process that cannot take advantage of the complete data sets.

    - The approach taken does not remove or change data from the raw state.

12. Internal alerting capabilities where Persons Of Interest (POIs) can be monitored.

    - POIs are able to monitored by the arrival of new data sets and feeds. A special list of known criminals and their associates can be actively monitored with the non-prescriptive schema definitions.

13. External alerting capabilities from partner agencies, will enable external agencies to have the ability to monitor POIs and report the results back to the ACC.

    - Schema definitions can be shared amongst agencies and can be used as a specification for data exchange. Unlike XML schema definitions, the model and structures are based domains which is a range of a values rather than a primitive data type.

14. Improve real-time access to data. This also includes the ability to Social Network Analysis (SNA) to identify groups or cliques, identify network density and identify possible POIs.

- The semantic and match store are specifically designed for this purpose. The combination of the store models, ontological structures, the links and match results allow for the identification and analysis of graph structures. This will allow the analysts to identify cliques, graph density and provide the platform for Social Network Analysis.

15. Lack of capacity to develop advanced analytic tools. The NCTL (National Criminal Target List) and the validation of the ACC's TRAM (Threat Risk Assessment Model).

    - The 'Schema-Last' Approach provides the capability and analytical platform to develop other analytical tools and provides the platform for any future analysis. Furthermore, the NCTL could be another *set store* within the repository.

10.3 Further Work

The research undertaken has adequately addressed the research question established for this work. Also, the process at arriving at an implementation went through a number of implementations and the work is still in progress. The *match store* (see Chapter 5.8) has not been fully implemented. The premise is that the bulk matcher would ultimately produce the potential matches and these matches would be recorded within the *match store*. Automated alert models can take advantage of the 'Schema-Last' models. Perhaps the greatest concern amongst analysts was the large result set returned from searches. The results sets could number in the many thousands and overwhelm the analyst. More research would be required to determine the most appropriate approach to display large search required to the user.

The three stores classifications, **set**, **semantic** and **match,** provide the foundation for the **'Schema-Last' Approach**. There may be other relationships that need to be captured and this may require other store classifications.

The schema structure and definition was deliberately kept simple so that analysts at the ACC with little training could successfully define and create set store models. If this is not sufficient, models can be extended to include other attributes that may be used as input to analytical processes.

Finally, the application of the ‘Schema-Last’ Approach artefacts is not just limited to index creation and data fusion, there may be other analytical processes that could take advantage of these artefacts for other purposes. Any application that must process dirty data should seriously consider the ‘Schema-Last’ Approach.

Bibliography

Aduna. Sesame user manual, 2013. URL http://www.openrdf.org/doc/sesame2/users/ch01.html.

Mouhib Alnoukari and Asim El Sheikh. Knowledge discovery process models: From traditional to agile modeling, 2012. URL http://www.irma-international.org/viewtitle/58566/.

Amazon. Amazon redshift, 2014. URL http://aws.amazon.com/redshift/.

J.A Armstrong. The funding base for australian biological collections. *Australian Biologist*, 5:80–81, 1992.

Mark Bedworth and Jane O'Brien. The omnibus model: A new model of data fusion?, 2000. URL http://isif.org/fusion/proceedings/fusion99CD/C-075.pdf.

Tim Berners-Lee. Uri specification. w3c, 1991. URL http://www.w3.org/Addressing/URL/uri-spec.html.

Anuradha Bhatia and Shefali Patil. Column oriented dbms an approach, 10 2011. URL http://www.ijcscn.com/Documents/Volumes/vol1issue2/ijcscn2011010203.pdf.

Joachim Biskup. *Lecture Notes in Computer Science*. Springer-Verlag, 1987.

Mihaela A. Bornea, Julian Dolby, Anastasios Kementsietsidis, Kavitha Srinivas, Patrick Dantressangle, Octavian Udrea, and Bishwaranjan Bhattacharjee. Building an efficient rdf store over a relational database. Technical report, IBM Research, 2012. URL https://cs.uwaterloo.ca/~gweddell/cs848/papers/Bornea.pdf.

John Boyd. A discourse on winning and losing. Lecture, 1987.

Michael E. Bratman. *Beliefs, Desires and Intentions*. CSLI Publications, 1999.

N. Brierley, T. Tippetts, and P. Cawley. Data fusion for automated non-destructive inspection. *Proceedings of the RSPA*, 2014. URL http://rspa.royalsocietypublishing.org/content/470/2167/20140167.abstract.

Federico Castanedo. A multi-agent architecture based on the bdi model for data fusion in visual sensor networks. *Journal of Intelligent & Robotic Systems*, 62(3-4):299–328, 2011. ISSN 0921-0296. doi: 10.1007/s10846-010-9448-1. URL http://dx.doi.org/10.1007/s10846-010-9448-1.

Fay Chang. Bigtable: A distributed storage system for structured data, September 2006. URL http://static.googleusercontent.com/bigtable-osdi06.pdf.

Arthur D Chapman. *Principles of Data Quality*. Australian Biodiversity Information Services, 2005.

Hsinchun Chen, Wingyan Chung, Jennifer Jie Xu, Gang Wang, Yi Qin, and Michael Chau. Crime data mining: A general framework and some examples. *IEEE*, 37:50–56, 2004. URL http://hdl.handle.net/10722/45461.

William W. Cohen. A comparison of string distance metrics for name-matching tasks, 2001. URL https://www.cs.cmu.edu/~pradeepr/papers/ijcai03.pdf.

Australian Crime Commission. Australian crime commission fusion capability. 2012.

Kenneth Neil Cukier and Viktor Mayer-Schoenberger. The rise of big data - how it's changing the way we think about the world, 2013. URL http://www.foreignaffairs.com/articles/139104/.

Quick Darren and Kim-Kwang Raymond Choo. Data reduction and data mining framework for digital forensic evidence: Storage, intelligence, review and archive, September 2014. URL http://aic.gov.au/media_library/publications/tandi_pdf/tandi480.pdf.

Belur V Dasarathy. Sensor fusion potential exploitation innovative architectures and illustrative applications. *Proceedings of the IEEE*, 85:24–38, 1997.

Tamraparni Dasu and Theodore Johnson. *Exploratory Data Mining and Data Cleaning:*. Wiley, 2003.

Datastax. Cassandra technical manual, August 2014. URL `http://www.datastax.com/`.

Ian Davis. The real challenge for rdf is yet to come, August 2011. URL `http://blog.iandavis.com/2011/08/18/the-real-challenge-for-rdf-is-yet-to-come/`.

Edd Dumbill. What is big data?, January 2012. URL `http://radar.oreilly.com/2012/01/what-is-big-data.html`.

Bradley Efron. *Large-Scale Inference: Empirical Bayes Methods for Estimation, Testing, and Prediction*. Cambridge University Press, 2010.

Timo Elliott. 7 definitions of big data you should know about, July 2013. URL `http://timoelliott.com/blog/2013/07/7-definitions-of-big-data-you-should-know-about.html`.

EMC. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east, December 2012. URL `http://www.emc.com/leadership/digital-universe/2012iview/executive-summary-a-universe-of.htm`.

Wei Fan and Albert Bifet. Mining big data: Current status, and forecast to the future, 2012. URL `http://www.kdd.org/sites/default/files/issues/14-2-2012-12/V14-02-01-Fan.pdf`.

Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17, 1996.

John Gantz and David Reinsel. The digital universe in 2020: Big data, bigger digital shadow s, and biggest growth in the far east, December 2012. URL `http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf`.

Lars Marius Garshol. Rdf triple stores - an overview, 9 2012. URL http://www.garshol.priv.no/blog/231.html.

Gartner. What is big data, 2014. URL http://www.gartner.com/it-glossary/big-data/.

Tom Gruber. What is an ontology, 1993. URL http://www-ksl.stanford.edu/kst/what-is-an-ontology.html.

Philip J Guo, Sean Kandel, Joseph M Hellerstein, and Jeffrey Heer. Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 65–74, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0716-1. doi: 10.1145/2047196.2047205. URL http://doi.acm.org/10.1145/2047196.2047205.

Albert Haque and Lynette Perkins. Distributed rdf triple store using hbase and hive, December 2012. URL http://web.stanford.edu/~akhaque/downloads/cs370.pdf.

Christopher Hayes. Parliamentary joint committee on law enforcement - inquiry into the gathering and use of criminal intelligence., May 2013. URL http://www.aph.gov.au/~/media/wopapub/senate/committee/le_ctte/completed_inquiries/2010-13/criminal_intelligence/report/report.ashx.

Rachel Heery and Pete Johnston. Metadata schema registries in the partially semantic web: the cores experience, 2005. URL http://opus.bath.ac.uk/23575/1/102_Paper29.pdf.

Scott Henry, Sherlynn Hoon, Meeky Hwang, Diane Lee, and Michael D. DeVore. Engineering trade study, extract, transform, load tools for data migration, 2005. URL http://www.sys.virginia.edu/sieds05/proceedings/A101.pdf.

Oliver Higgins. *The theory and practice of intelligence collection*. The Federation Press, 2009.

Pascal Hitzler, Markus Krotzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. CRC Press, 2010.

Informatica. Data fusion for cyber intelligence, 2014. URL http://www.informatica.com/Images/02435_data-fusion-cyber-intelligence_eb_en-US.pdf.

ISO/IEC. Information technology – metadata registries (mdr, April 2012. URL http://metadata-standards.org/.

Vaibhav Khadilkar, Murat Kantarcioglu, Bhavani Thuraisingham, and Paolo Castagna. Jena-hbase: A distributed, scalable and efficient rdf triple store. *The University of Texas at Dallas*, 2012.

Daniel T. Larose. *Discovering Knowledge in Data: An Introduction to Data Mining*. John Wiley & Sons, 2014.

Jimmy Lin and Dmitriy Ryaboy. Scaling big data mining infrastructure: The twitter experience. *SIGKDD Explor. Newsl.*, 14(2):6–19, April 2013. ISSN 1931-0145. doi: 10.1145/2481244.2481247. URL http://doi.acm.org/10.1145/2481244.2481247.

David S. Linthicum. Understanding the symbiosis of cloud computing, big data, and mobile, March 2013. URL http://research.gigaom.com/2013/03/understanding-the-symbiosis-of-cloud-computing-big-data-and-mobile/.

Steve Lohr. For big-data scientists, 'janitor work' is key hurdle to insights. *New York Times*, August 2014. URL http://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html.

Apache Lucene. Welcome to apache lucene, 3 2015. URL http://lucene.apache.org/.

Nathan Marz and James Warren. *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publishing, 2015.

Vincent McBurney. 17 mistakes that etl designers make with very large data, 2007. URL http://it.toolbox.com/blogs/infosphere. http://it.toolbox.com/blogs/infosphere/17-mistakes-that-etl-designers-make-with-very-large-data-19264.

Gery Menegaz. What is nosql, and why do you need it?, October 2012. URL http://www.zdnet.com/what-is-nosql-and-why-do-you-need-it-7000004989/.

Microsoft. Data cleansing, 2012. URL http://technet.microsoft.com/en-us/library/gg524800.aspx.

Marie-Franice Moens. *Automatic Indexing and Abstracting of Document Texts*. Klumer Academic Press, 2002.

Elkan C Monge, A. The field matching problem: Algorithms and applications. In *Proceedings of The Second International Conference on Knowledge Discovery and Data Mining*, 1996.

Emily Namey, Greg Guest, Lucy Thairu, and Laura Johnson. Data reduction techniques for large qualitative data sets, March 2007.

Thor Olavsrud, July 2014. URL http://www.cio.com/article/2449814/big-data/data-scientists-frustrated-by-data-variety-find-hadoop-limiting.html.

Christoph Pinkel. Optique 1.0: Semantic access to big data, 2013. URL http://scholar.google.com.au/citations?view_op=view_citation&hl=en&user=Jd7PlLEAAAAJ&citation_for_view=Jd7PlLEAAAAJ:zYLM7Y9cAGgC.

Fiona Swee-Lin Price. *success with Asian names*. Allen&Unwin, 2007.

Eric Prudhommeaux and Alexandre Bertails. A mapping of sparql onto conventional sql, 2010. URL http://www.w3.org/2008/07/MappingRules/StemMapping.

Neil Quarmby. Future work in strategic criminal intelligence. *Strategic Thinking in Criminal Intelligence*, pages 129–147, 2004.

Anand Rajaraman. *Mining of Massive Datasets*. 2014. URL http://infolab.stanford.edu/~ullman/mmds/book.pdf.

Jerry Ratcliffe. *Intelligence-Led Policing*. Routledge, 2008.

Eric S. Raymond. What is data munging?, 1996. URL `http://eduunix.ccut.edu.cn/index2/html/oracle/O'Reilly-Perl.For.Oracle.DBAs.eBook-LiB/oracleperl-APP-D-SECT-1.html`.

T.C. Redman. *Data Quality: The Field Guide*. Digital Press, 2001.

David Ruppert. Inconsistency of resampling algorithms for high-breakdown regression estimators and a new algorithm. *Journal of the American Statistical Association*, 97: 148–149, 2002.

Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. A sparql performance benchmark. *ICDE*, pages 222–223, 2009.

Kumar Setty. What is big data and what does it have to do with audit. *ISACA Journal*, 3:1, 2013.

Alan Shugart. Hardrive: Cost of hard drive storage space, September 2012. URL `http://ns1758.ca/winch/winchest.html`.

David Smiley and Eric Pugh. *Apache Solr 3 Enterprise Search Server*. PACKT Publishing, 2011.

Shan Suthaharan. Big data classification: Problems and challenges in network intrusion prediction with machine learning, 2012. URL `http://www.sigmetrics.org/sigmetrics2013/bigdataanalytics/abstracts2013/bdaw2013_submission_4.pdf`.

S. C. A Thomopoulos. Sensor integration and data fusion. *Sensor Fusion II: Human and Machine Strategies*, 1198:179–191, 1989.

S. C. A Thomopoulos. Decision and evidence fusion in sensor integration. *Advances in Control and Dynamic Systems*, 49:339–412, 1991.

Eric Tom. Mathematical models in decision analysis. *Chicago Journal - Shea - The Society of Healthcare Epidemiology of America*, 18:65–69, 1997.

Evangelos Triantaphyllou. *Multi-Criteria Decision Making Methods: A Comparative Study*. Kluwer Academic Publishers, 2002.

US-Army. *Human Intelligence Collector Operations*. Number 2-22.3. Pentagon Library, 2006. URL http://books.google.com.au/books?id=c6mp5QHkJ8YC&pg=PT4&dq=intelligence+source+reliability+a1&lr=&num=50&as_brr=3&cd=3&redir_esc=y#v=onepage&q&f=false.

W3C. Testimonials for w3c's semantic web recommendations - rdf and owl, 1 2004. URL http://www.w3.org/2004/01/sws-testimonial.

W3C. Sparql query language for rdf, January 2008a. URL http://www.w3.org/TR/rdf-sparql-query/.

W3C. Sparql protocol for rdf, January 2008b. URL http://www.w3.org/TR/rdf-sparql-protocol/.

W3C. Sparql 1.1 property paths, 2010. URL http://www.w3.org/2009/sparql/docs/property-paths/Overview.xml.

W3C. Owl 2 web ontology language structural specification and functional-style syntax, December 2012. URL http://www.w3.org/TR/owl2-syntax/.

W3C. Resource description framework (rdf), 2 2014. URL http://www.w3.org/RDF/.

Qiang Yang and Xindong Wu. 10 challenging problems in data mining research. *International Journal of Information Technology & Decision Making*, 5:597–604, 2006. URL http://cs.uvm.edu/~icdm/10Problems/10Problems-06.pdf.

Xingquan Zhu, Xindong Wu, and Qijun Chen. Eliminating class noise in large datasets, 2003. URL http://www.aaai.org/Papers/ICML/2003/ICML03-119.pdf.

Appendix A

Supporting Material

A.1 Publications

Author(s)	Title	Conference	Date
Neil Brittliff and Dharmendra Sharma	The Schema Last Approach to Data Fusion	AusDM	27/11/2014
Neil Brittliff and Dharmendra Sharma	A Triple Store Implementation to support Tabular Data	AusDM	27/11/2014

A.2 Letter of Appreciation

To Whom It May Concern,

Letter of Appreciation - Mr Neil Brittliff

The Australian Crime Commission (ACC) would like to thank Mr Neil Brittliff for his excellent work on the Fusion Program. This program involved significant research and implementation of our main analysis platform, including the development of ground-breaking algorithms for problems confronted by our Intelligence Agency. Mr Briffliff's work was instrumental for collation, process, analysis and knowledge discovery from data received by the ACC.

Mr Brittliff's efforts resulted in the introduction and successful implementation of **Eland** which employs his **'schema last' computational model**. This has meant the ACC has eliminated the need to 'cleanse' data and has allowed analysts to upload and model data without the need for specialist skills.

Mr Brittliff also developed advanced index strategies and mechanisms that have proven to be invaluable. This has allowed analysts to quickly search and retrieve records from Fusion Data Holdings.

Mr Brittliff also helped develop the ACC's bulk matching capability which allows the ACC to process large lists (of names, addresses, and so on) and match these with records held within Fusion Data Holdings.

The Fusion capability and the ACC as a whole has greatly benefited from Mr Brittliff's research and implementation work. The Agency's capability to deal with the wide variety of data sets received has been significantly enhanced

Issues experienced by the ACC motivated Mr Briffliff's research which became the topic for his PhD. The ACC is delighted to commend the outcomes.

Yours sincerely

Dr Maria Milosavljevic
Chief Information Officer
Australian Crime Commission

Dr John Moss
National Manager, Intelligence
Australian Crime Commission

14 November, 2014

ACC HEADQUARTERS
44 Mort Street Canberra ACT 2601
GPO Box 1936 Canberra ACT 2601
Tel: (02) 6243 6613 | Fax: (02) 6243 6679 | Internet: www.crimecommission.gov.au

Appendix B

Media Release - ACC Fusion Capability

Paul Jevtovic APM
Executive Director, Operations

The Australian Crime Commission is Australia's national criminal intelligence and investigative agency with a focus solely on issues of serious and organised crime. A key hallmark of the Australian Crime Commission is its Board, which comprises the heads of 14 agencies from state, territory and Commonwealth law enforcement, regulatory and national security and is the most powerful law enforcement and national security body in the country. Another key hallmark of the ACC is its specialist Royal Commission-style coercive powers and its ability to conduct special investigations and special operations where conventional law enforcement methods are unable or unlikely to be effective. We are also in the business of collecting and analysing criminal intelligence and data, and where possible, sharing that resulting information. To truly have impact against serious and organised crime, we must first discover and understand the national and international picture of its networks, methodologies, and the full range of vulnerabilities it exploits. We must then translate this into effective responses. To do this, we need rich, contemporary, and comprehensive criminal intelligence. Building this intelligence

picture and identifying organised crime trends and weaknesses is the Crime Commission's core business. Much of the Commission's intelligence is housed in our National Criminal Intelligence Fusion Capability, which brings together subject matter experts, analysts, technology and big data to identify previously unknown criminal entities, criminal methodologies, and patterns of crime. For example our Fusion capability identifies the threats and vulnerabilities through the use of data. It brings together, monitors and analyses data and information from Customs, other law enforcement and Government agencies and industry to build an intelligence picture of serious and organised crime in Australia.

Paul Jevtovic APM **- September 2014**

Appendix C

The Australian Criminal Intelligence Model

ACIM Phase	Description	Collaboration activities
Plan, Prioritize, Direct	**Establish the intelligence requirements, plan intelligence activities and direct resources according to priorities.** The planning, prioritization and direction phase, sets the stage for the Intelligence Cycle and provides the foundation from which all Intelligence Cycle activities are launched. Priorities are established when the threat and risk levels have been determined which in turn facilitates the planning and direction regarding the deployment and utilization of resources. The direction component can precede the planning where a requirement for a specific product is made, such as a full report, or tactical assessment to meet a particular business need.	Collaborative leadership. Use of online forums or discussions to share ideas, identify risks and opportunities and priorities in terms of strategic drivers.

Collect & Collate	**Gather the raw data and group related items together to facilitate further processing required to produce the finished product.** Collection involves the identification, location, and recording and storing of information and data. The collation groups together related information to facilitate further processing through the analytical phase of the Intelligence Cycle. This phase includes the extraction and collation of known information and intelligence, the identification of gaps in the information currently held and the preparation of a collection plan.	Collaboration activities include: assistance in locating and gathering information, sharing methodologies and facilitation e.g. locating information in systems not available to the searcher real-time information contributions
Analyze & Produce	**Integrate, validate, analyse, and prepare the processed information for inclusion in the finished product.** The analysis and production phase requires highly trained and specialist personnel (analysts) to give meaning to the processed information and to make inferences, conclusions and recommendations. The analyst role is to synthesize the processed information into a finished, actionable intelligence product.	Collaboration activities include Encourage and support communities of practice to develop professional practice and share skills. Share and comment on analysis
Report & Disseminate	**Delivery of the finished product (Dissemination) or an account or statement describing in detail an event or situation (Report) to the customer and to others as applicable.** Formal products, reports and ancillary information including collected data are only useful once it is made available to others. This may be in the form of a formal dissemination of a product; a report which provides a written account of a situation or event; or ancillary information exposing the support data to others for additional intelligence work.	Single source of intelligence. Collaboration activities include sharing and discussion of reports and ancillary information.

Evaluate & Review	**Continually acquire feedback during the Intelligence Cycle and evaluate that feedback to refine each individual step and the cycle as a whole.** Constant evaluation and feedback from the 'customer' is extremely important to enabling those involved in the Intelligence Cycle to adjust and refine their activities and analysis in a continuous improvement cycle to better meet changing and evolving information needs.	Collaborative activities include: Collect evaluative comments and feedback into one source by running discussion forums or blogs on issues.