Constructing Data Marts from Web Sources Using a Graph Common Model

Michael Scriney

B.Sc. Computer Applications (Hons)

A Dissertation submitted in fulfilment of the

requirements for the award of

Doctor of Philosophy (Ph.D.)

to

Dublin City University

Faculty of Engineering and Computing, School of Computing

Supervisor: Mark Roantree

June 2018

Declaration

Signed: ____

ID No.: 59592997

Date: 6th June 2018

List of Publications

1. Michael Scriney and Mark Roantree. Efficient cube construction for smart city data. In Proceedings of the Workshops of the EDBT/ICDT 2016 Joint Conference, EDBT/ICDT Workshops 2016, Bordeaux, France, March 15,2016., 2016.

2. Michael Scriney, Martin F. OConnor, and Mark Roantree. Generating cubes from smart city web data. In Proceedings of the Australasian Computer Science Week Multiconference, ACSW 2017, Geelong, Australia, January 31 - February 3, 2017, pages 49:149:8, 2017.

3. Michael Scriney, Martin F. OConnor, and Mark Roantree. Integrating online data for smart city data marts. In Data Analytics - 31st British International Conference on Databases, BICOD 2017, London, UK, July 10-12, 2017,Proceedings, pages 2335, 2017.

4. Michael Scriney, Suzanne McCarthy, Andrew McCarren, Paolo Cappellari,and Mark Roantree. Automating data mart construction from semi-structured data sources. 2018. To appear in the Computer Journal, Oxford University Press, 2018.

Acknowledgements

I would like to thank the Insight Centre for Data Analytics for funding my research under grant no. SFI/12/RC/2289.
I would also like to thank my supervisor Dr. Mark Roantree for his support and guidance. His hard work and dedication have been a constant source of inspiration.
I would also like to thank my colleagues who were always willing to help or listen to any problems I encountered during my research.
My family and friends also provided me with their support throughout my research and for that, I can never thank them enough.
Finally, I would like to thank Yvette for her continued love and support.

Abstract

Michael Scriney

Constructing Data Marts from Web Sources Using a Graph Common Model

At a time when humans and devices are generating more information than ever, activities such as data mining and machine learning become crucial. These activities enable us to understand and interpret the information we have and predict, or better prepare ourselves for, future events. However, activities such as data mining cannot be performed without a layer of data management to clean, integrate, process and make available the necessary datasets. To that extent, large and costly data flow processes such as Extract-Transform-Load are necessary to extract from disparate information sources to generate ready-for-analyses datasets. These datasets are generally in the form of multi-dimensional cubes from which different data views can be extracted for the purpose of different analyses. The process of creating a multi-dimensional cube from integrated data sources is significant. In this research, we present a methodology to generate these cubes automatically or in some cases, close to automatic, requiring very little user interaction. A construct called a StarGraph acts as a canonical model for our system, to which imported data sources are transformed. An ontology-driven process controls the integration of StarGraph schemas and simple OLAP style functions generate the cubes or datasets. An extensive evaluation is carried out using a large number of agri data sources with user-defined case studies to identify sources for integration and the types of analyses required for the final data cubes.

Table of Contents

List of Figures

Preface

This thesis presents a set of data models and corresponding processes which facilitate the construction of data marts from unseen web data in a semi-automatic fashion. In chapter one, we present an overview of database, ETL and data warehousing technologies, with their current limitations as motivation for this research. In chapter two, we examine the state of the art in detail. Related research across a number of sub-domains are provided which examine the state across all research threads in this thesis. In chapter three, we present the overall system architecture required to build a semi-automatic ETL process for unseen web data. This architecture was first published as part of a smart cities project in [64].

We then examine the main components of the system where chapter 4 outlines the common data model for the system called a StarGraph. This model consists of an annotated graph detailing the multidimensional components of a data source which can be used to construct a data mart and was published in [62]. An extension of this model which incorporated an integration process using a `Constellation` is presented in chapter 5. The model structure and construction methodology were published in [63].

Chapter 6 presents optimisations which improve the construction time and structure of data marts constructed from unseen web data. In chapter 7, we present our evaluation which examines semi-automatic data warehouse construction to fully automatic; we compare and contrast the benefits of each approach across three case studies. This detailed validation of our methodology was published in [61].

Chapter 1

Introduction

This dissertation is focused on the varied and plentiful data streams that originate on the web, are often rich in content and contain information that is not captured in an organisation's enterprise system. We will provide a framework for managing these streams, through an automated extraction and integration process, with the primary goal of delivering analysis-ready data to the end user. This dissertation will show that the transformation and integration of data streams is both difficult and costly and thus, we must provide a system to manage all layers of the process. This introductory chapter, begins with an overview of the benefits of business intelligence and data mining outlined in section 1.1 before a discussion on the importance of data warehousing in section 1.2. In section 1.3, we will highlight the particular issues faced when managing stream data before presenting the hypothesis and goals of this research in section 1.4.

1.1 Business Intelligence and the Importance of Data

The term Business Intelligence (BI) generally refers to methods, technologies and applications which together manage the harvesting, integration, analysis, and presentation of business information [66]. Its purpose is to support improved and effective business decision making. The terms *business intelligence* and *data analytics* are closely linked and often interchanged. One can consider business intelligence as the over-arching term for data usage in a predictive setting. Whereas data analytics is focused on the means of making these predictions. However, both refer to the usage of data to make decisions or predictions. Moreover, it has been cited many times that *"data is the new oil"*. In 2016, more data was created than in the previous 5,000 years and with the proliferation of data from sensors and wearables, volumes of data generated will continue to increase. However, less than 0.5% of this data is analysed for usage in domains such as business intelligence [72].

Accessibility and integration are two primary reasons for this extraordinarily low statistic [58]. These problems lie at the data management or engineering layer and *not* at the business intelligence layer. In a recent Gartner report [53], some important issues were highlighted with respect to the effort and time required in preparing data for business intelligence. One of these issues related to the preparation of data states "Data preparation is one of most difficult and time-consuming challenges facing business users of BI and advanced analytics platforms". In addition, this report suggests that new capabilities are emerging which address the extract, transform and load (ETL) functions, enabling users prepare, integrate, model and enrich data for analysis. This emergence of new capabilities is only at infancy with an urgent need for new methods and technologies to, in our opinion, ease the burden of data preparation and reduce the time taken to present data for analysis. The research presented in this dissertation aims to directly address this issue of moving data from its simplest, rawest format and preparing it for data analysts or BI domain experts.

Throughout this thesis, we will use the terms *data mining*, *data analytics* or simply *analytics* to mean similar activities: the usage of data to make predictions about the future. Data mining *algorithms* allow analysts to glean new insights from their data which can be used to enhance decision making. At a low level, analytics are merely descriptive statistics. For example, the total sales per quarter of an enterprise, or the average amount spent per customer. However, data mining operations can be used to derive more complex information from within a dataset. Algorithms such as *k*-means [30] provide a means of grouping data based on commonality which can then be used to provide *classifications* for unseen data. In other words, if you construct a new dataset (perhaps using sensor devices) or harvest a series of web streams, none of the data instances have a *classification*. Clustering allows us to group all of the data into a small number of clusters and after an analysis of what properties make the clusters similar, classify the instances that belong to each cluster. Clustering also provides a means of outlier analysis, where the data miner can detect those data instances that are outside available classifications [34].

In another form of application, association rule mining such as Apriori [29] can correlate which items are frequently purchased together, providing a decision maker with useful information about customer behaviour. This enables accurate predictions about future buying patterns. In this application, rules are generated regarding the simultaneous occurrence of two itemsets. Considerable research has been invested into determining *strong rules* [2], those that have higher thresholds for occurrence, and functions such as `Lift` and `Leverage` [29] to promote more interesting rules. These algorithms can then be used to enhance an enterprise with predictive capabilities and thus, give decision makers the capacity to plan for the future of the enterprise.

However, all of these approaches assume that the data has been collected, cleaned and integrated within a repository prior to the algorithms execution. The cleaning of data and determining the required sanity checks for data is itself no mean feat. There are numerous ways in which data may prove invalid. Missing values are presented as the most common across domains, however, within a domain more problems relating to the cleanliness of data may present themselves. In order to overcome these difficulties a domain expert is required in order to provide the domain specific sanity checks for the data. This necessitates the development of a "data dictionary" which can be used as a reference for all incoming data into an ETL system in order to ensure it is valid prior to loading within the data warehouse.

It is noteworthy that a paper dating as far back as 1958, presented three systems that were needed to deliver decision making or analytics to the end user: the auto-abstraction of documents; the auto-encoding of documents; and the automatic creation and updating of action-point profiles [45]. It was the first presentation of a business intelligence system. Its focus on an automated system for harvesting or abstraction of data and for encoding, which in today's terms means data transforming, remains in place today. Moreover action points refer to the separate stages in information processing. In effect, this laid the basis for we now refer to as the Extract-Transform-Load infrastructure, which underpins the data warehouse.

1.2 Data Warehousing and ETL

Most enterprises use database technologies in order to store data required for the running of the enterprise. These databases are usually structured in a normalised relational form [21], consisting of related tables. Each table representing an entity with entities relating to each other through the use of primary and foreign key relations. For example, Figure 1.1 presents a traditional sales database which keeps track of products bought in stores, and orders for products from suppliers. This model works well for the day-to-day operations of a business but it poses practical problems for an analyst wishing to provide reports or execute data mining operations. For example, in order to obtain the total price of a sale, a query must be executed per order to calculate the sum of all `sale_products` per sale.

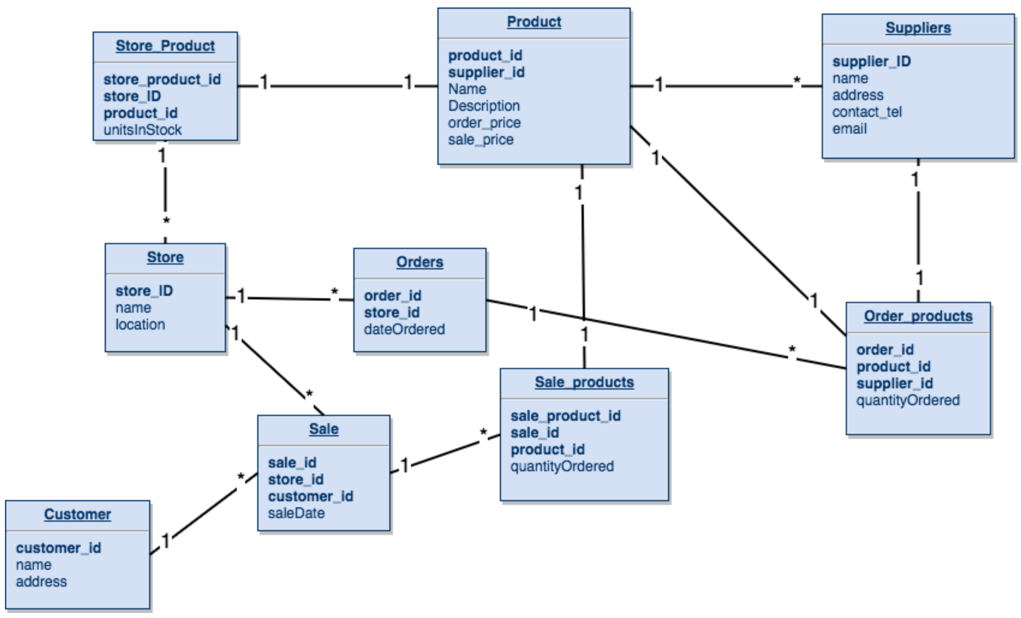


Figure 1.1: Example sales and orders database

As the database grows in size through normal day to day operations, the time taken to extract this required data increases, leading to a decrease in the responsive capabilities of the enterprise. This provides a motivation to archive data into a separate repository. However, the execution of ad-hoc predictive software or data mining operations can consume processing resources. This provides a more important reason to keep operational and analytical operations separate. However, most compelling of all, is the need to capture enterprise data from all available operational systems, both internal and external to the organisation. The integration of these sources, which requires resolving structural and semantic heterogeneities, led to the development of the storage mechanism known as the *data warehouse* and a process known as *Extract-Transform-Load* (ETL) [38].

1.2.1 The Data Warehouse

A data warehouse serves as a single repository for all analytic purposes across all applications of an enterprise. The differences between a traditional database and a data warehouse are architectural. The same underlying DBMS (DataBase Management System) may be employed but the structure and relationships of entities are different. In the author's original work on data warehousing [37], the following definition was presented: a data warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process. *Subject oriented* means it is *not* application oriented and thus, we must provide for user requirements which focus on the concepts captured in the warehouse. We will return to this point through the dissertation as we focus on the construction of data marts and user requirements. The *non-volatility* aspect will be captured by our system in the form of a Data Lake [17] and the *time-variant* aspect will be present in all of our case studies. The *integration* property will form a big part of our research as we seek to integrate multiple sources for the user.

Where database systems are large, *views*, representing a subset of the overall schema, are defined with a single user group in mind. Similarly, a Data Warehouse is too large and spreads across too many domains and for a individual analyst. The majority of a Data Warehouse is of little interest for the needs of a specific decision maker. The data warehouse equivalent of a view is a *data mart*, sometimes called a *Cube* an n-Cube where n specifies the dimensionality of the cube.

A Data Mart is generally centred around a single user requirement (for example analysis of sales, by product, by region, by month) and in an enterprise environment, these marts are provided to different departments depending on their needs. Conceptually, each department is considered the owner of their own data mart, which fits into the wider data warehouse. However, a data mart is generally formed through integrating data from multiple sources. A definition of system integration [32] is a process that builds applications that are adaptable to business requirements while allowing underlying systems to operate in an autonomous fashion. This research will focus heavily on integration systems which operate in an environment where the data mart is formed from sources over which it may have no control.

1.2.2 Warehouse Models

Each data mart corresponds to a set of interests for the analyst, with each individual data mart within a data warehouse being structured as either a `Star`, `Constellation` or `Snowflake` schema. We will briefly describe these different schemas in order to be clear on what each represents in our research. All models have two main entities: the fact and the dimension. The difference between these three models is the configuration of dimensions and facts.

A *fact* represents a single item of interest to an analyst. This is composed of a metric of interest (e.g. sales) along a series of analytical axes called *dimensions*.

Recall the database schema shown in Figure 1.1. If an analyst wished to examine total sales, a Star Schema such as that presented in Figure 1.2 would be used. In Figure 1.2, the dimensions used to represent the sale are `Store`, `Product`, `Customer` and `Date`. These all hold a one-to-many relationship to the fact table `fact-scale`. This table contains two more attributes, the quantity of a product sold, and the total price. This schema model is known as a Star Schema due to their representation in diagrams being similar to a star, with a single fact in the centre (which can have one or more measures) and dimensions as the points of the star. Using this schema, queries such as total sales on a given day can be computed a lot easier than the schema shown in Figure 1.1.

However, this structure requires the use of joins in order to consolidate data across dimensions and measures. As the size of the data warehouse increases this poses a

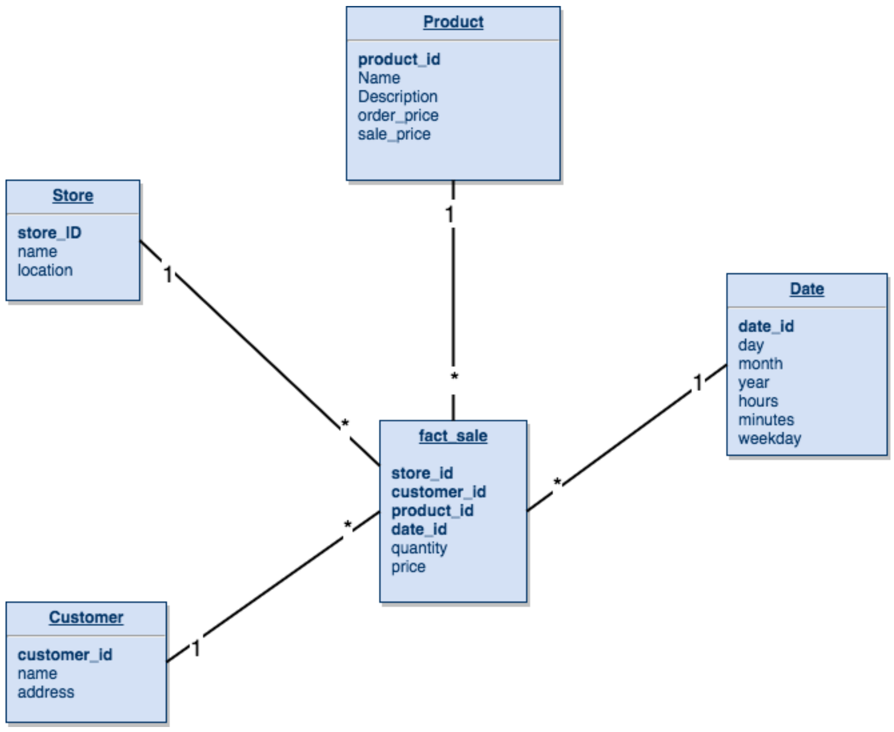


Figure 1.2: Star Schema with single fact and multiple (4) dimensions.

bottleneck for query response times, due to the number of join operations required. However, a data mart may contain multiple facts which share a set of conformed dimensions [42], and this type of schema is known as a *Constellation schema*. A Constellation schema consisting of the facts `sale` and `order` can be seen in Figure 1.3. In this figure, the two facts `fact_sale` and `fact_order` share the dimensions `Store`, `Date` and `Product`. In other words, there is an integrated feature to this type of schema model. We will exploit this relationship between a constellation schema and integration later in this dissertation.

Finally in order to aid analysis, dimensions may be hierarchical in nature, providing varying degrees of granularity to a fact. Such a schema in this case is called a `Snowflake Schema` [44]. It is a Constellation Schema with *normalised* dimensions. In Figure 1.4, the `Date` Constellation shown in Figure 1.3 has been changed into a hierarchical structure, surmising the `Month`, `Quarter` and `Year` for a given `Date`.

Data Cubes The goal of these schemas is to provide fast computation of analytical queries. The most common query executed on a `fact table` is the construction

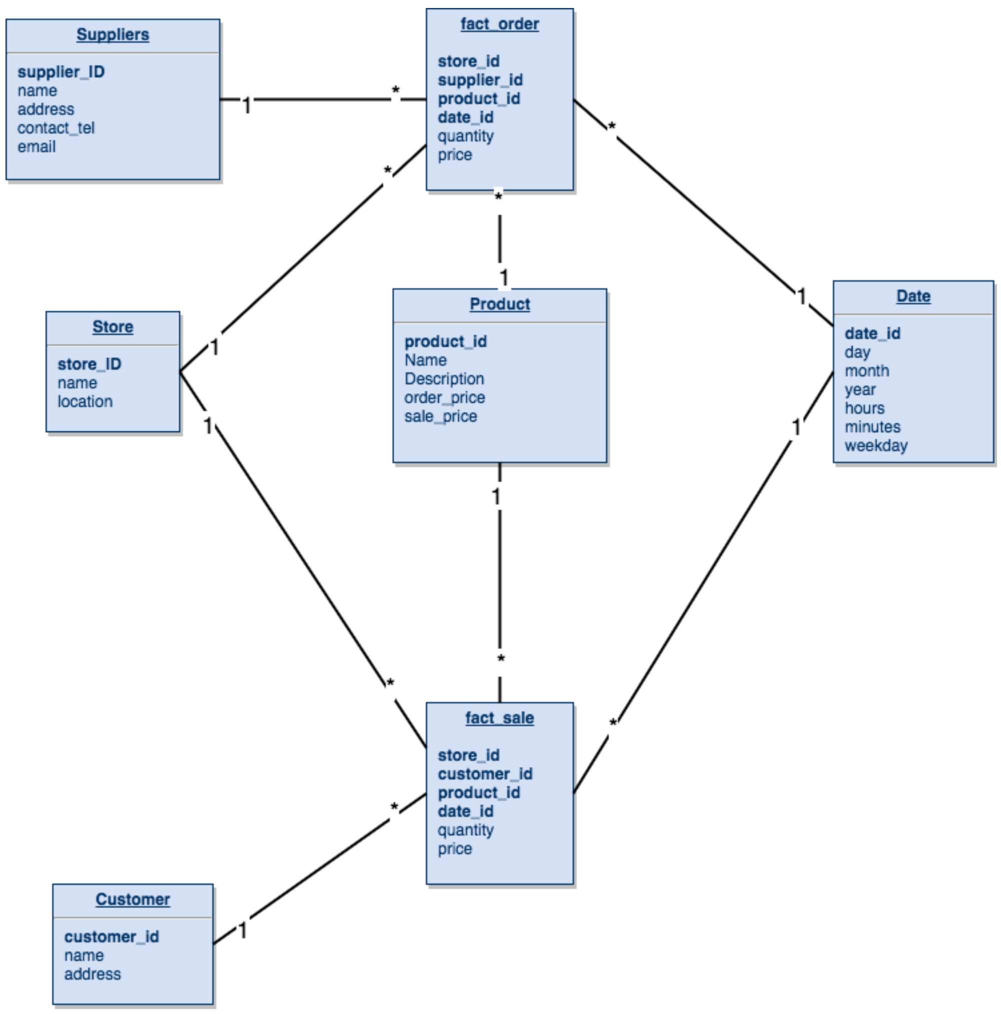


Figure 1.3: Sample Constellation with facts and dimensions

of a data cube [28]. A data cube aggregates a measure across a series of analysis dimensions. A data cube is composed of cells and dimensions. Each cell representing an aggregate value of a measure for a series of dimensions. Figure 1.5 presents a diagram of a data cube constructed from the dimensions `Store`, `Product` and `Customer`. Thus, a cube could be considered as a `SELECT ALL` query from a data mart and a clear correlation exists between both constructs.

Data cubes are very large and a lot of time is required to construct a full cube. This has led to numerous approaches in optimisation, from cube approximation [75] to distributed methodologies [76]. In addition, there are different types of data cubes, such as Iceberg cubes [11] and Dwarf cubes [68] which provide optimisations for

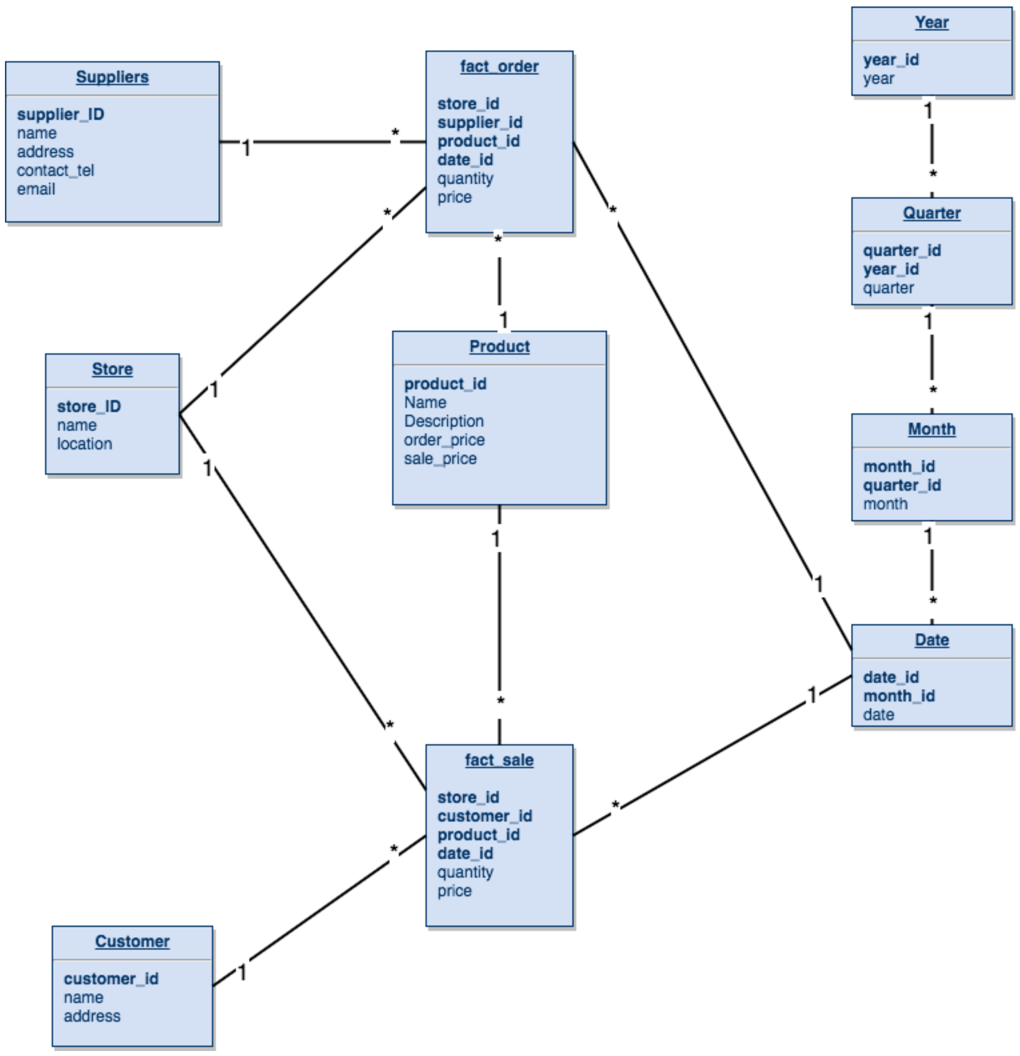


Figure 1.4: Sample Snowflake with Normalised Date Dimension.

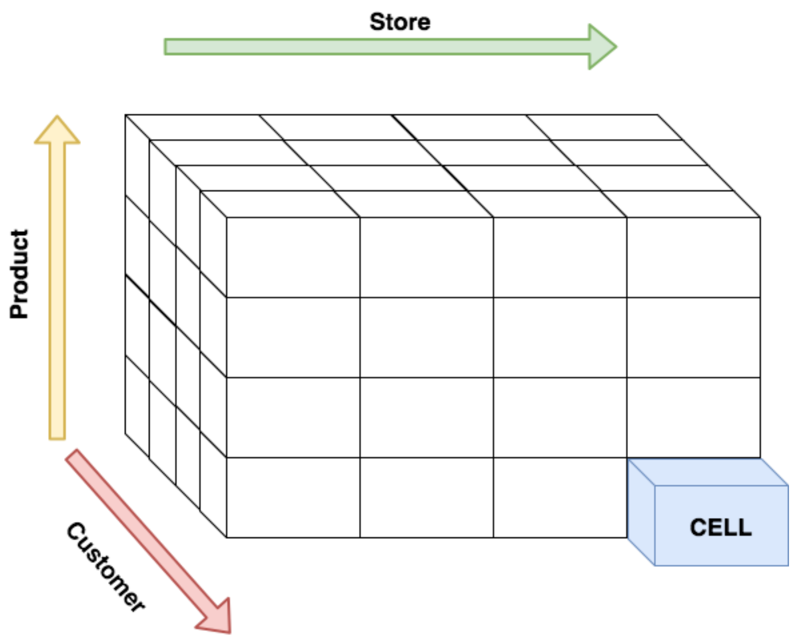


Figure 1.5: Example Data Cube

specific use cases.

1.2.3 OLAP

On-Line Analytical Processing (OLAP) [22] provides a means of posing multidimensional queries and requires the use of an OLAP server which loads the cube. There are many flavours of OLAP, from MOLAP (Multidimensional-OLAP), ROLAP (Relational-OLAP) and HOLAP (Hybrid-OLAP). The differences lie in implementation and storage, with MOLAP opting to use multidimensional arrays, ROLAP resting on top of a traditional RDBMS and HOLAP using a combination of both.

OLAP provides a set of functions which provide analysts with a means of querying a data cube. The main types of queries are:

- `ROLLUP` reduces the granularity of a dimension within a cube (for example, $Date \rightarrow Month$).
- DRILL DOWN performs the opposite to ROLLUP and adds a dimension.
- `SLICE` extracts cells from the cubs corresponding to particular values within

a dimension (e.g. $Month = Jan$) while `DICE` provides the same functionality for a series of dimensions (e.g. Month=Jan AND Region=Europe).

- PIVOT allows an analyst to pivot the cube, re-arranging the way in which dimensions are displayed for the cube.

1.2.4 The Extract-Transform-Load Architecture

Constructing a Data Warehouse is a difficult task. It involves enterprise stakeholders, developers, designers, users and domain experts, with numerous factors influencing the success of the project [77]. The first step in designing a data warehouse lies in the identification of the data at hand [42]. In conjunction with end-user requirements, these form the basis of the data warehouse, where each fact and dimension is specified and marts constructed.

The next step is to construct a process which extracts the required data from its source, performs any necessary transformations and integrations and stores the data in the data warehouse. These processes are called ETL (Extract-Transform-Load) processes. The construction and deployment of the overall process consumes most of a data warehouse's development time and budget [24, 36].

The use of domain experts, people who understand the data, is necessary for the construction of this process in order to influence the application designers on how data transformations should occur. The ETL process populates the data warehouse and it is the only process that may write to this repository.

1.3 Mining Data Streams

Traditionally, there is a time delay between data used for the day-to-day operations of a business and data used for analysis. This is because the ETL process to populate the data warehouse is run on a batch schedule periodically to populate the warehouse for analysis [16]. However, this can lead to crucial information and events becoming apparent to analysts *after* the knowledge was available for important decision making as well as the time taken for ETL population.

The need for real time (or near real time) BI has led to the development of `active data warehousing` [54]. Active data warehouses attempt to address this lag between data generation and analysis by improving the ETL process to reduce the time taken to populate the data warehouse. Active data warehouses necessitate a continuous flow of data for population such as a stream [41] and an efficient ETL process.

However, the presence of *data streams* pose additional challenges in addition to time-base responsiveness. Data streams are continuous collections of data which arrive over time [26]. Data streams are used to publish high velocity data. Numerous domains utilise streams, such as the Internet of Things, the Sensor Web [71] and Smart Cities. The continuous granular nature of some streams pose additional problems to the ETL process, `Edge computing` [60] seeks to address these issues by providing a means of pre-aggregating and cleansing data arriving from continuous streams. In addition, some streams are available publicly on the web which are updated at a set interval, although these web data streams are not as high velocity as sensor data, they may still pose problems to an ETL process depending on the size of the data obtained per stream update and the frequency of the streams update interval.

Traditional ETL processes were designed with the aim of reconciling various relational databases and document repositories spread across the departments of an enterprise. As all of this data was under the control of the enterprise, designers and developers had guarantees regarding the cleanliness of data, the update frequencies and the structure of this information.

However, these guarantees also expose the main issue underlying traditional ETL methodologies, that they are *resistant* to change. Within the context of a highly regulated enterprise environment, where the data is known, structured and well understood, this is not a problem. However, with the rise of the internet and its technologies, valuable data is increasingly accessible online in the form of streams. These streams may enhance data residing in a traditional DBMS, or a collection of these streams may form a data mart of their own. However, incorporating these streams poses a number of issues. As these data sources are outside the control

of an organisation, a domain expert is required in order to understand this data. Additionally, designers must manually investigate the data to determine extraction methods. Finally, a mechanism to integrate the data and construct the ETL pipeline must be developed.
As this data is out of the control of the organisation, the assumptions made during the construction of a traditional ETL process no longer hold. The structure of the data and its format may change, which subsequently required a re-engineering of the entire ETL process to accommodate these changes. The data sources on the web may change [47], or disappear while new sources of interest may appear in future.

1.3.1 Problem Statement

The issues involved in web data management have been presented as far back as [33], where a flexible approach was advocated, which suggested both a bottom-up and top-down design in such an environment. In a similar fashion with data warehouses, it has become evident that with all of these issues described above, the ETL process and data warehouse must be re-designed for web data streams. When dealing with web data, a new warehouse construction methodology is required which does not incur the technical debt posed by traditional methodologies.
While issues surrounding the responsiveness of a Stream Processing System (SPS) are dependant on user-requirements and the update intervals of the stream, an SPS must be able to process an instance of a stream prior to the stream updating. A flexible, faster approach which is more lightweight, could solve the issues to creating data marts from streams. The issues can be highlighted as:

- Traditional ETL processes consume too much time and expense, requiring the involvement of multiple designers, developers and domain experts. Clearly, this is impractical in the rapidly changing environment of stream data.

- A traditional ETL system is resistant to change, leading to any extensions requiring a re-engineering of ETL processes, further adding to the time and expense involved. These issues are further compounded when dealing with web data outside the control of the organisation, as they may change over time, sources may disappear, or may change their location on the web.

- It is not impossible to build a traditional ETL process to construct a warehouse from web data. However such a process would be ongoing, requiring the continual involvement of designers, developers and domain experts to react and re-engineer the process in the face of changing data.
- Web data and data obtained from streams require cleanliness and sanity checks prior to loading into the Data Warehouse. This problem is further compounded with the fact that these sources are outside the control of the enterprise and are subject to change at a moments notice. Once again, the issues of "dirty data" require the continuous involvement of a domain expert and developer to continuously update an ETL process as issues present themselves.

Ultimately the issues surrounding traditional ETL pipelines can be summed up by time and expense. The initial time and expense required to construct the Data Warehouse and subsequent ETL process coupled with the time and expense required in order to overcome the technical debt incurred when making extensions to this process.

1.4 Hypothesis, Aims and Goals

The construction of a data warehouse along traditional lines is costly and time consuming. Furthermore, updates are always on a batched basis and take time to process. If we are to incorporate web streams into a data warehouse, a new approach is required to construct and manage warehouses from streams in a fast and efficient manner. Ideally, such a process would be fully or semi-automatic and where possible, reduce or eliminate the burden on users. Such a system requires the ability to capture unseen data sources, determine a suitable mechanism for integration and construction of an ETL pipeline, with minimal user-interaction. In specific terms, this means that the construction of the data mart is built from new or changing web streams in an automated fashion. In order to achieve this goal, a process is required to extract multidimensional components from streams, determine commonality across streams and combine them.

The hypothesis presented in this research is that if a new data model can be constructed which understands the multidimensional requirements of a data mart (or star schema), and can feature extract from data streams. Subsequently it is possible to deliver data marts from stream data. Such a model would provide an end user with fast access to the most recent data. There are numerous benefits to providing users with up to the minute data; the users are given the ability to react to events as they are happening or in some cases old data may not prove useful (e.g. real time traffic routing).

A number of research questions can now be posed which serve to highlight the major goals of this research.

The main goal in this research and thus, the main research question is: could it be possible to construct a data mart structure in an automated fashion, by analysing the data captured inside the streams? We can break this overall goal into a manageable set of research questions.

- The first step is in the specification of a canonical model for the system. Given the genericity of graph structures [67] and their ability to represent data. Can graph structures be used as a common data model to overcome the differences posed by integrating web data sources? What type of data model structure can be used as a mechanism to identify the facts, measures and dimensions within a data source to construct a data mart?

- Once a common data model has been specified, the next stage would be to identify items of interest within a data source. As the end goal of this system is to construct data marts, these items of interest are namely: facts, dimensions and measures. Is our data model semantically rich enough to interpret multidimensional constructs?

- With facts, dimensions and measures identified, the next stage determines commonality and an integration strategy to combine these sources. Is it possible to determine a suitable integration strategy which combines graphs structures representing web data to produce an integrated data mart?

- Finally, once the schema and integration strategy have been determined, the next stage would be to provide an ETL pipeline which can automatically translate this data from its source format into the Data warehouse. Is it possible to automatically construct an ETL pipeline which can be used to populate a Data Warehouse from web data?

1.5 Contribution and Thesis Structure

In this chapter, we presented the background to our work, motivated the area of research in which our work is based, and presented our hypothesis. In chapter 2, we examine the state of the art in data warehouse construction with a specific interest on the management of streaming data. This addresses the open research questions to be presented in this dissertation. In chapter 3, we present our overall system architecture required in order to construct data marts from web data. The novelty to our research begins in this chapter with an entirely new ETL approach which is designed specifically to manage new and changing data sources. This architecture was presented in a publication by the authors in [64] in the smart city domain. In this dissertation we concentrate on the Agri domain, using chapter 3 to introduce real-world case studies from the agri (agriculture) industry.

In chapter 4, we present a new data model (the StarGraph) and a methodology designed to construct a data mart automatically from a single data source, and was first presented by the authors in [62]. This forms a major part of the contribution to our research as this data model captures facts, dimensions and measures and together with the model functionality can analyse web sources to extract these multidimensional constructs. The graph is called a StarGraph for two reasons: it is a representation of the facts, dimensions and measures of a data source and it is a graph representation of a Star (Constellation or Snowflake) Schema. In chapter 5, we provide a detailed discussion on how multiple StarGraphs may be integrated to produce a Constellation in a semi-automatic manner with the aid of a lightweight ontology. We presented our work on an integrated StarGraph in [63] and subsequently demonstrated its effectiveness in conjunction with a lightweight ontology in [61]. This extended the StarGraph functionality to introduce an integration strategy which creates data marts from multiple sources. In chapter 6, we present some optimisations which remove redundancies within a StarGraph or Constellation to improve the speed of materialising data marts. In chapter 7, we present our evaluation and discuss the benefits and drawbacks of our approach compared to traditional ETL processes. Finally, in chapter 8, we present our conclusions and discuss potential areas for extending our research.

Chapter 2

Related Research

The construction and updating of data warehouses is costly both financially and in terms of the time and human effort involved. In this thesis, we are proposing a more automated approach not only to address the cost of these systems but in recognition of the fact that online streams are wide ranging, heterogeneous and prone to change. For this reason, a new approach to Extract-Transform-Load (ETL) is necessary and as such, requires a combination of methodologies drawn from traditional data warehousing techniques, warehousing web data and the use of ontologies in data warehouse design. For this reason, the literature review is split into a number of sections. In section 2.1, we begin by examining different approaches to warehouse construction including research into ontology based ETL; approaches to warehousing data streams are discussed in section 2.2, finally section 2.3 summarises the state of the art and identifies outstanding issues within current research.

2.1 ETL and Data Warehouse Construction

The aim of this section is to discuss and analyse research into warehousing with a focus on the Extract-Transform-Load (ETL) process. Ontologies are seeing increasing use in Enterprise environments, and as such have seen extensive use within ETL applications. Their ability to capture information centred around a domain and provide a means of relating these abstract concepts to real data has proven invaluable. The works presented all propose ETL frameworks with an ontology as the main

focus of the application. This can take many forms, from a single global ontology, to more complex approaches where each source contains a local ontology, coupled with a global ontology, a data warehouse ontology and a requirements ontology.
In [9], the authors present a system to automatically facilitate extraction and population in order to integrate additional data sources to a pre-existing data warehouse. Their system constructs schemas for each data source required in the ETL process through a series of wrappers. The process then annotates the source with terms from the system's ontology. Once this stage has been completed, the next step involves examining the relationships found within a document in order to create a thesaurus of relationships of terms found within the ontology. The next phase in the process is the generation of clusters. This process utilises the thesaurus in order to construct clusters of common terms. These serve to link attributes between data sources. Finally, the addition of transformation functions and mapping rules provide an automated means of running the extract phase of the ETL process.
The similarities between our research objectives and this research lie in the annotation or enrichment of the original source. However, in their case, it is ontology driven. A further and significant difference lies in the approach: they assume a pre-constructed warehouse whereas our agile approach does not. Their approach to using numerous semantic technologies to integrate data with the pre-existing warehouse data only works if there exists a pre-defined data warehouse from which a global schema can serve as a reference point.
Berro et al. present a graph based ETL process where the assumption is that source data is a series of spreadsheets [10]. Their system uses an ontology to model the representation of data within a spreadsheet. When a source is added to the system, it is analysed using the ontology and annotated appropriately. As with many approaches (including our own), the authors place an emphasis on spatio-temporal datatypes (Date, Geo), naming them `Semantic Annotations`. This process outputs an annotated graph representing system data types. The next phase of the system is to integrate multiple annotated graphs into a single unified graph. The integration approach consists of three phases: the first seeks to construct a similarity matrix between two graphs; the process then uses a set of rules based on maximising these

similarities; the system then constructs the integrated graph.

The similarities between this and our research is again in the requirement to annotate the source data. However, our construction of annotated graphs does not require an underlying ontology to influence the annotation process. Furthermore, we would not restrict source data to exclude tree-type data sources such as XML or JSON. Finally, we advocate the usage of a metamodel in order to influence the integration process which seeks to determine common types between sources in a semi-automatic fashion.

The authors in [59] present a system which constructs ETL processes based on user requirements. The system takes two inputs, the first being a series of required sources for the data warehouse. All sources must be previously annotated with a corresponding OWL ontology describing the structure and semantics of the data source. The second input is a series of business requirements where the authors represent these requirements as structured XML files. The first file represents the functional requirements of the data warehouse, a process similar to the definition of a data warehouse schema. It contains the measures and analysis axis required for the data warehouse. The second file represents non-functional requirements, for example, the age of the data within a respective OLAP view.

There are five main steps in generating an ETL workflow and multidimensional schema from business requirements and data sources. The first step is a verification step, which examines these requirements with respect to the sources shown to determine if the mart is in fact possible, with respect to the sources presented. We would not adopt this approach as it lacks a means of constructing a warehouse with respect to a pre-defined schema derived from requirements.

The second step examines the sources in order to determine other items which are required in order to satisfy the schema. The third step classifies the attributes of data sources with respect to the components within a data warehouse (e.g. facts, dimensions and measures). This process is similar to the dimensions and measures identification process we will present in chapter 4. However, we advocate that this be based on the data sources themselves, rather than this approach which uses ontology sources. Their approach requires prior knowledge of all schemas before importation.

Step four constructs the ETL *process*, driven by the dimensions, measures and facts identified in the previous step. Once this is complete, the final step is the output step, which produces the ETL workflow and corresponding multidimensional schema.

Their system places a heavy emphasis on user requirements, and is dependant on a domain ontology (a global ontology) and an ontology for each source (local ontology). Our approach will not require local ontologies.

An ontology-based ETL system was presented by Zhang et al [79]. The system assumes a global ontology is present which serves as a reference to integrate the individual data sources. There are four phases to the ETL process: `Metadata abstract`, `Ontology mapping`, `Rule reason` and `ETL direction`. The first seeks to build local-ontologies constructed from the metadata of data sources. The second phase examines the local and global ontologies in order to determine how the data source should be integrated. Once the local ontology has been linked with the global ontology, the next phase is to provide a mapping between the two in order to facilitate integration and querying. The final phase is the ETL phase, which uses the mappings generated previously to extract data from source to target.

The 4-step process is similar to what we will design in order to materialise data marts. The differences between the two approaches are as follows: while the authors seek to generate a local ontology in order to link it with the global ontology, our global ontology serves this purpose, with the layer of abstraction and metadata analysis being performed by our proposed integration process. In addition, while the authors refer to the mapping created between the local and global ontologies, they fail to specify at what stage mappings and transformation functions required for the ETL process are generated.

One common theme throughout all processes is the need for an ontology, the specific implementation does not matter, but it does identify that ETL processes with minimal user-engagement require a means of capturing information and abstractions which would be present to a designer creating an ETL workflow manually. In [51], the authors present an ETL methodology using RDF and OWL technologies. The system works by converting all required sources into RDF files which conform to pre-existing ontology-maps.

Similar to our approach, the schema of the data mart is constructed automatically. However, while our approach will construct a data mart by examining the sources required for integration, their system constructs a data mart through a user supplying RDF queries, which are then used to generate the schema.

In [69], Skoutas et al outline how to construct an ETL process using OWL. The first step in this process is the construction of a suitable domain ontology constructed by a designer which captures all semantics relating to the application domain. The next phase involves examining each data source required for the ETL process and annotating the source. Once again, we discover an approach similar to our proposed graph annotation stage. However, here the data store is additionally annotated with an application vocabulary. This vocabulary outlines commonalities between attributes in each data source, a function that we will also use in term and type mapping processes discussed later in chapter 3.

Using the application vocabulary and the annotated data sources, the next step involves generating the application ontology. This ontology is used to represent a specific ETL process, linking the concepts derived from the domain ontology to the vocabularies defined in the data sources. The final step is the design of the ETL process. This process determines what is required form each data source and provides necessary transformations by examining the domain ontology and the application ontology. While the authors examine the use of multiple ontologies to construct an ETL process, our process uses only one ontology to facilitate semantic integration and construct the ETL process.

As part of the same research [70], the authors detail a means of constructing an ETL process for semi-structured *and* structured data using an ontology. Here, they use an XML graph construct called a `datastore graph` to internally represent data sources. In this case, the annotated graph contains mappings to source data for each node in the graph. The first stage in the process converts all required data sources into the `datastore graph` format. The next step utilises a suitable domain ontology in OWL format which will be used to annotate the data stores with semantic information. This ontology is then itself converted into a graph format named an `ontology graph`. The next step in the process seeks to map the `datastore graph` processes to the `ontology graph`.

This differs from our approach in that the authors state that the mapping from `datastore graph` to `ontology graph` is a manual process, with a designer manually linking each node. Our goal is to annotate each node in the StarGraph with its respective term and type in order to provide a level of automation to the linking process.

The authors in [65] describe a means of constructing structured web data warehouses using an ontology. The resulting data warehouse is constructed from first analysing user requirements. Requirements are captured within a requirements model to specify what is required of the data warehouse. These requirements are modelled at the ontological level and are linked to a domain ontology. The system assumes a global domain ontology which links web sources. Once the requirements for the ontology have been captured, the process for constructing the data warehouse and ETL process is as follows: a data warehouse ontology is constructed from the global ontology by examining the user requirements; this ontology represents the structure of the data warehouse and can be edited by a designer in order to provide additional information.

The second phase analyses the requirements with respect to the sources in order to determine that the data warehouse can be constructed. Specifically, the analysis is used to determine required relationships and items required for the data mart to facilitate semantic integration. The next phase annotates the data warehouse ontology with facts, dimensions and measures. We will adopt a similar approach as one of our stated aims is to automatically determine facts, dimensions and measures from source data. However, while this process requires their ontology to be constructed in advance, our process is more automated, extracting the multidimensional data without the need for an ontology.

The final step in the process is the generation of the data warehouse schema from the final ontology. The authors present a system which can create a warehouse from web data with respect to user requirements. However it assumes all sources are already known, and a global ontology linking said sources exist.

Summary All ontology based process rely on the existence of a suitable domain ontology previously created by a designer. In addition there is a common series of steps for all sources, namely the generation of a data warehouse schema from the domain ontology.

While ontologies can be used to resolve differences between data sources and such provide a means on integration, there is limited focus on the use of ontologies for web data outside the control of an application designer.

In addition, all sources presented assume mappings exist within the ontology, and contain an annotation step for identifying data warehouse concepts once a requirement or query has been posed. Our system instead generates mappings prior to any source being in contact with our ontology as they are generated during our annotated graph construction phase, with dimensions, facts and measures being identified without the need for consulting an ontology.

2.2 Data Stream Warehousing

This section examines the state of the art in warehousing web data. Data Warehouses assume a relational structure for data, consisting of facts, dimensions and measures. Such a structure is easy to create if the data is already relational. Traditionally, data warehouses were created by combining data from various relational databases. However, web data can take many forms, from structured, to unstructured data. This poses problems for traditional warehouses, as there is now a need to capture this semi or unstructured data and store it in a data warehouse.

A significant amount of research has been presented on capturing and transforming XML data compared to other semi-structured data sources. The main reason for this is the age of XML compared to newer semi-structured containers such as JSON. The XML standard was first published in 1998 [15], whereas JSON is relatively new having two competing standards published in 2013 [25] and 2014 [14] respectively. The amount of time XML held as the only means of online data representation solidified its exclusive use in both private and enterprise environments for a wide array of use cases, from representing enterprise data to personal sensing [52] and video annotation [7]. As such, many problems today which are grouped under semi-structured data representation focused on the specifics of the XML language. Research on XML and data warehousing falls into one of two categories; the first focuses on creating data warehouses from XML data or from a combination of XML data and an existing data warehouse. The second category focuses on XML formalisms of data cubes and warehouses, to reduce transmission times for OLAP analysis. While the latter may not immediately appear relevant to the research presented in this thesis, these works present means in which a data mart may be represented outside of a traditional relational data warehouse.

In [13], the authors present a means of creating a data warehouse and data cubes from XML data through the use of X-Warehousing. This process uses XML as an abstract model to capture the requirements of a data warehouse, so that XML data can easily be populated. Like many ETL processes, the system focuses on the requirements of the warehouse first, with a user defining the formalism of the data warehouse with respect to the desired queries.

The process of mapping the user-requirements to the underlying data is achieved through the use of XML Schemas with the schemas transformed into attribute-trees to facilitate integration. This differs from our approach as the authors in this work use attribute trees as they are dealing with XML data, while we propose the usage of Graphs. This is necessary as not all streaming data will be represented as tree structures so a more complex level of abstraction is required. Interestingly, the resulting data cubes and facts are not stored in a Data Warehouse, but as a series of homogeneous XML documents, with each document constituting a single fact. This approach would allow for fast query times for small facts, as loading a fact for OLAP analysis would constitute simply loading a single XML document.

We will adopt a metamodel approach which will have the similar effect of defining the minimum amount of required data, so the user can easily formalise the definition of the cube. In this research, it is referred to as the `Minimal XML document content`. However, we will opt for a more automated approach which will only require the user to supply missing data (where possible) in order to ensure a *data sources* compliance with the metamodel.

Azabou et al present a new multidimensional model, designed to create OLAP cubes from documents (XML) [6]. The model consists of three layers, each with a key role in the overall process. The `Standard Layer` which represents standard dimensions, derived from the top-level elements of a document. The `Semantic Layer` represents semantic information obtained from the document. This is a similar resource to the ontology used in many systems, which seeks to form semantic connections between data sources. Finally, the `Document Layer` is a set of documents containing a similar structure.

The transformation process is composed of three phases: `Pretreatment`, `Generation` and `Instantiation` of the model. `Pretreatment` is equivalent to an *Annotated Graph* phase. The pretreatment phase examines documents in their native format in order to provide annotations which can help further analysis. This step annotates edges between nodes in the XML tree with annotations provided by a user who has examined the source document. In our approach, we require this step to be fully automatic. The generation phase then constructs the model. This is achieved using a series of rules for XML document parsing. Similar to our approach, this phase utilises an ontology (or other resource) represented by a semantic dimension. This dimension is examined in order to extract measures from text content. Finally, user-intervention is required at the end of this process in order to verify the structure of the model.

The authors in [40] present a system which integrates XML data at a conceptual level where they present a methodology for integrating XML data with relational data. The system accepts a Document Type Definition (DTD) from which the UML representation of the document is generated. This makes the DTD easier to comprehend by the user. Additionally, a UML representation of the relational data to be integrated is also presented. A similar method of abstraction is proposed in our approach but we opt to use a graphs model as opposed to the UML model employed in this research. A user of the system creates a UML diagram representing the structure of the cube they wish to create. This structure is then analysed and the required queries are generated to extract the data. These queries can either take the form of XML queries or SQL depending on where the source data is located.

In [55], Pujolle et al focus on text-heavy XML documents, in which they present a methodology for extracting data from XML in order to create OLAP systems which can utilise this data. The first stage in this process captures users requirements using a matrix to represent the *requirement query.* This matrix is then analysed in order to determine which attributes require interaction. This matrix and the corresponding interactions are then used to construct a Galaxy model, a model of the authors design which encapsulates all required attributes necessary for a fact to be constructed from text data. Once this multidimensional model is constructed, it must be verified against the data sources. Our approach will have no requirement for this step, as the multidimensional model is generated from the data sources themselves. The authors use XPath as a means of mapping from source to target schemas which is similar to our approach. However, as we may deal with data sources with multiple formats, multiple mapping languages are are necessary.

In [35], the authors present X-Cube, a set of XML document standards aimed at reducing the time taken to transmit data cubes over a network and presenting an open document format for interoperability between data warehouses. X-Cube aims at representing the constituent components of a data mart through three documents: `XCubeDimension` detailing information about hierarchical dimensions; `XCubeFact` representing the individual facts (cells) in the data cube; and `XCubeSchema` which represents the cube through combining the dimensions and facts detailed in the other documents. Similar to our approach, the authors provide a system for representing the components of a data mart outside of a traditional relational data warehouse. However, this approach is created from analysing existing data warehouses while we propose to create a data mart by analysing different forms of semi-structured data sources.

Niemi et al present a system which collects OLAP data from heterogeneous sources, represented by XML to construct an OLAP query (cube) [50]. This allows the user to pose a query to an OLAP server, and create the cube by obtaining data from all relevant data stores. The user expresses an MDX query which runs against a global OLAP schema. The system then analyses the query and determines a suitable OLAP schema for the query. This process requires a deep knowledge of the source data and requires the construction of mapping rules in order to translate data from source to target similar to our approach. This is quite different to what we propose in that our schemas and mappings are constructed automatically, by extracting the facts, dimensions and measures from within the data source. Similar to other approaches, their XML representation of the cube consists of fact, dimension and measures objects.

The authors in [73] use a combination of UML and XML in order to represent multidimensional models when designing a data warehouse. A designer would structure the data warehouse using UML. This allows the data warehouse to contain complex structures such as many-to-many relationships and multi-inheritance. Once the data warehouse has been created using these means, the warehouse is then represented using XML notation for ease in transmission. The XML notation is formalised in a DTD (Document Type Definition) and is structured in a number of steps. Each document contains a series of `PKSCHEMAS` representing a data warehouse schema. Each schema contains one `PKFACT` (fact) and many `PKDIMS` (dimensions). While this approach models the base attributes of a data warehouse (facts, dimensions and measures), our StarGraph process may produce multiple `facts` for a data source.

Previously, research focused on XML data for one primary reason: that it was the only structured or semi-structured means of representing data on the web, which gave it a strong foothold in enterprise environments, specifically in data transmission. In [46], the authors presented a means of creating a data warehouse from semi-structured social network data using Twitter as a motivating example. This task focuses on the extraction of dimensions and measures from semi-structured objects (e.g. tweets). While the source data is represented in JSON, the system first converts these objects into XML and stores them in an XML database. A similar method of storing data prior to processing will be used in our approach. However, our system stores data in its raw format through the use of a Data Lake. This has the benefit of providing us with a mechanism for quickly detecting changes in the structure of web data.

These XML database schemas are then queried in order to construct a relational model for the source data. This step provides the system with information regarding the relationships between attributes, their respective data types and their cardinalities. While a similar approach will be used by our system, this is achieved through an `Annotated Graph Construction` process. Also similar to our approach, all numeric data items that are not marked as keys are candidates to be used as measures. However, unlike our approach, their system contains a data enrichment step, using external APIs to enrich the tweet for example using sentiment analysis.

Ravat et al present a means of combining an existing multidimensional warehouse with linked open data at the conceptual level' in order to provide more information to decision makers [57]. This methodology takes the form of a modelling solution which they name a *Unified Cube*. The model consists of four objects, the first of which is a `Dimension schema` that is a series of tuples used to model a hierarchical dimension at all levels. A `Dimension instance` seeks to map a hierarchical dimension to the conceptual level provided by the linked open data. A `Unified Cube schema` which consists of a finite set of dimensions and measures, and a set of rules (`links`) which are used to combine data from multiple sources. A similar approach will be used in our research when constructing the mapping rules for the final data mart. Finally, a `Unified Cube instance` is constructed to connect the attributes of the cube to the linked open data repository at the conceptual level.

The construction of a unified cube is a two stage process. The first step is to construct an `exportation cube` which serves as an abstract representation of all schemas set to be integrated into the cube. A similar approach is used by our process where individual StarGraphs are used to construct a Constellation. The second step proceeds to link these individual schemas to construct a Unified Cube. A similar approach is used by our methodology, however in our case the `type mapping` and `term mapping` processes provide these semantic links between StarGraphs.

The authors in [49] present a methodology for integrating ontologies using a series of approaches including machine-learning, information retrieval and matching ontologies from different languages. The machine learning approach takes input in the form of two separate ontologies, and an integrated ontology, representing the two previous ontologies having been created by a domain expert. The machine learning process then examines the two source ontology's with respect to the integrated ontology in order to learn classifications of integrated attributes and how they relate to each other. Using our approach, the integration ontology would represent our global integration ontology with individual source not requiring separate ontologies in order to facilitate integration.

The information retrieval method assumes that previously integrated ontologies are not present and thus, the machine learning method cannot learn features. This method examines attributes in each ontology and attempts to provide mappings based on a combination of the similarities of labels and the *informativeness* of ontology attributes. It is important to note that our process will also require an integration ontology in order to make correct integration decisions. The final approach seeks to resolve differences between ontologies of differing languages. While this may not seem pertinent to our work, remember that data from the web may be in any language and the case studies provided in Ch 3 utilise datasets from different countries, and are published in different languages.

The approach for ontologies of different languages is similar to the information retrieval approach, however there exists an additional step, namely the translation of terms. The authors use Bing Translate, however any number of translation tools may be used. Our work overcomes the differences in languages using the integration ontology, as all attributes are assigned a canonical term during the `term-mapping` process, with a further level of abstraction being applied during the `type-mapping` phase.

In [43],Kittivoravitkul et al present a means of integrating semi structured data sources in which they focus on normalising semi structured data before integration and storage of this data in a relational database. The authors use YATTA which is a modelling formalism for semi structured sources. It consists of two tree structures: the `YATTA schema` and the `YATTA data tree`.

The `YATTA schema` can be roughly equated to an annotated graph with both storing the name of attributes, their data types and relationship cardinalities. While the YATTA schema contains more in-depth possibilities for relationship modelling (e.g. 1-n, n-m etc...), it does not contain mappings to source attributes present in our proposed annotated graph. The `YATTA data tree` represents the physical data, as expected, as a tree structure representation. Data is normalised to 3NF in order to present a traditional normalised schema based on the source data. The integration process seeks to construct a global schema by a series of transformations with a user generating this global schema and providing mapping rules to the system. This differs from our approach, where mappings must, in terms of scaling, be automatically generated.

The authors in [80] present `Graph Cube`, a data warehouse model which supports OLAP queries on graph models. There are three separate models necessary to represent a data cube from graph data. The first model captures the `multidimensional network`. This model represents the graph 'as-is', consisting of a series of nodes and edges, each containing dimensions and attributes. The second model, which provides OLAP functionality, is the `aggregate network`. This model represents a graph which has undergone an aggregation function (e.g. ROLLUP) which stores the aggregates as weights between nodes (attributes). The final model is the `graph cube` model. This is constructed in a fashion similar to a traditional data cube by performing multiple aggregation functions along a combination of axes in order to represent the full data cube. In this instance, the `graph cube` is a collection of aggregate networks.

Our proposed graph model will contain the ability to construct the all cube (*) as our work is not focused on storage efficiency. The additions of the aggregate model graph to provide our approach with a more robust query mechanism is discussed in the final chapter of this dissertation.

2.3 Related Research: A final Summary

There is a large amount of work present in both supplementing existing data warehouses with semi-structured data, and creating data warehouses from entirely semi-structured data. However, the common theme presented is the need for an abstract representation of the data warehouse to overcome the differences presented by different data formats. These abstractions present in multiple ways, such as an XML logical model [13], a multidimensional conceptual model [57], use more conventional notation such as UML [73]. All approaches tend to follow the same basic steps in achieving this goal. The first step involves analysing a data source and converting it into an abstract format. The next step involves comparing this abstract representation to a pre-constructed semantic model, which can take the form of an ontology, a specific dimension or as a series of rules. Once this is step is completed, the sources can be integrated on a semantic level and it is logical that our approach would follow a similar path.

However there remain outstanding issues.The first is that despite the fact that the research analysed present a means of integrating data, they focus more on the integrated model rather than the process for integration. The second lies in the method for materialising data marts. The inclusion of web data requires an entirely new approach as traditionally, data warehouses are updated on a batch schedule. This may not provide the up-to-the-minute information that high velocity web streams are capable of providing. We will directly address this issue using a Data Lake and a metabase to manage the update velocity. Indeed, additionally, none of the research we have uncovered utilises a Data Lake in order to capture semi-structured web data efficiently.

ETL processes differ based on requirements of the system. Some focus on a particular domain of data [10] while others may focus on user-requirements [59]. However, the common trait for all processes is the need for some form of ontology, or a means of representing how data is mapped. It should be noted that in addition to the ontology, most sources assume a mapping to source exists within the ontology. We will adopt a different approach. We will employ a method which constructs the mappings during StarGraph construction and, (at that point), without the need for an ontology.

When considering XML-based approaches, they all construct cubes from XML data by adopting a user-centric approach. The user specifies a query which is then used as the basis for the construction of the data mart. Furthermore, a level of abstraction is required when dealing with data from multiple sources. Here, many chose to use UML for two reasons: firstly, as a means of abstraction as the modelling language has sufficient capabilities to represent data from both XML and relational data;

and secondly, because UML is considered a well understood means of conveying information to designers. However, all approaches assume a query designer has sufficient knowledge of the underlying documents and in many instances, has access to DTDs or XML Schemas which can be fed to a system in order to understand and analyse the structure of the document. We cannot make this assumption when constructing data marts from web sources. Our approach must deal with data available from the web, where the ability to obtain a well structured schema is not always possible. As such, our system must be designed to analyse the source documents *without* the need for a schema. Additionally, our system uses graphs as a means of abstraction at the conceptual level, as unlike UML models, they can be analysed and converted and processed. Finally, approaches focus on facts as *individual* entities (e.g. data cubes) rather than an attempt to model the data mart *in its entirety.* Although multiple XML documents could be used to represent the individual facts within the data warehouse, the absence of shared dimensions reduces each fact to exist in isolation. For our work, where multiple facts may be created from a single data source, such an approach would not work, as updating a dimension would require an update to multiple documents, rather than a single update required by a database. We feel that this is a crucial requirement to constructing data marts from multiple, possibly heterogeneous, web sources.

While the process of creating data warehouse from other non-XML formats is relatively recent compared to the large body of work presented on XML specific solutions, there exists a requirement to integrate non-XML web data with existing data warehouses. This is in part, due to the rise of semantic technologies (RDF, SPARQL etc..) but also to the growing number of online resources providing data in non XML formats (i.e. JSON, CSV). This requires a sufficiently rich canonical model to enable communication between various web data representations.

Chapter 3

An Architecture for Dynamic Processing of Data Streams

In order to deliver our research goals, we require a system capable of interpreting unknown data streams, extracting data in multi-dimensional terms and integrating across data streams. In this chapter, we provide a high level overview of our system in terms of the components which deliver the different steps in this process. In section 3.1, we introduce the architectural components; section 3.2 details the data-stores used by the system; In section 3.3, we discuss the system ontology which has a crucial role in stream integration, and in section 3.4, we provide details of the case studies which will serve as running examples throughout the remaining chapters.

3.1 System Overview

An Extract Transform Load (ETL) process must provide critical functionality in order to deliver a system suitably agile to process semi-structured web streams or unknown source data. Firstly, the system must be able to capture and store data from relevant data sources. Secondly, the system must possess an *understanding* of the source data, specifically the ability to identify facts, dimensions and measures required for analysis and creation of a data mart schema. The system must be able to devise extraction and transformation rules to take the data from its raw format and store it in a data mart. For multiple data sources, the system must first identify

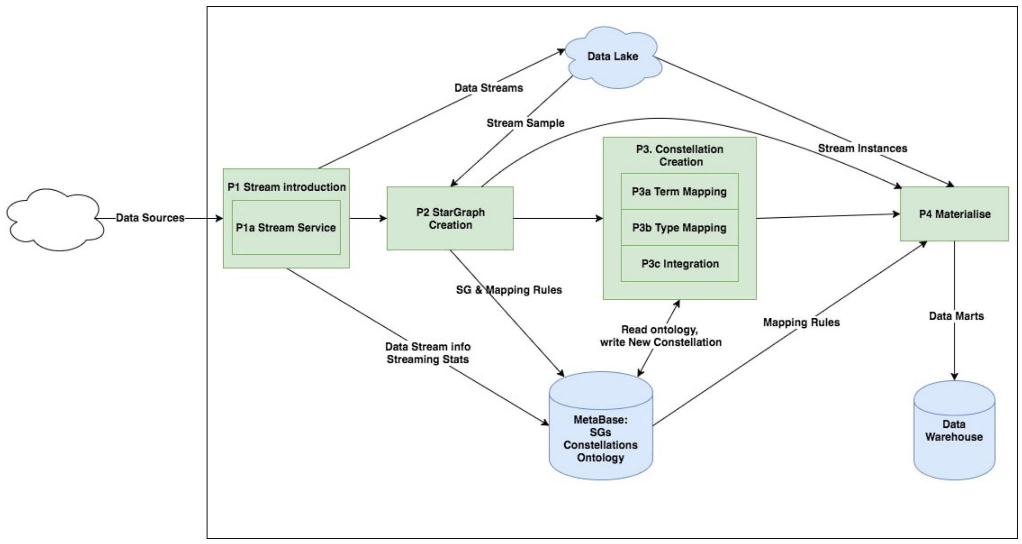


Figure 3.1: Extract-Transform-Load Architecture for Web Streams

the facts, dimensions and measures per data source, then devise an integration strategy for the data sources to present a unified data mart. When integrating data from multiple sources, the system must be able to make use of abstractions and generalisations which would be apparent to a manual ETL designer. This information must be stored in the system for future verification, optimisation and other forms of analyses. Finally, the system must include the ability to incorporate optimisations that are provided by a domain expert to increase the efficiency of the system.

The ETL architecture developed as part of this research, comprises four main components and three different storage models to manage the demands of the different types of processing involved. Figure 3.1 outlines the system architecture with the system processes (and sub-processes) in green and the system data stores in blue. In the remainder of this section, the four main components (Stream Introduction, StarGraph Creation, Constellation Creation and Materialisation) are described in detail.

3.1.1 Stream Introduction (P1)

This process is used to add new data sources to the system. An overview of this process can be seen in Fig. 3.2.

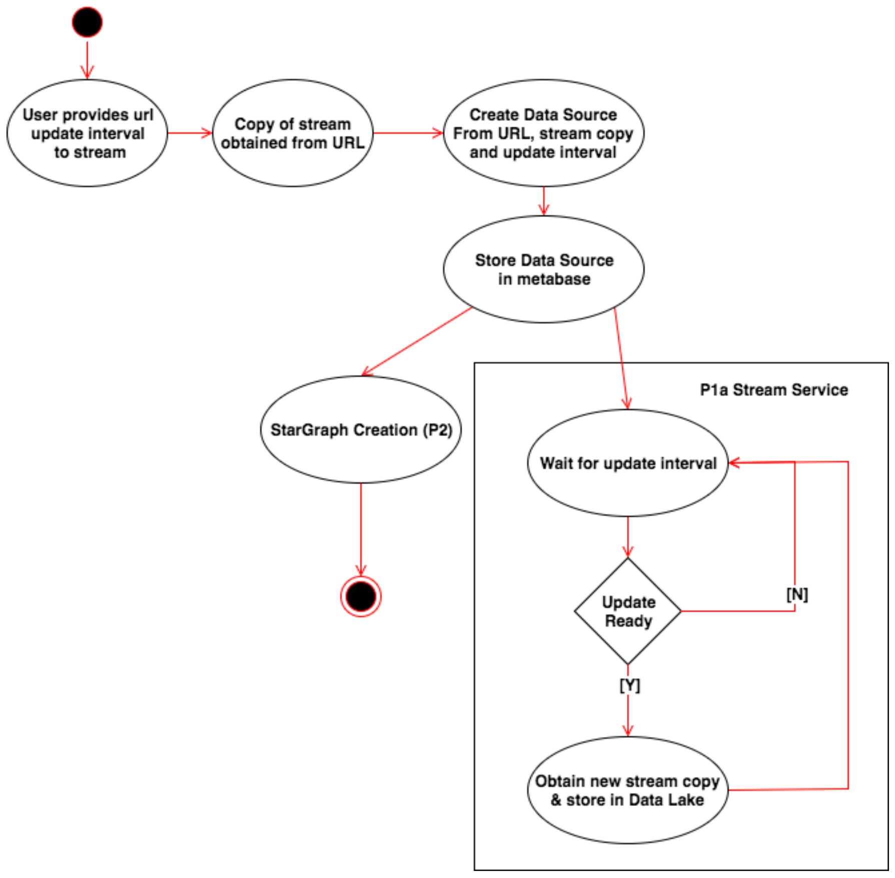

User provides url update interval to stream
Copy of stream obtained from URL
Create Data Source From URL, stream copy and update interval
Store Data Source in metabase
StarGraph Creation (P2)
P1a Stream Service
Wait for update interval
Update Ready
[N]
[Y]
Obtain new stream copy & store in Data Lake

Figure 3.2: P1 Stream Introduction Activity Diagram

In order to create and populate a data mart from a data stream (or series of data streams), each data source must first be introduced to the system by a user. The user provides a *url* to the stream (which serves as the streams unique identifier) and supplies an update interval in milliseconds, to represent how often new information from the data stream becomes available. At this point, the primary copy of the data stream is retrieved and the *url*, update interval and the copy of the stream obtained are stored in the metabase. For some data sources the stream update may provide data already stored within the system along with new data, while other streams may provide an entirely new copy of data.

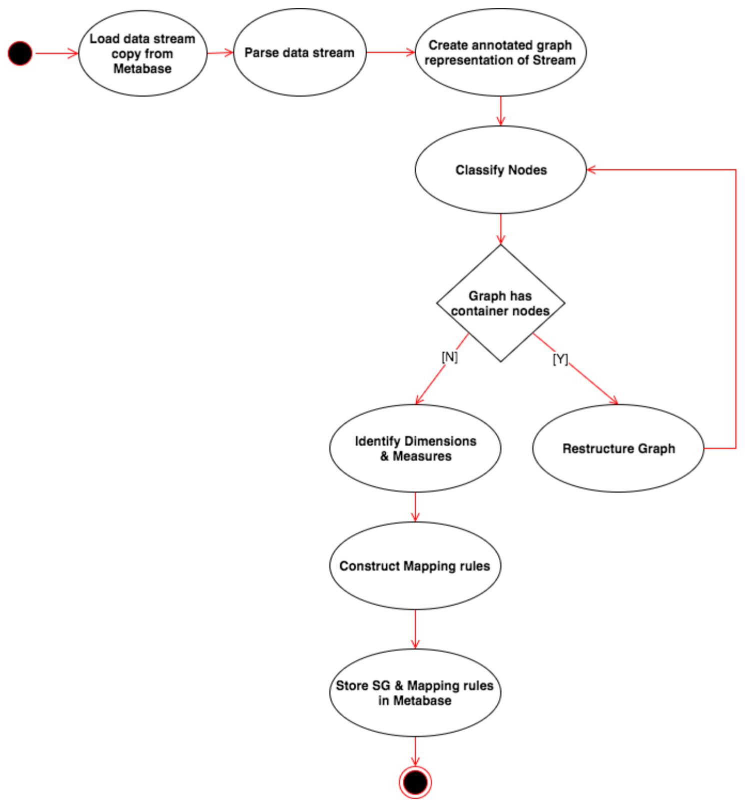

Load data stream copy from Metabase
Parse data stream
Create annotated graph representation of Stream
Classify Nodes
Graph has container nodes
[N]
[Y]
Identify Dimensions & Measures
Restructure Graph
Construct Mapping rules
Store SG & Mapping rules in Metabase

Figure 3.3: StarGraph Construction: Activity Diagram

Once these items are stored, two independent processes are triggered: a metadata (or data definition) process, followed by a process for extensional data (data insert). The first process is StarGraph Creation (`P2`) which takes the copy of the stream and uses it as a basis for constructing a StarGraph. The second process is the stream service. This process continuously obtains new copies of the data stream and stores them in their RAW format in the Data Lake to await population.

The stream service waits until the specified update interval has occurred. Once this happens, it fetches a new copy of the stream and stores it in the Data Lake in its raw format. Additionally, it creates `Data Instance` records in the Metabase indicating when the raw data in the Data Lake was obtained.

3.1.2 StarGraph Creation (P2)

The goal of the *StarGraph Creation* process is to construct a *StarGraph* from a single data source. A StarGraph is a graph structure which acts as the common data model upon which the entire process is based. Data sources provided by the user in (`P1`) may be of a variety of data formats (XML,CSV, HTML or JSON) with each having a unique structure. However, as the main goal of the system is to provide data marts by integrating this data, there is a need to provide a common model which is powerful enough to represent both data and context in any of the required data streams. It should also be suited to a suitable integration strategy and be supported by the functions required to deliver the necessary integration [8]. In the case of this research, it must capture facts, measures and dimensions as these are the structures crucial to our research.

The process is illustrated in Fig. 3.3. A Stargraph is created from the primary copy of a data stream stored in the metabase which was acquired during `P1`. The data stream copy is parsed and converted into an annotated graph. **Nodes** in the graph represent objects found in the data source, and **Edges** represent the relationships between these objects. Each node contains the *data type* of the object while a generated *mapping query* is used to extract the object from the data source. Each edge is annotated with the *cardinality* of the relationship between objects. At this stage, the graph is simply a graph representation of the data source, with no understanding of the constructs of a multi-dimensional schema. As this is a high level overview of the transformation process, a detailed description of the StarGraph is deferred until chapter 4.

The next stage in the process is to classify nodes based on their data type and relations to other nodes. The classification process seeks to identify important dimensions (e.g. Date and Geo dimensions), items which may become measures and nodes which are redundant and should be removed. This step involves a traversal of the graph and removal of all nodes which have been classified as *containers*. However, by removing some nodes marked as containers, this process may in fact create more nodes which should be removed. To overcome this, once the graph has been restructured and containers are removed, all nodes are then re-classified and any new nodes marked as containers are also removed. This process repeats until no nodes are marked as containers.

At this point in the process, all numeric items are considered measures and all other nodes classified as dimensions, attributes to dimensions or sub-dimensions (of a dimensional hierarchy). The process now has sufficient information to create a data mart with identified measures and dimensions, and mappings to bind each node to its place in the data source. The next step is to use this information to construct mapping rules, which will be used to populate a data mart at a later stage. Once the mapping rules have been created, the StarGraph and the mapping rules are stored in the Metabase.

Issues to be Addressed. There are a number of challenges to be overcome in the transformation process, caused mainly by the quality of the source stream. In an integration process, all possible data formats and structures must resolve into the same StarGraph structure. XML data is highly structured in a tree-like format. The structure of the nodes in an XML tree indicate a hierarchy of sorts, which can be used to identify objects which may potentially become dimensions. HTML sources also take a tree-like structure but in these instances, nodes may not represent a hierarchy of objects, but instead the structure of the document. In addition, as HTML is primarily designed to be used in conjunction with CSS and Javascript to be displayed in a browser, data which appears structured to a user, may be represented internally as an entirely different structure.

CSV files also have their benefits and disadvantages. Firstly, their structure is quite simple, namely, a matrix where the first row represents column names. However, in comparison to other data sources. the CSV file does not have a means of indicating a data type which is essential for detecting measures (which are numeric). For example in JSON, all strings must be encapsulated by " or ', while numeric types are written without quotes. However, the CSV structure does not contain this level of type information and relies on the user's knowledge of the data in order to determine types and relations. Finally, missing values (which are a common occurrence when using real-world datasets) adds additional complexity to this process.

All difficulties listed above relate to the structure and data presented by a data source. One of the main goals in this research is to automate, as much as possible, the integration process and construction of the multi-dimensional data mart. However, there still exists issues related to the semantics between data sources which must be captured in order to facilitate semantic data integration. The solution, presented in detail, in the next two chapters, must address these issues.

3.1.3 Constellation Creation (P3)

The goal of this process is to create an integrated data mart from two or more data sources, now in the form of *StarGraphs*. The integrated StarGraph is referred to as a `Constellation`. While the previous process (`P2 StarGraph Creation`) ensures that all data sources are using a common data format and structure, there remain many issues when integrating these data sources.

An automatic approach to integration may provide an integrated data mart, however it does not provide a *usable* mart. This is mainly due to the semantics of the underlying data between sources. Data may be of differing granularities, use different units of measurement, or may contain abstractions required for integration which are not apparent to the automatic process. To address these issues, this process utilises an ontology to influence the integration process in order to provide a semantically correct data mart.

Let us begin with a brief outine of how an integrated data mart is made. Initially, the designer selects the data sources to be integrated. These data sources have previously been added to the system in process (`P1`) and have associated StarGraphs previously constructed (`P2`). The creation of a Constellation is delivered using a loose version of the ladder approach described in [8], where two sources are merged first, and in each subsequent step, one new source is added. This loose version of the integration strategy will always deliver an identical version of the final integrated schema, irrespective of the ordering of sources during the integration process. This is discussed and shown in chapter 5.

The purpose of the three subprocesses `P3a`, `P3b` and `P3c` is to examine StarGraphs with respect to their semantic meaning in order to determine the subtleties in the integration process that address the issues described earlier.

The processes `P3a` and `P3b` examine each StarGraph and query the ontology to identify common types and abstractions present in both data sources. The process

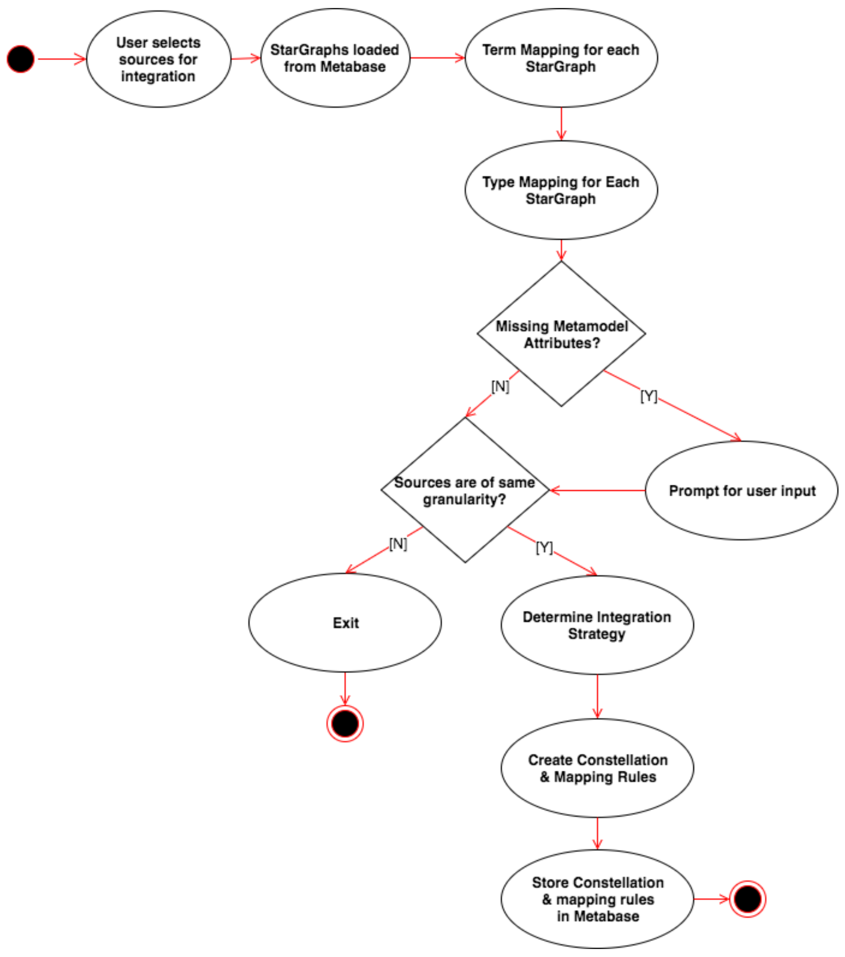

User selects sources for integration
StarGraphs loaded from Metabase
Term Mapping for each StarGraph
Type Mapping for Each StarGraph
Missing Metamodel Attributes?
[N]
[Y]
Prompt for user input
Sources are of same granularity?
[N]
[Y]
Exit
Determine Integration Strategy
Create Constellation & Mapping Rules
Store Constellation & mapping rules in Metabase

Figure 3.4: Constellation Construction: Activity Diagram

`P3c` then examines each StarGraph with respect to their compliance with the metamodel. This process is used to identify missing attributes which are required in order to facilitate semantic integration.

Term Mapping (P3a).

The goal of the *Term Mapping* process is to resolve inconsistencies arising from naming conventions across different data sources. The process attempts to capture the knowledge the ETL designer has when making connections and comparisons between data sources. A common use for Term Mapping resides in the *Geo* dimension in our case studies, where there are numerous ways in which different country names can be represented.

However, the main use for Term Mapping is to provide *direct equality* between terms. For example, two of the case study terms '`IRL`' and '`Éire`' are not related using any standard string matching algorithm. However, both terms refer to the country Ireland. The term mapping phase provides this equality using a list of terms stored in the ontology. This does require that a set of canonical terms are recorded in the StarGraph Ontology and over time, each term used in the integration process, is added to the ontology.

Type Mapping (P3b).

The *Type Mapping* phase begins once canonical terms have been applied to the nodes in a StarGraph, and its goal is to assign canonical *types* to each node.
While term mapping assigns canonical terms, type mapping links these terms across different types. For example, consider two terms '`IRELAND`' and '`FRANCE`'. A manual designer may have the knowledge to integrate these two nodes as they both occupy the `Geo` dimension. The type mapping process seeks to capture the *abstract concepts* which link these nodes in order to further facilitate semantic integration.
There are five base types defined in the metamodel and described later in section 3.3.3. These types have been identified as the minimum requirement of semi-automatic semantic integration. However, a user may define their own types in addition to those defined in the metamodel.
Both the term map and the type map wordnets are loaded into an in-memory database to reduce the overhead incurred by querying the ontology for each node in both StarGraphs marked for integration. Similar to term mapping, the difficulties arising from the type mapping process are centred around the completeness of the ontology, ensuring that every possible abstraction is captured.

Integration (P3c).

The integration process examines two StarGraphs which have undergone term and type mapping and determines an `integration strategy` which will be used to create an integrated graph (Constellation) and mapping rules to create an integrated data mart. The integration process is a three phase process: metamodel check, granualrity check and strategy selection.

- **Metamodel Check**. The first phase in the integration process is to examine each StarGraph for compliance with the metamodel. In brief, the metamodel is a series of types which *must* be present in a data source in order to provide semantic integration. The metamodel is discussed in-depth in section 3.3.

  These required types are assigned during the type-mapping phase, although at this stage, there remains the issue of missing attributes. To address this issue, the system first verifies that each data source contains all of the abstract types necessary for semantic integration. In the event that one (or both) StarGraphs do not contain all the types as specified in the metamodel, a `MISSING_ATTR` event will be triggered. This event prompts a user to supply a value for the missing metamodel value. The end result of this process is a fully annotated StarGraph with all required metamodel attributes.

- **Granularity Check**. Once compliance with the metamodel has been verified, the next step is to examine the data within each StarGraph to determine their granularities. The granularity check examines both StarGraphs for hierarchical dimensions. In the event both graphs occupy the same hierarchical dimension, nodes are checked to see if they both reside on the *same level* of the hierarchy.

  The granularity check resolves conflicts which arise when integrating data across dimensions which are hierarchical in nature. A common example involves the date dimension where one data source has a monthly granularity and the second source a daily granularity. In this event, the data cannot be directly compared. These events trigger a `GRAIN_MISMATCH` and requires the finer-grained StarGraph to undergo a `rollup` operation so that both StarGraphs occupy the same dimension in the appropriate dimension hierarchy. The challenges in this process reside in ensuring the system has a deep understanding of hierarchical dimensions, and where data extracted from a data source would lie inside these hierarchies.

- **Strategy Selection**. The final step of the Integration process is to determine the integration strategy to create an integrated data mart. There are two possi-

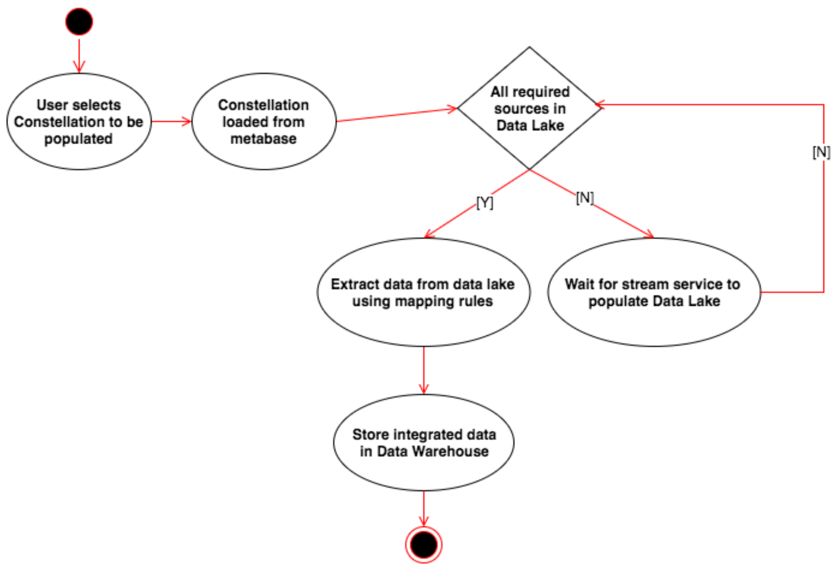

User selects Constellation to be populated
Constellation loaded from metabase
All required sources in Data Lake
[Y]
[N]
[N]
Extract data from data lake using mapping rules
Wait for stream service to populate Data Lake
Store integrated data in Data Warehouse

Figure 3.5: Materialisation: Activity Diagram

ble integration strategies: `row-append` and `column-append`. The row-append strategy is used for StarGraphs which contain identical metamodel attributes. In this instance both StarGraphs can be directly integrated. The column-append strategy encompasses all other possible integration approaches. This process identifies common metamodel attributes between both data sources and using these combinations, identifies the integration attributes for both StarGraphs and creates a unique integration strategy for the Constellation. Once the integration strategy has been determined, a new series of mapping rules are constructed to extract data from multiple sources residing in the Data Lake and store them in a unified Data Mart.

3.1.4 Materialisation (P4)

The materialisation process populates a data mart by applying mappings stored in the metabase, to raw data stored in the Data Lake. An overview of this process can be seen in Fig. 3.5.

The materialisation process must initially be driven by the data scientist or analyst who requires the dataset. The analyst selects an integrated data mart (Constellation) or in some cases, a single source data mart (StarGraph) for population.

If the data mart has not been previously populated, the metabase is queried for all Data Instance items related to the sources. Using these instances, the appropriate (raw) data is obtained from the Data Lake. These raw instances are grouped together based on the times they were obtained from the Streaming Service. In the event that not all data sources are yet present in the Data Lake, the system will wait for the Stream Service to gather new data for the streams. This can occur when the update granularity for 2 sources in the same mart differ. For example, one source may have a 2-min update interval while a second has a 5-min update interval and thus, the second source may be awaiting a refresh.

Once appropriate data instances have been grouped, the mapping rules are applied, and this populates the data mart. This process continues until a termination event occurs. Termination events are discussed in chapter 6.

3.2 System Data Stores

In this section, we describe the data stores necessary to support the different processes in the StarGraph system. These include the Data Lake, which is used for storing raw data streams; the StarGraph metabase which acts as a repository for instances of the system's constructs; and finally, the StarGraph ontology which is central to integration and data mart construction.

3.2.1 Data Lake

The *Data Lake* acts as a repository for the raw data streams where a single stream per data source is stored, in preparation for population commands. The benefit of using a data lake is twofold: firstly, by storing the data in its raw format, it allows the system to quickly detect changes in a data source; and secondly, the implementation of a data lake separates the source data from the ETL process. This separation provides a means of reproducing results, and allows the StarGraph process to be modified and extended without the loss of data.

The Data Lake is written to by the Stream Service (`P1a`) and the information is read by the materialise process (`P4`). When the stream service process obtains a

new copy of a stream, it creates a new `Data Instance` object inside the Metabase and stores the raw stream into the Data Lake. The Materialise process (`P4`) extracts the raw data for all data sources specified in a `Constellation`.
However, once population occurs, the Data Lake retains its raw copy of the stream so that future data marts have a historical backlog of data from which to populate.

3.2.2 StarGraph Metabase

The Metabase contains all metadata generated by the system. This metadata includes information about data sources (Definition 3.1), StarGraphs (Definition 4.1 in the following chapter), instances of data streams captured in the Data Lake (Definition 3.2) and the data mart definitions and mappings used to populate the data mart (Definition 3.3). Each data source captures information about an individual data stream which is used to construct a data mart. The system captures the *url* for the data source with the corresponding update interval for that source. This update interval is used by the Stream Processor (sect 3.1.1). Data sources are created by the Stream Introduction process (`P1`) and are read by the StarGraph Construction process (`P2`).

Definition 3.1. Data Source.
A data source $DS = \langle U, N, W, I \rangle$ is a four element tuple where U is the unique identifier the system generates for each new data source; N is the mnemonic associated with the data source; W is the url *at which the raw data is available for retrieval from the remote data source, and I is the update interval of the* url *source.*

A *data instance* is created for each data stream stored in the Data Lake. Data Instance objects are created by the Stream Service Process (`P1a`) and are read by the Materialisation process (`P4`).

Definition 3.2. Data Instance.
A data instance $DI = \langle I, U, D, T \rangle$ is a four element tuple where I is the unique identifier the system generate each time new data is retrieved from a specific source; U is the identifier of the source from which data is retrieved; D is the UTC date-time

of data retrieval; and T is the file format of the raw data file.

A Cube Instance describes a populated data mart. The identifier D is used to determine data inside the Data Lake which has yet to be extracted and used to update the Data Mart. The Cube Instance is created when a Constellation is set to be populated for the first time using the Materialisation process (`P4`).
Additionally, this process reads this information in order to determine what raw data residing in the Data Lake has yet to be extracted and transformed and stored in the Cube residing in the Data Warehouse.

Definition 3.3. Cube Instance.
A cube instance $CI = \langle C, D, L \rangle$ is a three element tuple where C is the unique identifier the system generates for each new data mart created; D is the UTC date-time of the cube creation/update; and L is number of times the cube has been loaded since its (first) creation.

3.2.3 Data Warehouse

A data warehouse stores populated data marts. The schema for the data mart (facts, dimensions and measures) are defined by a StarGraph and the populated data is obtained by using the mapping rules provided by the StarGraph on the raw data stored in the Data Lake.
Each data mart residing in the Data Warehouse has its own structure, which is generated by the Constellation Creation process (`P3`). However, this schema is not stored in the Data Warehouse until the Materialisation process occurs. When the Materialisation process is triggered, it will create the data mart schema as defined by the Constellation and begin to populate the data mart with data it obtains from the Data Lake.

3.3 StarGraph Ontology

An Ontology is a resource which captures all knowledge of a particular domain [7]. The StarGraph ontology stores information about abstractions between data sources which drives the semantic integration process. There are three main components that comprise the ontology: *Terms*, *Types* and the integration *Metamodel*.

3.3.1 Terms

In the StarGraph Ontology, *Terms* are triplets which map a term found in a data source to a canonical term. These terms resolve differences in naming (e.g. '`US`' and '`America`'. These differences may arise from different domain language, or different coding mechanisms employed by the stream designer.

Definition 3.4. Terms.
Terms are stored as a triplet $T = \langle ST, N, CT \rangle$ *where* ST *relates to the source term, a node in a StarGraph;* N *is the unique url to the data source; and* CT *is the ontology term.*

A Term triplet stores the data source URL in addition to the source term, as some strings which may be identical have different contexts. For example, one data source may have the term '`Ireland`' relating to the total *sale* of pigs in Ireland while in another source, this term relates to the total *slaughter* of pigs in Ireland.
Terms are created by domain experts as part of the ontology and are used during the term mapping process.

3.3.2 Types

Types capture abstractions known to the domain expert (or ontology designer) which are crucial to the automation of the overall StarGraph process. The common usage for *Type* instances is in identifying components of the `Geo` dimension. While term mapping resolves naming differences between the same canonical term, *type mapping* links canonical terms into groupings under a common abstraction (e.g. a country). For example, the two terms '`Ireland`' and '`France`' are both canonical terms. However, an application designer would know that these terms are of the same type (in this case, a country).

Definition 3.5. Types.
A Type is represented as a tuple $T = \langle CT, TY \rangle$*, where* CT *refers to the* `canonical`

`term` *and TY is the* `type` *to which this term belongs.*

Types are created by a domain expert as part of ontology construction and are used during the type mapping process.

3.3.3 Metamodel

The integration metamodel describes a series of attributes whose presence is crucial for StarGraph integration to accurately process. When two StarGraphs are selected for integration, their compliance with the metamodel is first confirmed. The presence of the metamodel attributes determine the specific methodology for this specific integration.
An enrichment process uses the metamodel to annotate each attribute in a StarGraph with its metamodel type. Grouped together, these annotations are called the `Attribute Semantic` property, which is specified in Definition 3.6.

Definition 3.6. Attribute Semantic.
An attribute semantic $AS = \langle D, G, I, M, U, \rangle$ is a five element tuple where D is a Boolean describing whether the attribute represents the Date dimension; G is a Boolean describing if the attribute is a Geo dimension; I indicates the name of the measure in case the attribute is a measure of interest; M describes the metric for the measure and U specifies the units for the metric.

These properties influence the integration process (`P3`) in order to construct a semantically-correct data mart.
The metamodel is immutable. It is an abstraction based on the types in the ontology, a series of properties required for semantic integration to occur. Metamodel *instances* are created and modified as the ontology is extended for new data sources. The presence or absence of metamodel attributes is determined by the types assigned to a data source during the type mapping phase. These types are then examined during the metamodel verification phase of the integration process.

3.4 Evaluation Case Study Descriptions

In the final section of this chapter, we will provide an overview of the three main case studies which will serve as running examples throughout the remaining chapters. These case studies utilise real-world datasets, have real customers/users, and operate in the agri (agricultural) domain.
Agricultural industries had a global market value of US$8 trillion in 2016. A fast-paced global industry such as agriculture requires decision-makers to have the most up-to-date data in order to predict market trends. Like many other sectors, these industries rely heavily on existing data warehousing technologies and as such, fall victim to the problems traditional ETL processes entail. This problem has been further compounded with the rise of web-APIs provided by many governmental bodies and private organisations which provide important statistics on different aspects of a countries agricultural sector.
These APIs are outside the control of agri decision-makers and are prone to change. However in these instances, the time taken to re-engineer a traditional ETL process to accommodate even simple changes ensure that the decision maker must work on out of data information. Additionally, when creating ETL processes from global data, careful attention must be made to how data is represented. Providers may use different units of measurement and provide prices in using different currencies. Further complicating issues is the presence of different languages. With each provider using different terms which relate to the same product.
Such abstractions and difficulties require an ontology to provide semantic mappings between data sources and additionally, provide functions which can be used to convert units of measurement and currencies. We will now present three case studies using data obtained from the web, drawn from disparate sources, to convey how our system may be used to construct agri-data marts in an efficient and extensible manner compared to their traditional ETL counterparts. Case studies were specified by experts from the agri domain, representing real problems, using real world data.

3.4.1 Case Study 1: Pig Price Predictions

The motivation for this case study is to create a data mart consisting of pig prices and production globally. Such a data mart would feed data mining algorithms to predict global market trends and future pricing for pig meat and production. In turn, this allows producers to estimate profits, and assign production resources (where necessary) to react to the changing forces driving supply and demand.
There are six agri data providers used in this case study, delivering thirteen separate streams to contribute to the data mart. The data sources and providers are listed in Table 3.1. We now briefly outline the role of each data provider.

- Agriculture and Agri-Food Canada [4] is a governmental body who provides statistics and weekly updates on various sectors in the agri-industry All data published is freely available on the web. Specifically for this case study, we will be using their reports on hog slaughter at various packaging plants in Canada.

- Bord Bia [12] is the Irish food board whose objective is to promote Irish food and horticulture. They provide a web interface which can be used to look up various statistics about Irish agriculture. The data used in this data mart is their statistics based on the prices and production of pigs in Ireland.

- ADHB [3] is the Agriculture and Horticulture Development Board. This is a statutory levy organisation centred around providing research and information to stakeholders in the agricultural sector. They provide weekly statistics on the price of pigs and pig production numbers.

- CME Group [20]is a designated contract market who provide economic research and publish their findings. The data used for this case study is their statistics on pig futures.

- IMF [39] is the International Monetary Fund. They publish exchange rates between currencies daily based on SDR units (Special Drawing Rights). Their data is required in order to compare pig prices that are using different currencies.

- USDA [74] is the United States Department of Agriculture. They provide an API called `Quickstats` which allows a user to query various agriculture related statistics. The data used for this data mart relate to the number of pigs slaughtered on a weekly basis.

Of the thirteen data sources listed in Table 3.1, twelve are updated weekly and the remaining source (`imf`) updates daily. Most data sources are HTML, which are displayed as tables to a user. Finally `ID` is the identifier for a specific data source. These will be used to refer to individual data sources in later chapters.

Table 3.1: Data sources used for Case Study 1

Name	ID	Format	Update Interval
Aimis_1	aim_1	HTML	Weekly
Aimis_2	aim_2	HTML	Weekly
Bord Bia_1	b_1	HTML	Weekly
Bord Bia_2	b_2	HTML	Weekly
AHDB_1	p_1	HTML	Weekly
AHDB_2	p_2	HTML	Weekly
AHDB_3	p_3	HTML	Weekly
AHDB_Bpex_1	bp_1	CSV	Weekly
AHDB_Bpex_2	bp_2	CSV	Weekly
cme_1	c_1	HTML	Weekly
cme_2	c_2	HTML	Weekly
imf	imf	XML	Daily
usda	usda	CSV	Weekly

3.4.2 Case Study 2: Price Trend Comparison

This case study compares the prices of various vegetable oils and the price of butter. A data mart is necessary to enable the comparison of vegetable oils and butter to determine what effects the production and sale of one item may have over the sale of others. For example, how do the sales and production of palm oil, affect the sale and production of sunflower oil?
While there are only two data providers for this data mart, multiple data sources are used from these providers.

- Quandl [56] is an online platform which provides economic statistics through the use of an API. Data is generated from several queries relating to the prices of various vegetable oils and dairy products.

- GlobalDairyTrade [27] is a private organisation who provide statistics related to dairy production and sales online.

Table 3.2 outlines the 9 data sources used for this data mart. The sources `ren` and `gdt` are obtained from GlobalDairyTrade, with the remainder being provided by Quandl. All sources are CSV apart form one HTML source.

Table 3.2: Data sources used for Case Study 2

Name	ID	format	Update Interval
PPOIL_USD	{`pp`}	CSV	Weekly
PROIL_USD	{`pr`}	CSV	Weekly
PSOIL_USD	{`ps`}	CSV	Weekly
PSUNO_USD	{`psu`}	CSV	Weekly
REN_SU	{`ren`}	CSV	Weekly
WLD_COCONUT_OIL	{`cn`}	CSV	Weekly
WLD_PALM_OIL	{`po`}	CSV	Weekly
WLD_SOYBEAN_OIL	{`so`}	CSV	Weekly
GlobalDairyTrade	{`gdt`}	HTML	Weekly

3.4.3 Case Study 3: Analysing Milk Production

The ability to analyse milk production worldwide can be used by agri-decision makers to identify potential trends and gaps in supply and demand for dairy products. Alternatively, this information could be used by governmental agricultural organisations to compare and contrast their country's dairy exports with others. There are three main providers of the data for this data mart:

- CLAL [19] is an Italian organisation which provides worldwide statistics centred around dairy production. The data sources used for this data mart are statistics about milk deliveries in Argentina, and milk production in New Zealand.

- Dairy World [23] is a German organisation which provides statistics about dairy production in Germany. The data sources used in this data mart are milk production statistics for Germany.

- USDA. Similar to Case Study 1, the `Quickstats` api was used to gather information about dairy production for the united states.

Table 3.3 outlines the data sources used to create this data mart. There are two CSV sources and three HTML sources, with all data sources update weekly.

Table 3.3: Sources used for Case Study 3

Name	ID	Type	Update Interval
Argentina_Milk_Deliveries	{`amd`}	HTML	Weekly
Cows' milk collection and products obtained	{`wdp`}	CSV	Weekly
USDA	{`usda_2`}	CSV	Weekly
Milk Production Germany	{`mpg`}	HTML	Weekly
NZ Milk Production	{`nzmp`}	HTML	Weekly

3.5 Summary

The purpose of this chapter was to provide a high level overview of our system architecture and the steps involved in taking unknown data streams and constructing a fully integrated data mart. This helps to provide the context for our research and allows us to present the most critical components in depth, over the next few chapters. This chapter was also used as an opportunity to introduce three real-world case studies with queries provided by agri-partners which use live data to generate the inputs for prediction algorithms. What is now required is a deeper understanding of how streams are transformed into a canonical format that facilitate an easier manipulation by our integration algorithms. In the following chapter, we provide a specification of our StarGraph model and present a detailed description of how StarGraphs are automatically created from web sourced data.

Chapter 4

A Canonical Model for Multidimensional Data Streams

In order to deliver any integration strategy, it is necessary to select or specify a data model which acts as the canonical model for the system. Its role is to represent all participating data sources by using a transformation process to convert sources into the system's representation, or into canonical model format. In this chapter, we present The StarGraph model, a canonical model for the integration of multidimensional data. In section 4.1, we provide a detailed description of the StarGraph model; in section, 4.2, we describe our methodology for extracting multidimensional data from web sources and transforming these sources into StarGraphs; in section 4.3 we describe the methods used to remove heterogeneity from sources during population and finally in section 4.4, we present an assessment of the transformation process using a large number of agri data sources that are necessary for the evaluation of our work in chapter 7.

4.1 Requirements for Managing Multidimensional Data

In order to understand the capabilities necessary for a model to capture multidimensional data, we must first specify the system requirements. In chapter 3, we outlined the processes involved in transforming online data into a multidimensional format. This makes demands of the system model, both in terms of its structural makeup, and in the functions that operate around the model. In this section, we examine in finer detail, the components involved in this transformation, in terms of the requirements that they place on the system. In other words, these present the functional requirements for the system's canonical model.

1. The first requirement is the ability to parse and interpret data sources of various formats and map the data from different formats into a usable, intermediary format. This requirement also involves the ability to identify and extract structures and items which would be of interest in a data mart, specifically dimensions and measures. This is effectively a form of enrichment where a process seeks to identify multidimensional constructs and annotates the data stream accordingly.
2. The next requirement is to restructure the original source data so as to prepare data sources for transformation into a multidimensional format.
3. Finally, there is a requirement to perform the actual transformation and record the process as a series of mappings which can be used later in materialising data marts.

In the remainder of this section, we will examine these requirements in detail, to serve as a specification for our system model, both in terms of its structure and functional components.

4.1.1 Requirement 1: Enriching the Data Source

This requires a process to analyse the content of the data source, make determinations about what data values, and the relations between data items mean, and enrich the original data source with annotations that can be used in subsequent transformations.

4.1.1.1 Data Source Analysis and Annotation

A process is needed to parse a data stream and produce an enriched graph representation of the stream. It takes input in the form of a sample data stream, and produces an annotated graph representing the data stream.

Each node in the graph represents a single item in the data stream, these nodes are annotated with the datatype of the item and a generated query which can be used to extract this data from its source. Edges in the graph represent relationships between items in the data source. Each edge is annotated with the cardinality of the relationship. There are three possible types: *ONE_TO_ONE*, *ONE_TO_MANY* and *MANY_TO_MANY*.

It is important to note at this point, that any strict relations such as 1-n, or n-n are classed as `ONE_TO_MANY` and `MANY_TO_MANY` respectively. This is because such relationships would not be apparent in the data stream and would require a schema of the data stream in order to be detected. Additionally, such a constraint has no impact on the algorithms and procedures used to create the data mart. This step is required in order to abstract the source data streams from their individual formats and structures.

Annotated graph construction will require multiple stream parsers as each data format (XML, JSON, CSV etc...) require different mechanisms to be read and understood. The difficulties in this process mainly revolve around resolving the differences between data formats and structures. For example, consider a ONE_TO_MANY relationship. Using XML data, this relationship would be represented through the use of repeating sub-elements inside the document. On the other hand with JSON data, the array datatype would be used to convey such information.

HTML data streams prove to be the hardest to parse and understand and have the highest degree of failure among all datatypes. At a high level, this is due to the decoupling between the way in which data is displayed, and the way it is internally structured.

Additionally, HTML data may contain errors which would not be present in other data formats, most common of which is the incorrect use of HTML tags, resulting in a malformed document.

Due to the large degree of representations HTML data may use, the HTML parser seeks table elements and scans them for structured information. Any more detailed parsing would require writing custom queries based on a specific HTML document, negating the benefits of an automatic process.

4.1.1.2 Node Classification

While the previous process provides an annotated graph, it will not contain enough information in order to construct a data mart. A `Node Classification` step should examine each node in the annotated graph and provides each node with a classification to indicate which role it may play in a data mart. Preliminary analysis of a large number of streaming data sources indicated five possible classifications:

- Potential Measure
- Dimension
- Dimension-attribute
- Subdimension
- Container

These property classifications are used in the following steps in order to identify facts, dimensions and measures and construct a data mart. Additionally at this stage, some nodes may be classified as redundant (Container) and should be removed during the next step. The difficulties in this process will revolve around identifying how graph topologies and graph subsets should be transformed into a data mart.
The classification of dimensions, their attributes and sub-dimensions relies heavily on examining the structure of the graph and the cardinality of the edges between nodes. For example, for any node classified as a dimension, all 1-1 nodes this node is liked to will automatically be classified as `dimension_attribute` nodes. However, a dimension which holds a *1-m* relationship with another dimension implies a dimensional hierarchy, and as such should become a sub-dimension.

4.1.2 Requirement 2: Graph Restructuring

The previous step should produce an annotated graph with each node having been assigned a classification. Some nodes produced by this step may be classified as

redundant. A process is needed to examine these classifications, remove nodes classified as redundant and restructure the graph.
Redundant nodes can take many forms. For example, XML data creates redundant nodes when representing ONE_TO_MANY relationships. This is because the markup dictates that a container tag must be used in order to manage this relationship. Using JSON data, the same problem is encountered when dealing with array datatypes. These container nodes serve no purpose in the graph, as edges denote cardinality and as such they can be removed, once the relationship information has been recorded.
Graph restructuring removes each node which was classified as redundant and relinks the nodes descendants to its parent nodes. However, graph restructuring may create new redundant nodes. This can occur if a parent node of a node classified as a container holds a 1-1 relationship to the container node. Once the container node is removed, the parent node will inherit the `1-m` relationship held by the container node. This in turn, provides the parent node will all conditions required to be classified as a container. Figure 4.1 provides an illustration showing how this process can occur. In Step 1, the node `C` has been identified as a *container* and is removed. The node B previously classified as a Dimension $\langle DIM \rangle$ then inherits the *1-m* relationship to node D. This in turn, means that B has all of the conditions required to be re-classified as a container (Step 2).
Finally, in Step 3 the new container node `B` is also removed, leading to node `A` inheriting the `1-m` relationship. After re-classification, the node `A` does not satisfy the requirements to be classified as a measure and graph restructuring is completed. In order to overcome this, the process re-classifies all nodes after restructuring and proceeds to restructure if any redundant nodes are found. This process should repeat until all redundant nodes are removed from the graph. The output of this process is a restructured annotated graph with node classifications.

4.1.2.1 Dimension and Measure Identification.

Once the graph has been annotated, all nodes have been classified and all redundancies removed, the next step is to identify the constituent components of a (star)

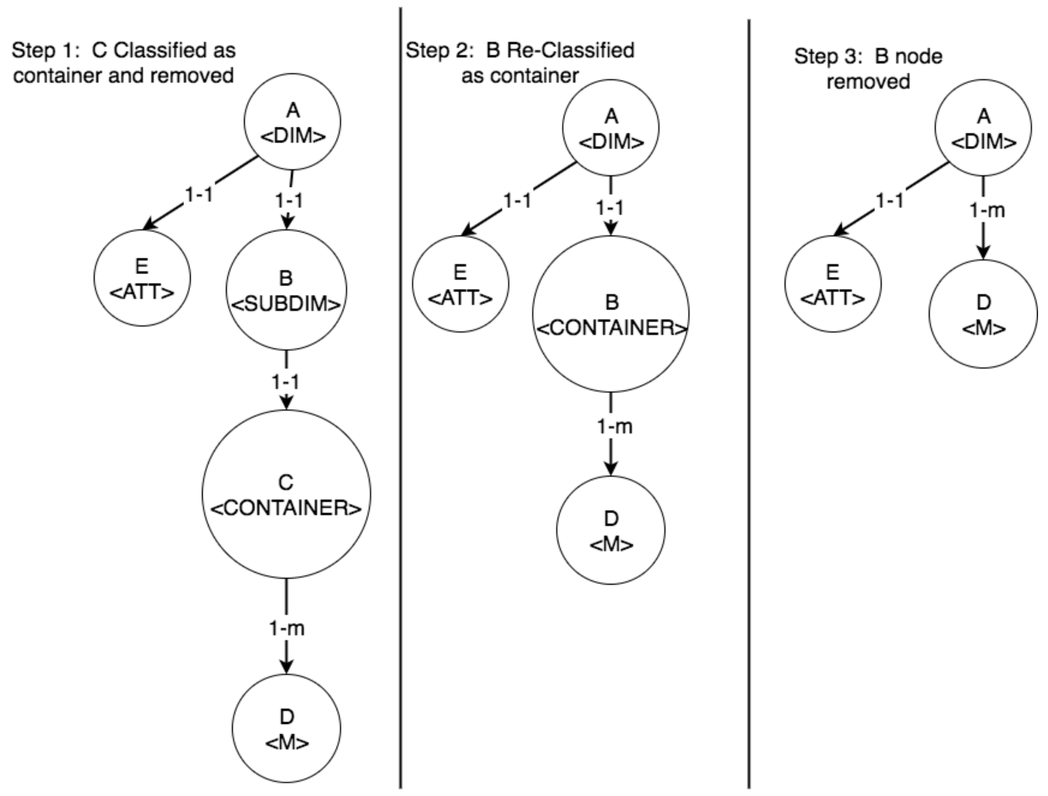


Figure 4.1: Example of graph restructuring and creating new container nodes.

schema (dimensions and measures) from the graph. This process should examine the graph structure, the classifications of nodes, and their datatype in order to create dimensions and measures.

The assumption of this process is that all numeric datatypes are *measures*. In the event this process incorrectly identifies an attribute as a measure, the type mapping phase of the integration process will re-assign the attribute to its correct type.

There will be difficulties in this process. Firstly, the construction of dimensions can either be a simple or very complex process. For example, attributes which are date datatypes are automatically considered to be part of the date dimension. While others may prove more complex, constituting sub-dimensions with multiple attributes. The second difficulty is in constructing the required facts. As measures may appear at any place in the graph, measures may depend on different dimensions and would constitute different 'facts' with shared dimensions.

Ultimately, this process constructs dimensions, measures and facts from the graph.

Once all facts, dimensions and measures have been identified and their structure created, the graph now represents a data mart. However, mapping rules should be generated during graph construction in order to populate a data mart.

4.1.3 Requirement 3: Generating Mappings

The overall process should construct mapping rules which can be used to extract the data in the data stream to materialise a data mart. One design might see a process that examines the dimensions and measures previously identified and constructs a JSON file specifying how the data should be extracted and stored.
Mapping rules should be derived from the queries generated for each node during the `Data source analysis and annotations` step. Once the mapping rules are generated, they should be stored alongside the data mart in the metabase. A design requirement should insist that mapping rules be designed in such a way that they can be used to extract data from a data stream without the need for the data mart (or StarGraph presented in the following section). This will ensure that they may be used to complement other systems. For example, these rules may be extracted and implemented within a traditional ETL process.

4.2 The StarGraph Model

As explained in the previous section, our canonical model must be capable of representing multidimensional data: the facts, dimensions and relationships that conceptually represent a star schema. Our canonical model which we have termed the *StarGraph*, is a graph based model with constructs to represent facts and dimensional hierarchies that are captured from semi-structured data sources. In the first part of this section, we focus on the properties of the StarGraph and in the second part, we focus on its functionality or *behaviour*.

4.2.1 StarGraph Properties

Definition 4.1. StarGraph.
A StarGraph is composed of a set of nodes N and a set of edges E.

Definition 4.2. StarGraph-Node.
A StarGraph Node $n \in N$ is a four-tuple $n = \langle name, class, source, dType \rangle$; where name is the name of the node; class indicates the type of node; source specifies the location of the data item in the schema; and dType is the datatype of the defined node.

In Definition 4.2, a StarGraph node is defined. The *source* attribute can be XPath [18] (for XML/HTML data) or dot-notation for JSON data. For the *class* attribute, there are five possible types:

- `Dimension` marks a node which is the beginning of a dimension.
- `dimension_attribute` is a marker denoting that the node in question is an attribute of the parent dimension node.
- `container` indicates the node is an instance containing other nodes.
- `measure` indicates that this node is a measure.
- `subdimension` indicates the node is a subdimension

Definition 4.3. StarGraph-Edge.
A StarGraph Edge is a three-tuple $E = \langle X, Y, REL \rangle$ where: $X, Y \in N$; REL is a type denoting the relationship which exists between the nodes X and Y.

In Definition 4.3, we see the definition for an Edge as a triple connecting 2 nodes, with a relationship type between nodes. The possible values for *REL* are: *1-1*, *1-n* and *m-n*. The relationship type is obtained from examining the cardinality between attributes such as the use on an array datatype in a JSON data source.

4.2.2 StarGraph Functions

StarGraph functionality is driven by the requirements specified in the previous section. There are six main functions which define the behaviour of the StarGraph. These functions manage all of the operations involved in constructing the StarGraph, binding StarGraph elements to the original source data; and enabling materialisation of StarGraph instances. In later chapters, we will show how a special form of the StarGraph can represent a data mart constructed from multiple sources. For this chapter, we will focus on the core aspects of StarGraph construction from a single data source. The six functions are:

- `GraphAnnotate`: analysis and markup of the initial graph.
- `ClassifyNode`: node analysis and classification.
- `GraphEdit`: different graph update operations.
- `ClassifyDim`: as part of graph analysis, detect and classify dimensions.
- `ClassifyMeasure`: as part of graph analysis, detect and classify measures.
- `Materialise`: populate or update the StarGraph.

4.2.2.1 The `GraphAnnotate` function

The role of `GraphAnnotate` is to examine a data source and construct a graph representation of the data source which has been annotated with valuable metadata about the source. This is necessary for two reasons; the first is that the graph structure decouples source data from its original format, the second reason is to provide metadata to the data source. When a data source is introduced to the system, it is initially transformed into a typical graph format representing the source. Each node captures a point of information: nodes will either become nodes in the StarGraph, or be deemed unnecessary and removed. The edges between nodes are derived from the structure of the data source (sub-elements in XML for example). As part of the metadata obtained during this process, each node in the graph is annotated with a query which can be used to extract information from the data source. This query is created by scanning the data source. If the data source is XML, or HTML , XPath queries are used. If the data source is JSON, dot-notation is used, and if the data source is CSV the node contains a reference to the column number.

Each edge in the graph is annotated with the cardinality of the relationship. Once the queries for each node have been constructed, datatypes are assigned to each node. The datatypes are discovered by using the queries generated previously to extract the data found in the source. The datatypes applied are as follows:

- STRING - This type represents any textual information.
- OBJECT - This type is assigned to nodes which themselves do not contain any information, but instead link to other nodes. These nodes become Dimensions in the generated StarGraph.
- NUMBER - This type represents any *numerical* datatype. These nodes represent potential measures.
- DATE - This type represents *date* datatypes.
- GEO - This type is reserved for nodes containing GEO dimensions.

This process produces an annotated graph representation 'as-is' from the data stream. However, in order to identify the required dimensions and measures, each node must be classified.

An example of the graph generated by the aim_1 dataset from Case Study 1 (section 3.4.1) is shown in Figure 4.2. The purpose of the box around the three blank nodes is to simplify the visualisation, where each edge to the box represents an edge to *every* node inside the box.

Algorithm 4.1 Algorithm for Assigning Node Types

```
function setNodeType(node)
    switch node do
        case isContainer(node)
            node.setType(Container)
        case isDimension(node.parent)
            if node.relType="one" then
                node.setType(Dimension-Attribute)
            else
                node.setType(Subdimension)
            end if
        case isNumeric(node)
            node.setType(p-measure)
        case Default
            node.setType(Dimension)
    for child in node do
        setNodeType(child)
    end for
end function
```

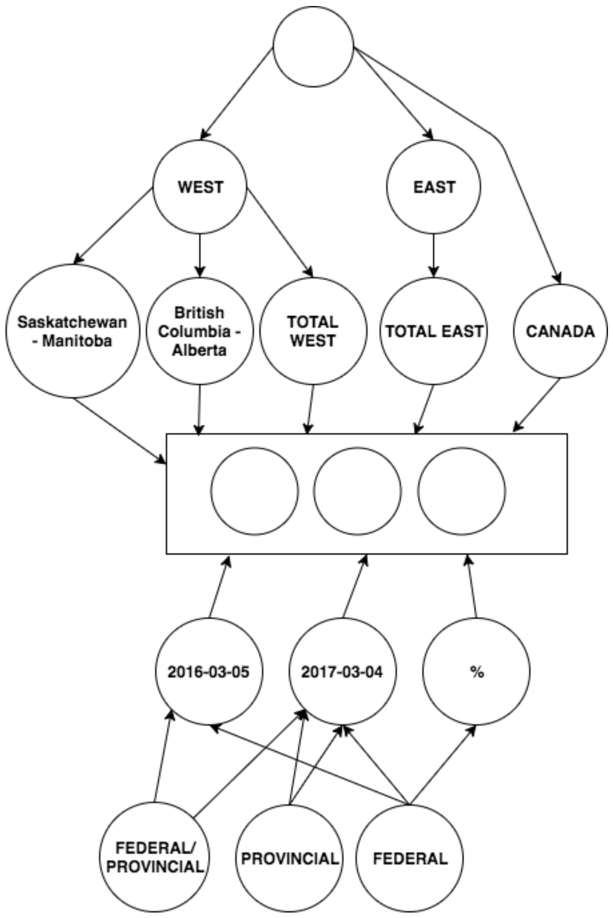


Figure 4.2: Graph representing the `aim_1` dataset.

Additionally, a sample of the mappings produced at this stage are provided in JSON format in Example 4.2.1. These mappings are very simple, consisting of; the node name, the generated source query (XPath, JSON, CSV), the data type of the node and a set of relationships connecting this node to other nodes. A relationship in this instance is represented as a tuple $\langle name, reltype \rangle$ where `name`, is the name of the related property, and `type` is the cardinality of the relationship.

Example 4.2.1. Sample mappings produced by the `GraphAnnotate` function.

```
[{
 name:"",
 src:"//table/tbody/tr[1]/th[1]",
 type:"STRING",
 relationships:[{
    name:"WEST",
    type:"one-to-one"
 },...]
},
{name:"WEST",
   src:"//table/tbody/tr[3]/th[1]",
    srcformat:"HTML",
    type:"STRING",
    relationships:[
        {name:"British Columbia - Alberta",
         reltype:"one-to-one"},...]
},
{ name:"Federal/Provincial",
   src:"//table/tbody/tr[1]/th[2]",
    srcformat:"HTML",
    type:"STRING",
    relationships:[
        {name:"2016-03-05",
         reltype:"one-to-one"},...]
}...]
```

4.2.2.2 The `ClassifyNode` function

Once datatypes have been assigned to nodes, the next step of the process is node classification. The role of `ClassifyNode` is to examine each node and provide a classification related to a nodes role within a data source. This is necessary to determine the relevance of each node and determine its potential place within a data mart. At the outset, the graph is scanned node by node, using the classification logic shown in Algorithm 4.1.

The `ClassifyNode` function examines a nodes information (name, data type) and adjoining relationships to determine what class the node belongs to.

The function first tests to see whether or not a node is a container node and as such is redundant. The function `isContainer` on line 3 of Algorithm 4.1 is specified in Algorithm 4.2. This algorithm (4.2) examines a node's direct relationships to adjoining nodes within the graph in order to determine whether the node is redundant (a container node). The conditions for a container node are as follows:

1. If the node is a root node, it must have a name
2. A node may hold an edge to a single child node only, as shown in Line 2 in Algorithm 4.2.
3. The edge between these two nodes must have a *one-to-many* cardinality, as shown in Line 4 in Algorithm 4.2.
4. The child node may *not* hold a edge to any other parent node, as shown in line 6 in Algorithm 4.2.

If all of these conditions are true, then the node is classified as a *container* node. Lines 5-10 of Algorithm 4.1 determine whether a node is an *attribute* of a dimension or a full sub-dimension itself. This is determined by examining the cardinality stored in the annotated edge. If a node holds a *1-1* relationship with a Dimension node, it is regarded as a dimension attribute. If they hold a *one-to-many* relationship, the node is considered a sub-dimension.

Lines 11-12 of Algorithm 4.1 assign each numeric data type with the *p*-measure classification. These nodes are considered to be potential measures within the data mart. In the event that any of the previous classifications are not met, the algorithm assumes the node is a *dimension* node as shown in line 14 of Algorithm 4.1.

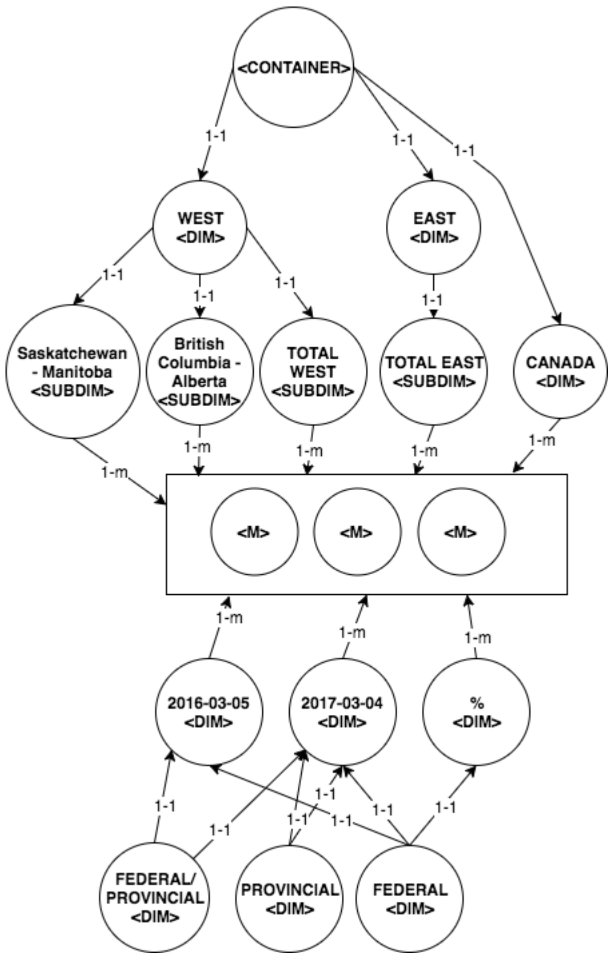


Figure 4.3: `aim_1` dataset after classification

Fig 4.3 presents the `aim_1` graph after node classification. Classifications are expressed using '⟨' and '⟩' brackets. Nodes expressed as $\langle M \rangle$ are classified as potential measures. $\langle DIM \rangle$ refers to dimensions, $\langle SUBDIM \rangle$ refers to sub-dimensions and $\langle CONTAINER \rangle$ identifies nodes which are candidates for removal.

Algorithm 4.2 Algorithm for Detecting Container Elements

```
1: function isContainer(node)
2:     if node.isRoot AND isEmpty(node.name) then
3:         return True
4:     else if node.children.size!=1 then
5:         return False
6:     else if node.children.relType!=one-to-many then
7:         return False
8:     else if graph.instance(node.children)>1 then
9:         return False
10:    end if
11:    return True
12: end function
```

These classifications extend the mappings provided in Example 4.2.1 by adding a classification property to each node entry. An example of these updated mappings can be seen in Example 4.2.2.

Example 4.2.2. Example mappings produced by the `ClassifyNode` function.

```
[{name:"",
 src:"//table/tbody/tr[1]/th[1]",
 type:"STRING",
 classification:"CONTAINER"
 relationships:[{
    name:"WEST",
    type:"one-to-one"
 },...]
},
{name:"WEST",
   src:"//table/tbody/tr[3]/th[1]",
    srcformat:"HTML",
    type:"STRING",
    classification:"DIM"
    relationships:[
        {name:"British Columbia - Alberta",
         reltype:"one-to-one"},...]
},
{ name:"Federal/Provincial",
   src:"//table/tbody/tr[1]/th[2]",
    srcformat:"HTML",
    type:"STRING",
    classification:"DIM"
    relationships:[
        {name:"2016-03-05",
         reltype:"one-to-one"},...]
}...]
```

4.2.2.3 The `GraphEdit` function

Graph restructuring is a multi-phase process, the goal of which is to remove all container elements from a StarGraph. The role of `GraphEdit` is to restructure the graph (change the set of edges) upon removing redundant nodes. This is necessary to simplify the graphs structure and ensures that redundancies are not present within the data mart.

The first step is the removal of all *container* elements. This is achieved by removing all nodes annotated as containers and relinking the child edges from the container node to the containers' parent nodes. The logic is shown in Algorithm. 4.3. Lines 4-9 within Algorithm 4.3 remove a container node from the graph, and re-link any relationships between parent and children node which may have been broken by removing the container node.

Once this process has been completed, all nodes in the graph are reclassified to ensure no new container nodes have been created through the restructuring process. If new container elements are detected, this process repeats until no container elements are found in the graph. In the example Figure 4.4 the unnamed node previously identified as a container (figure 4.3) was removed.

Algorithm 4.3 Algorithm for graph restructuring

```
1: function removeContainer(node)
2:     parentNodes = node.parents
3:     childnodes = node.children
4:     for pNode in parentNodes do
5:         for cNode in chlidNodes do
6:             pNode.addChild(cNode)
7:             cNode.addParent(pNode)
8:         end for
9:         p.removeChild(node)
10:    end for
11:    for c in childNodes do
12:        c.removeParent(node)
13:    end for
14: end function
```

This process updates the mappings produced, by removing nodes, and updating the relationships for all nodes affected by the removal. In the case of the mappings

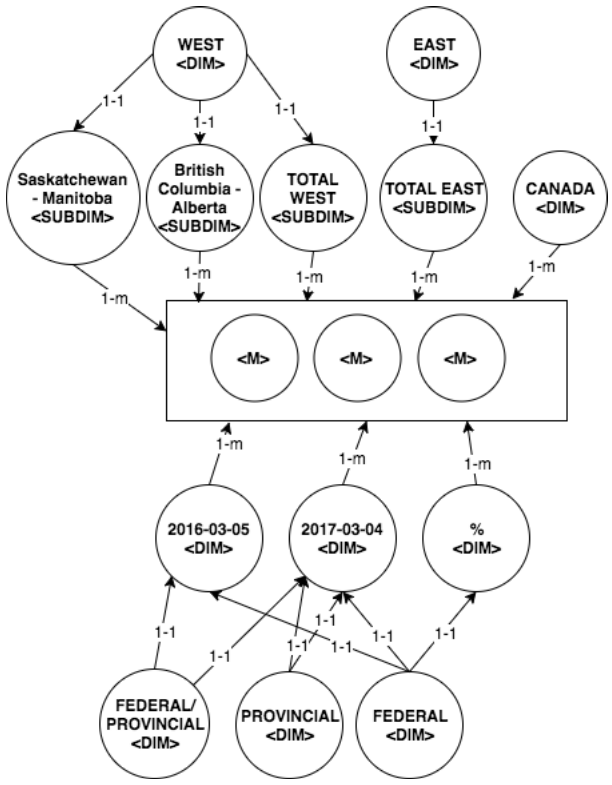


Figure 4.4: `aim_1` dataset with containers removed

shown in 4.2.2, the only change is the removal off all references to the blank node. This can be seen in Example 4.2.3.

Example 4.2.3. Example mappings produced by the `GraphEdit` function.

```
[{name:"WEST",
   src:"//table/tbody/tr[3]/th[1]",
    srcformat:"HTML",
    type:"STRING",
    classification:"DIM"
    relationships:[
        {name:"British Columbia - Alberta",
         reltype:"one-to-one"},...]
},
{ name:"Federal/Provincial",
   src:"//table/tbody/tr[1]/th[2]",
    srcformat:"HTML",
    type:"STRING",
    classification:"DIM"
    relationships:[
        {name:"2016-03-05",
         reltype:"one-to-one"},...]
}...]
```

4.2.2.4 The `ClassifyDim` and `ClassifyMeasure` functions

The next step in the StarGraph process is to identify all measures and dimensions. The role of these functions is to determine the *structure* of data mart dimensions and identify the measures within the data source. This is a crucial step in constructing the StarGraph schema that provides the data mart with its multidimensional properties.

All items which have previously been classified as *candidate* measures are evaluated at this stage. In the event they are determined *not* to be measures, they are re-classified as `dimension-attribute` nodes and become part of a dimension object. The algorithm for constructing dimensions is shown in Algorithm 4.4. Line 6-7 of the Algorithm collects all child nodes for a dimension which have been identified as properties of that dimension. Similarly, lines 8-10 examine a dimension's relationship with sub-dimensions and construct a hierarchy within the dimension.

When constructing the fact object, the dimensions which describe the fact are those through which the measure is either linked directly or transitively. In the event a dimension does not have a specified key, a primary-foreign relationship will be created for it. This relationship will be generated using attribute(s) captured on both sides of the relationship: if two dimension objects contain the same values for their attributes, they are considered to be of the same dimension.

Measures are grouped into the same fact based on their shared relationships. That is, if two nodes classified as measures contain edges to the same dimension nodes, they will share the same fact. This is determined by examining the edges connecting each measure to dimensions and grouping them into *dependency sets* (Ex 4.4). Similar to functional dependencies, the dependency sets identify which dimensions are required for each measure. However, in addition to the structure of the graph, the cardinality of the relationships must also be considered in order to determine which measures can occupy the *same* facts.

Definition 4.4. Dependency Set.

A Dependency Set $DS = \langle (d_1, c_1), (d_2, c_2)...(d_n, c_n) \rangle$ is a set of tuples $(d, c) \in DS$ where d is the dimension to which a measure is connected, and c is the cardinality *of that connecting edge.*

Our method makes the important assumption that measures that have the same dependency set S are considered to be of the same fact: they share the same dependencies. There may be multiple facts created from a single data source but these facts will share dimensions in the same fashion as a constellation schema.
Once all dependency sets have been created, the final step is to construct the individual facts, as at this stage, we have the set of all measures and their dependencies to dimensions (i.e. all the required properties to construct a fact).

Algorithm 4.4 Algorithm for dimension construction

```
1: function constructDimension(node)
2:     attributes = []
3:     subdims = []
4:     for n in node.children do
5:         if n.classification!="measure" then
6:             if n.edgeType=="1-1" then
7:                 attributes.add(n)
8:             else if n.edgeType=="1-m" then
9:                 sub =constructDimension(n)
10:                subdims.add(sub)
11:            end if
12:        end if
13:    end forreturn
14:     Dimension(node, attributes, subdims)
15: end function
```

These properties enrich existing mappings as they now capture the constituent parts of dimensions and a set of measures. These additions can be seen in Example 4.2.4.

Example 4.2.4. Example mappings produced by `Classify` functions.

```
{nodes:[{name:"WEST",
    src:"//table/tbody/tr[3]/th[1]",
     srcformat:"HTML",
     type:"STRING",
     classification:"DIM"
     relationships:[
         {name:"British Columbia - Alberta",
          reltype:"one-to-one"},...],
     attributes:["",..],
     subdimensions:["TOTAL WEST"]
},
{ name:"Federal/Provincial",
    src:"//table/tbody/tr[1]/th[2]",
     srcformat:"HTML",
     type:"STRING",
     classification:"DIM"
     relationships:[
         {name:"2016-03-05",
          reltype:"one-to-one"},...],
     attributes:[],
     subdimensions:["2016-03-05"]
 }...],
 dimensions:["WEST","FEDERAL/PROVINCIAL",...],
 measures:["","%",...]}
```

4.2.2.5 The `Materialise` function

The role of the Materialise function is to populate the data mart by using the mappings to a data source. As dimensions, subdimensions, measures and facts have been previously generated, the first step in materialise is to simply render these as JSON.

Example 4.2.5 details the schema generated by the StarGraph. The document contains a set of dimensions, measures and facts which have been detected within the StarGraph by the `ClassifyDim` and `ClassifyMeasure` functions. For example, the dimension called '`West`' contains no attributes, but does contain a subdimension called '`British Columbia - Alberta`'. This document constitutes the target schema for the mappings.

Example 4.2.5. Target schema for `aim_1`.

```
{
    name:"aim_1",
    type:"HTML",
    source:"http://aimis-simia.agr.gc.ca/..."
    dimensions:[
        {
          name:"West",
          attributes:[],
          subdimensions:[
            {
                name:"British Columbia - Alberta",
                 attributes:[],
                 subdimensions:[]
            },
            ...]
        },
        {
            name:"Federal/Provincial",
            attributes:[],
            subdimensions:[...]
        }...
    ],
    measures:[
        {
            name:"",
        },
    ,    ...]
    facts:[
        {
            dimensions:["West"...],
            measures:["",....]
        }]}
```

Definition 4.5. Mapping.

Each mapping is of the form $(\langle query \rangle) \rightarrow \langle schema\_position \rangle$ *Where* `query` *is the source query for a node within a StarGraph and* `schema_position` *is where the value for the node should be stored within the target schema. This is represented using JSON dot notation.*

Example 4.2.6 illustrates a subset of the mappings, containing a source query and their corresponding place within the target schema, with the definition of a mapping being presented in Definition 4.5.

Example 4.2.6. Mappings for `aim_1`.

```
("//table[1]/tbody/tr[3]/th[1]") -> West,
("//table[1]/tbody/tr[3]/th[2]")-> West.subdimensions
            ["British Columbia - Alberta"]
("//table[1]/tbody/tr[3]/td")->measures[""]
```

These mappings can then be used to extract data from its source and place them within the target schema. As the majority of data sources available on the web are in XML, JSON, HTML or CSV formats, our research focused on these standard formats. However, this section presents the generic `Materialise` function and it requires customisation depending on the structure of the underlying data source. In the following section, we provide a detailed description of the process for managing heterogeneous data sources.

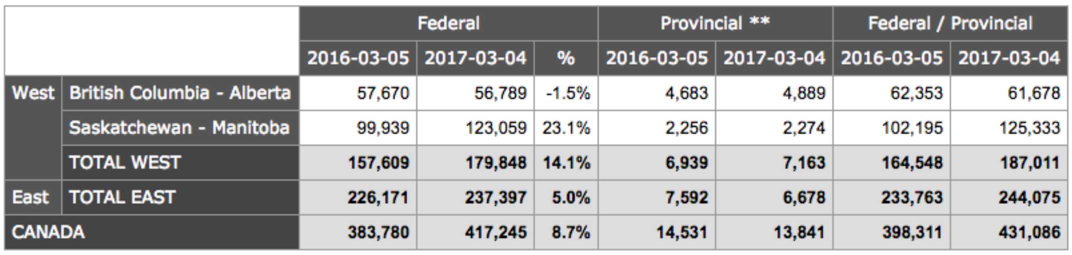
4.3 Materialisation Wrappers

The purpose of the wrappers is to remove heterogeneity between data source formats. This necessitates a wrapper data model to which each individual wrapper conforms. Our wrapper is called an *Annotated Tuple Set* (ATS) shown in Definition 4.6. An annotated tuple set represents source data as a series of tuples, annotated with metadata such as the name of the attribute and the source query. Each wrapper examines a data source and produces an annotated tuple set. The mappings are then applied to this tuple set in order to extract the data.

Definition 4.6. Annotated Tuple Set.
An Annotated Tuple Set is of the form:
$ATS = \langle A_1....A_n \rangle$ *Where* A *represents a list annotated attributes, corresponding to a particular data instance (e.g. a row in a CSV file). This is of the form:* $\langle (n, s, v)_1...(n, s, v)_n \rangle$*, ehere* n *is the name of an attribute,* s *is the source query and* v *is the value for the attribute.*

4.3.1 HTML Data Streams

HTML data streams such as `aim_1` prove the hardest to transform into an ATS. While wrappers may use knowledge about the rigid structure of other formats, the wide array of possibilities used to represent HTML data poses problems.
HTML data makes use of both content (the HTML markup) and layout (CSS) to convey information to the user. This means that multiple combinations of content and layout may appear visually similar to a user, however they may be quite different in their underlying implementations. Despite the fact that a specification for the tags involved in the markup exist, enforcement of this structure is rare. Due to the large degree of possibilities, the wrapper limits itself to searching for structured tabular data. In short, it examines a page for `table` elements, and parses them. This works well for data sources which use these elements correctly, for example `b_1`.



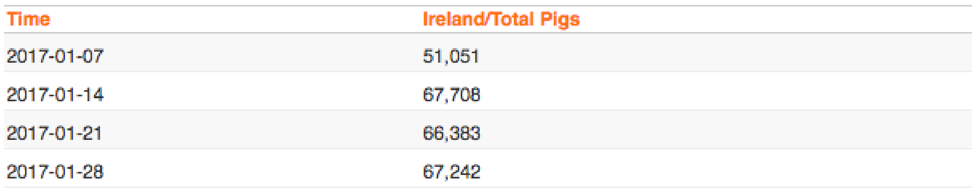
		Federal			Provincial **		Federal / Provincial	
		2016-03-05	2017-03-04	%	2016-03-05	2017-03-04	2016-03-05	2017-03-04
West	British Columbia - Alberta	57,670	56,789	-1.5%	4,683	4,889	62,353	61,678
	Saskatchewan - Manitoba	99,939	123,059	23.1%	2,256	2,274	102,195	125,333
	TOTAL WEST	157,609	179,848	14.1%	6,939	7,163	164,548	187,011
East	TOTAL EAST	226,171	237,397	5.0%	7,592	6,678	233,763	244,075
CANADA		383,780	417,245	8.7%	14,531	13,841	398,311	431,086

Figure 4.5: Sample of `aimis_1` data source

The process for transforming a HTML table into an ATS first requires constructing a matrix representation of the data source. Tools such as Pandas [48] can be used to accomplish this first task, consuming a HTML table and constructing a Data Frame from it. Once the matrix representation is complete, the data source is examined to determine the structure of the HTML table. If the table contains only column headers, the table can be scanned row-by-row in order to flatten it, similar to a CSV file. However if the source contains row *and* column headers, or *multiple* row and column headers, the source must be traversed by first flattening the row and column headers and then, iterating through the matrix in a $row * column$ fashion. For example, Figure 4.5 presents the raw table shown by the `aim_1` dataset. Once the wrapper has been applied, the ATS can be seen in Example 4.3.1.

Example 4.3.1. Annotated tuple set for `aim_1`.

```
({name:Federal, src:"...", val:"Federal"},
{name:"2016-03-05", src:"...",
val:"2016-03-05"},
{name:"West",src:"...", val:"West"},
{name:"Total West",src:"...",
val"Total West"},
{name:"",src:"...",val:"226,171"}),
...
```



Time	Ireland/Total Pigs
2017-01-07	51,051
2017-01-14	67,708
2017-01-21	66,383
2017-01-28	67,242

Figure 4.6: Sample of the `b_1` data source

The StarGraph produced by this dataset can be seen in Fig 4.4, while simpler HTML structures as shown in the `b_1` dataset (shown in Figure 4.6) produce a series of tuples in the form shown in Example 4.3.2. Each item in the tuple list in this instance contains three attributes: the column name; the source query (the row number); and the value of the *td* element under the `val` header. These examples are simple tables, containing a date and a value, which produces a StarGraph consisting of a single measure and a Date dimension as shown in Figure 4.7.

Example 4.3.2. Annotated tuple set for `b_1`.

```
({name:Time, src:"...", val:"2017-01-07"},
{name:"Ireland/Total Pigs",src:"...",val:"51,051"}),...
```

This graph produces rather simple mappings, consisting of one dimension and one

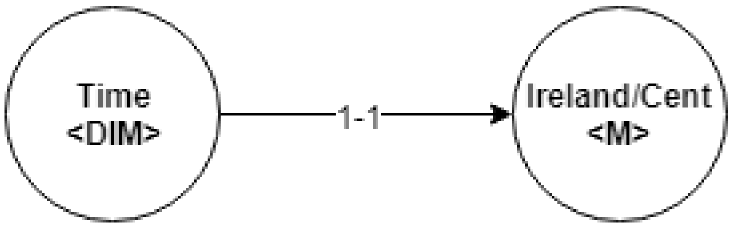


Figure 4.7: Sample of simple `b_1` StarGraph

measure and thus, renders a fact object as a tuple. The target schema produced by the `b_1` StarGraph transformation can be seen in Example 4.3.3 and the mappings are shown in Example 4.3.4.

Example 4.3.3. Target Schema produced from `b_1` dataset.

```
{
    "name" : "b_1",
    "type" : "HTML",
    "source" :"https://www.bordbia.ie/ind...",
    dimensions : [
    {
        "name" : "Time",
        "type" : "Date",
        "attributes":[],
        "src" : "//table/tbody/tr/td[1]"
    }
    ],
    measures : [
    {
        "name" : "Ireland/Total Pigs",
        "type" : "MEASURE",
        "src" : "//table/tbody/tr/td[2]"
    }
    ],
    facts:[{
        dimensions:["Time"],
        measures:["Ireland/Total Pigs"]
    }]
}
```

Example 4.3.4. Mappings produced from `b_1` dataset.

("//table/tbody/tr/td[1]")→ Time

("//table/tbody/tr/td[2]")→ "Ireland/Total Pigs"

A complete target schema and mappings for the `aim_1` StarGraph are provided in Appendix A.

4.3.2 XML and JSON Data Streams

The process to construct a tuple list from XML or JSON data is relatively simple, requiring a single traversal of the structure. This essentially flattens the data source which in normal circumstances, results in the loss of metadata or semantics captured in the XML or JSON documents. However, with our approach, this is not lost as the StarGraph captures this metadata. Thus, information such as the cardinality between nodes and the structure of dimensions and measures as extracted from the documents structure are retained in the StarGraph.

Algorithm 4.5 presents a high level overview of how the traversal takes place. The function `getSource` at line 7 extracts the data source query from the raw data source. It then traverses a data source recursively, at each level extracting a nodes name, source path and value. These constitute a set of nodes found at a level within the data source structure. This set is then passed to the nodes parent element, which is then used to formulate another set of values. The process repeats until the entire data source has been parsed leading to a set of tuples representing the flattened XML or JSON data source.

Algorithm 4.5 High level algorithm for XML and JSON traversal

```
 1: function TRAVERSESTRUCTURE(nodelist)
 2:     end_set = []
 3:     if NODELIST(size)==0 then
 4:         return []
 5:     else
 6:         for node in nodelist do
 7:             node_structure = {node.name, NODE.GETSOURCE, node.value}
 8:             if NODE.HASCHILDREN then
 9:                 child_set = TRAVERSESTRUCTURE(node.children)
10:                 for item in child_set do
11:                     END_SET.APPEND(node_structure + item)
12:                 end for
13:             else
14:                 END_SET.APPEND(node_structure)
15:             end if
16:         end for
17:     end if
18:     return end_set
19: end function
```

There is only one XML data stream presented in the case studies, the `imf` dataset. This dataset has a rich structure, containing many nodes, representing a series of elements for each possible currency, listing its values as SDRs (Special Drawing Rights). A subset of the `imf` dataset can be seen in Example 4.3.5, and the corresponding tuple set can be seen in Example 4.3.6.

Example 4.3.5. Sample of the `imf` dataset.

```
<EXCHANGE_RATE_REPORT>
  <EFFECTIVE_DATE VALUE="09-Nov-2017">
    <RATE_VALUE CURRENCY_CODE="ALGERIAN DINAR" ISO_CHAR_CODE="DZD">
        0.00617914
    </RATE_VALUE>
    <RATE_VALUE CURRENCY_CODE="Australian dollar" ISO_CHAR_CODE="AUD">
        0.547397
    </RATE_VALUE>
 </EFFECTIVE_DATE>
</EXCHANGE_RATE_REPORT>
```

Example 4.3.6. Sample tupleset for `imf` dataset.

```
({name:"EXCHANGE_RATE_REPORT",src:"/",val:"EXCHANGE_RATE_REPORT"},
{name:"EFFECTIVE_DATE",src:"..",
val:"EFFECTIVE_DATE"},
{name:"VALUE",src:"..",
val:"09-Nov-2017"},
 {name:"CURRENCY_CODE",src:"..",
val:"ALGERIAN DINAR"}
 {name:"ISO_CHAR_CODE",src:"..",
val:"DZD"},
 {name:"RATE_VALUE",src:"..",
val:" 0.00617914"}),
...
```

Figure 4.8 shows the StarGraph created from the `imf` dataset which produces a reasonably complex StarGraph. However, many of the nodes found in this graph are redundant. For example, the dimension named `USER_SELECTIONS` contains only metadata about the query used to generate the data. Similarly the node `DISCLAIMER` is a legal disclaimer stored inside an XML node. These values provide no information to the data mart. However, as they contain data, the StarGraph process assumes they are of use and will be populated in a data mart unless a user specifies they should be removed. Approaches to dealing with these types of redundancies are discussed in chapter 6.

The one measure found is unnamed and resides as a child of the `RATE_VALUE` node, with the single fact holding an edge to all dimensions.

Example 4.3.7. Target schema produced by the `imf` StarGraph.

```
    {
        "name" : "imf",
        "type" : "XML",
         source:"http://www.imf.org/external/...",
        dimensions : [
        {
            "name" : "ExchangeRateReport",
            "type" : "Object",
            "src" : "//ExchangeRateReport",
            "attributes":[{
                "name":"ReportName",
                "type":"String",
                "src":"//ExchangeRateReport/ReportName/text()"
            },
            :::;]
        },:.
        ],
        measures : [
        {
            "name" : "RateValue",
            "type" : "MEASURE",
            "src" : "//ExchangeRateReport/EffectiveDate/RateValue/text()"
        },:.
        ],
        facts:[{
            dimensions:["ExchangeRateReport","UserSelections,....],
            measures:["RateValue/text()"]
        }]
    }
```

Example 4.3.8. Mappings for `imf` StarGraph.

("//ExchangeRateReport/ReportName/text()")→ExchangeRateReport["ReportName"]

("//ExchangeRateReport/EffectiveDate/RateValue/text()")→RateValue

A sample of the target schema produced by this StarGraph is shown in Example 4.3.7, with sample mappings in Example 4.3.7, and a full version provided in Appendix B.

As there are no JSON data sources in the agri case studies, we present a JSON data source from one of our earlier research collaborations from the Smart Cities domain. Transport Infrastructure Ireland [1] provide a JSON data set which details the travel times between all junctions on all motorways across Ireland. This dataset updates every five minutes.

JSON data is parsed in a similar manner to XML. However, less work is involved in terms of detecting structures and inferring datatypes as JSON provides array, object string and number data types. These provide an easier mechanism for determining node classifications. Figure 4.9 shows the constructed StarGraph as consisting of three measures: the distance between two junctions, the current travel time, and

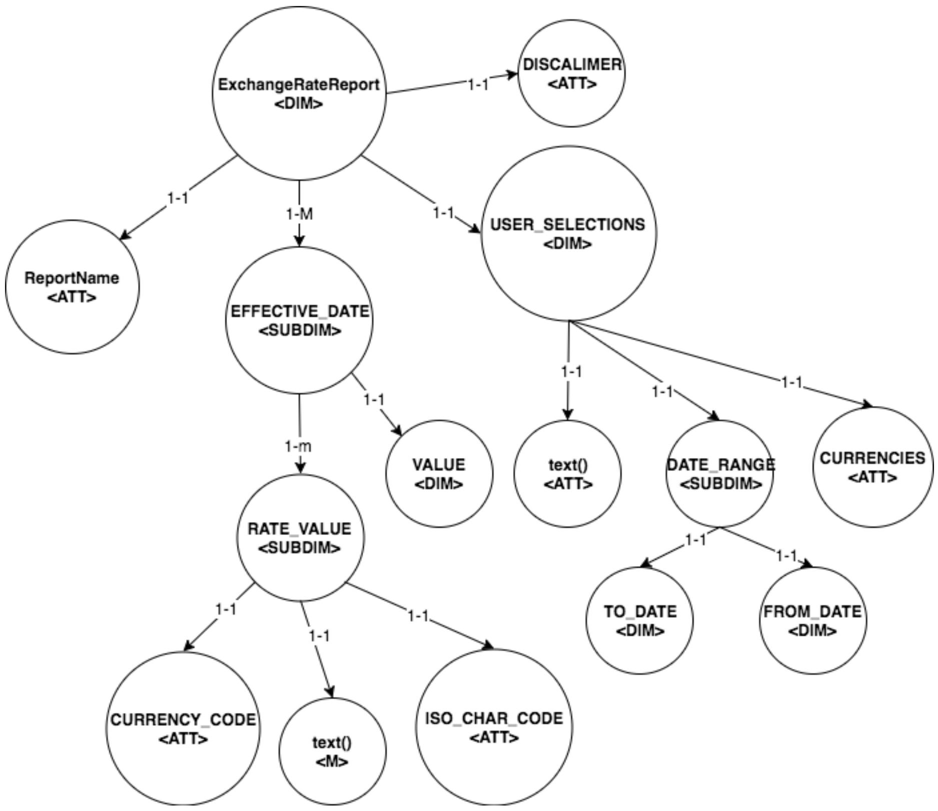


Figure 4.8: StarGraph created from the `imf` dataset

the minimum travel time. A sample of the target schema and mappings produced by the `tii` StarGraph can be seen in Example 4.3.9 and Example 4.3.10 with a full set being provided in Appendix C.

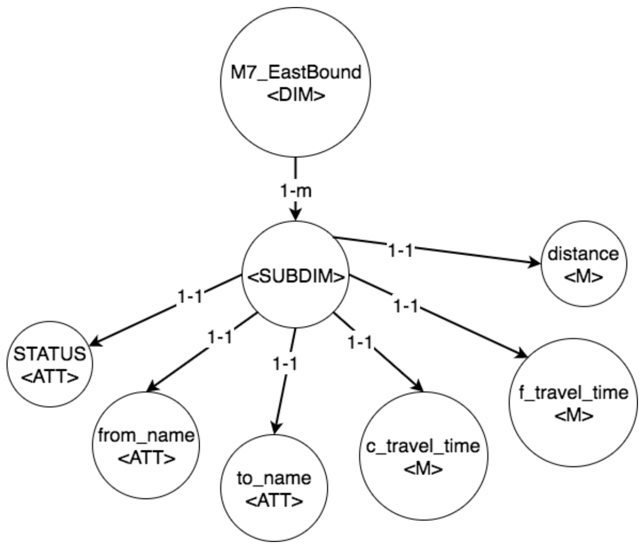


Figure 4.9: StarGraph for `tii` dataset

Example 4.3.9. Target Schema produced by the `tii` StarGraph.

```
{
    "name" : "tii",
    "type" : "JSON",
     source:"dataproxy.mtcc.ie/..",
    dimensions : [
    {
        "name" : "M&_EastBound",
        "type" : "Object",
        "src" : "M7_EastBound",
        "attributes":[],
        "subdimensions":[
            {
                name:"",
                "type":"Object",
                "src":"M7_EastBound.data[*]",
                "attributes":[{
                    "name":"from_name",
                    "type":"String",
                    "src":"M7_EastBound.data[*].from_name"
                },...]
            }
        ]...,
    },..
    ],
    measures : [
    {
        "name" : "c_travel_time",
        "type" : "MEASURE",
        "src" : "*.data[*].current_travel_time"
    },..
    ],
    facts:[{
        dimensions:["M7_EastBound",....],
        measures:["current_travel_time", "distance",...]
    }]
}
```

Example 4.3.10. Mappings produced by the `tii` StarGraph.

("M7_EastBound.data[*].from_name")→ M7_EastBound["from_name"]

("M7_EastBound.data[*].to_name")→ M7_EastBound["to_name"]

("M7_EastBound.data[*].c_travel_time")→ c_travel_time

4.3.3 CSV Data Streams

CSV data streams are the easiest to transform into an ATS. However, determining data types requires examining every value for a column. As such, the process for determining datatypes is $\mathcal{O}(nm)$ where n is the number of rows and m the number of columns. Additionally, relationships in CSV files are assumed to be `ONE_TO_ONE`. This

is due to the complexity involved in determining relationships which is $\mathcal{O}(\binom{n}{2}m)$. CSV datasets also generate simple StarGraphs, with each column occupying a node in the StarGraph. These columns can then either become dimensions or measures. This is due to the fact that complex dimensions cannot be obtained from CSV files, as they are largely dependent on the structure of the raw data. Case Study 2 utilises many CSV data sources and they produce similar StarGraphs to `b_1` from Case Study 1. This is because like these HTML data sources, the CSV sources contain only two columns, a date and a measure. More complex CSV sources such as `usda` produce StarGraphs with a large degree of dimensions and measures.
A sample of this StarGraph can be seen in Figure 4.10 while the target schema and mappings produced can be seen in Example 4.3.11 and Example 4.3.12 with a complete version being provided in Appendix D.

Example 4.3.11. Target Schema produced from the `usda` dataset.

```
{
    "name" : "usda",
    "type" : "CSV",
    "source" :"http://quickstats.nass.usda.gov/api/..",
    dimensions : [
    {
        "name" : "source_desc",
        "type" : "STRING",
        "attributes":[],
        "src" : "[0]"
    },..
    ],
    measures : [
    {
        "name" : "Value",
        "type" : "MEASURE",
        "src" : "[37]"
    },..
    ],
    facts:[{
        dimensions:["source_desc",....],
        measures:["Value",....]
    }]
}
```

Example 4.3.12. Mappings produced from the `usda` dataset.

("[0]")→ "source_desc"

...

("[37]")→ "Value"

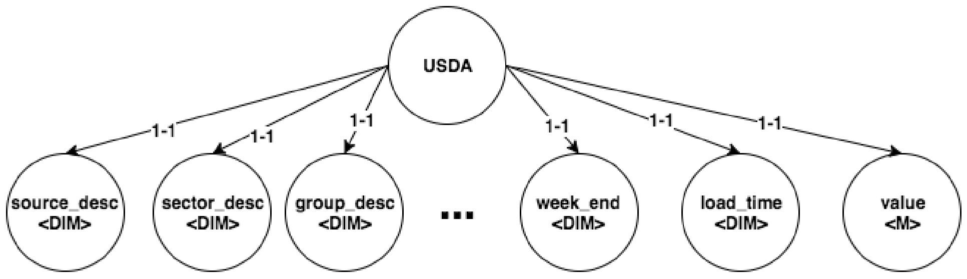


Figure 4.10: Sample StarGraph created from `usda` dataset.

However, all CSV data sources, produce StarGraphs similar to the source data. This is because unlike JSON and XML, the data is tabular in nature, and unlike HTML, does not have dimension hierarchies or complex structures.

4.4 StarGraph Transformation: An Interim Case Study

At this point, the reader may have a fundamental question to ask: is there an overriding assumption that all sources are multi-dimensional? Where is the value in our research if there exists no multidimensional data in the sources that make up our *multidimensional* data marts? Our assumption is that analytical data (data used in reports, tables, analyses) is by its nature, multi-dimensional in structure. To demonstrate that our hypothesis is sufficiently on track to continue with our stated goals, we were provided with the entire set of agri sources (120 in total) used in a research project by one of our collaborators. The goal of this analysis is a mid-term evaluation of our research to ensure that the StarGraph concept works, and can be used in later stages to integrate sources and form usable and potentially complex data marts.

As part of this evaluation, we examined 120 unseen Agri-data sources and as expected, not all of these sources could form part of a StarGraph while others produced StarGraphs of varying quality. For all 120 data sources, we provided classifications regarding the suitability of the resulting StarGraph with the results presented in Table 4.1. The classifications for sources are as follows:

- **Full**. This classification was reserved for sources which produced full StarGraphs complete with dimensions and measures.

Table 4.1: Results of Analysing 120 Agri Data Sources

Classification	Facts	Dims	Holistic	Non-Spurious	Count	Sources(%)
Full	✓	✓	✓	✓	41	34%
Partial	✓	✓	✓	✗	29	24%
Descriptive	✗	✓	✓	✗	14	12%
Missing	✗	✓	✗	✓	2	1%
Unusable	✗	✗	✗	✗	34	29%

- **Partial**. This classification was used for StarGraphs which were missing attributes and metadata but still provided a usable data mart.
- **Descriptive**. This classification was used for StarGraphs which did not produce any measures, but instead contained rich dimensional hierarchies which could prove useful when integrating StarGraphs.
- **Missing**. This classification was used for StarGraphs which did not contain enough information in order to be usable. Dimensions may be present for sources found within this classification, however they are not of sufficient depth to be usable.
- **Unusable**. This classification was reserved for sources which failed to produce a StarGraph.

We can see from Table 4.1 that 70% (84) of the unseen data sources produced usable StarGraphs and of this 84, 41 produced Full StarGraphs. The columns `Facts`, `Dims`, `Holistic` and `Non-Spurious` refer to the presence (or absence) of desired attributes within each classification. `Facts` indicates that one or more facts were identified. `Dims` indicates one or more dimensions were found. `Holistic` means no attributes were discarded during construction. `Non-Spurious` indicates whether or not the StarGraph was 'usable'. Finally, `Count` details the number of sources which fell under a classification.

Unusable sources included non supported formats (e.g. PDFs, JPEG etc...), HTML files which contained malformed markup. In these instances the misuse or absence of recommended tags halted program execution. Finally, some sources which failed to integrate were small data cubes represented as pivot tables in Excel. The two sources classified as `Missing` found only a date dimension and nothing else. In the absence of any other attributes the dimension is of little use. No facts were found for the `Descriptive` data sources, however rich dimensional hierarchies were discovered which could prove useful when integrated with another StarGraph. Finally, `Partial` data sources produced unusable StarGraphs. A common example was the misuse of $\langle table \rangle$ tags within HTML sources, using these elements to present data and determine page layout.

One may ask through our analysis did a standard of data sources emerge which could be used to facilitate top down integration [31]. Our metamodel (Sect. 3.3.3) emerged from our search for commonality across the full set of agri sources. In essences, this provided us with a form of standard for multi-dimensional representation of agri sources.

4.5 Summary

This chapter presented a system and methodology which can be used to construct and populate a data mart from a single data source. The process uses an annotated graph as a common data model, with a series of node classifications to identify the facts and dimensions located within a single data source. These measures and dimensions are subsequently used to construct a series of facts which will constitute a data mart. The mappings constructed by the process are subsequently used to populate the data mart. The overall methodology provides a means of automatically capturing an individual data stream and populating a data mart without the need for user intervention. However, most analyses will require data from multiple sources and at this point, this approach can only construct a data mart (as a StarGraph) from a single data source. To facilitate more powerful analyses, it is necessary to integrate data sources. This provides the focus for the next chapter.

Chapter 5

StarGraph Integration

While the previous chapter presented a system which provided a means of constructing a data mart from a singe data source, this chapter provides a series of extensions to this system which allow a user to construct a data mart from multiple data streams.

In the previous chapter we presented a detailed description of our method for constructing a data mart from a single data source. However, data marts generally comprise multiple heterogeneous data sources, requiring a significant investment in data integration. There are multiple problems associated with automatic and semi-automatic data integration. Specifically, the fact that these approaches fail to capture and understand the abstractions known to the ETL designer or domain expert. In this chapter we present a multi-source representation of the StarGraph which we call a *ConstellationGraph* or *Constellation* for short. In section 5.1, we provide a discussion on the functions necessary to create the Constellation. This is similar to the requirements section in chapter 4 but here, we focus on the integration process and the specific issues causes by the semi-structured nature of the data. In section 5.2, we present the functional aspects to the StarGraph model that manage integration; and finally in section 5.3, we provide a detailed description of data mart construction and the mappings generated by the transformation process.

5.1 Integrating Semistructured Data

In chapter 4, it was necessary to highlight the issues to be addressed when transforming data from one model representation to another. The process of integrating two (or more) data sources requires a similar process. In this section, we ensure that these issues are highlighted and understood, before specifying the integration properties of our model.

5.1.1 Resolving Mapping Difficulties

When designing an ETL system, the designer draws upon domain and general knowledge to provide semantic integration. As we provide a more automated integration, the reliance on the data engineer must be minimal. Therefore, we must provide a method for resolving structural and semantic issues that arise during the integration process.

Term Mapping.

There is generally a requirement to resolve different terminologies used in separate data sources. These differences may manifest in various ways, most obvious is the fact that different languages use different strings to represent the same concept. Another reason for differences is using different specific terminologies to a domain or having multiple strings which may represent the same concept (for example abbreviations such as the strings 'EU' and 'European Union'). However, all of these problems can be overcome by providing canonical terms to each item found amongst all data sources. By replacing individual strings with canonical terms representing the concepts that the string represents, the system is able to identify common features across datasets. This process requires the existence of some form of ontology which captures these terms and serves as a lookup table (called a term-map).

Type Mapping.

Unfortunately term mapping does not resolve all differences between data sources. While term mapping seeks to resolve entities of the same concept, there still exists the need to determine concepts of the same type. Some types may be automatically determined, for example `date` types and `geo-location` data. However other types are not present in an automatic system which a designer would have knowledge of. A common example of types lies within a *Geo* dimension. For example, the strings ‘Ireland’ and ‘France’ are not similar. However, an application designer would know that these are both countries and as such would both be values within a country dimension. This process should examine the canonical terms assigned during the previous step, and annotate these with the abstract types that the concept represents. This requires an extension to the term-map ontology which provides the addition of type information.

5.1.2 Identifying Integration Attributes

One of the tasks for integration is the identification of the attributes which are common to both sources and these attributes are subsequently used to integrate the data sources. Considerable research into warehouse building [42] has show the same subset of dimensions used in almost all application areas, if we consider them in their most abstract form. Using this research, we propose that an ontology or metamodel should capture and understand the following properties:

- Date. These must be a value for a data dimension
- Geo. This type is of the Geo dimension.
- Item. This is a string denoting what item is being measured. (e.g. product)
- Metric. This is a string what method is being used to provide the measure
- Units. This is a string denoting the units of measurement for the measure. These can be standard units of measurement (e.g. kg, lb) or aggregate functions (SUM/TOTAL, AVERAGE, COUNT).
- Value. This type is the value of the measure itself.

However, all of these values may not be present within a data source, so it should possible for a user to provide these values in the event that they are not found. It is also important to note that these attributes are required `per-measure`. That is for each measure (Value) found within a data source.

5.1.3 Selecting an Integration Strategy

A process is required to examine the two data streams for integration in order to determine the possible integration attributes. The process should combine dimensions which are of the same value, and determine the degree of integration for the two data sources. Some data sources may be directly integrated, producing a data mart which closely matches the system metamodel while others may produce marts of varying complexity, depending on the degree of integration. These two possibilities are encapsulated within the two integration strategies `row-append` and `column-append`.

`row-append` is reserved for data sources which share the exact metamodel. This is reserved for data which may be fully integrated while `column-append` encapsulates all other possibilities deriving from the possible combinations of metamodel values present within both data sources.

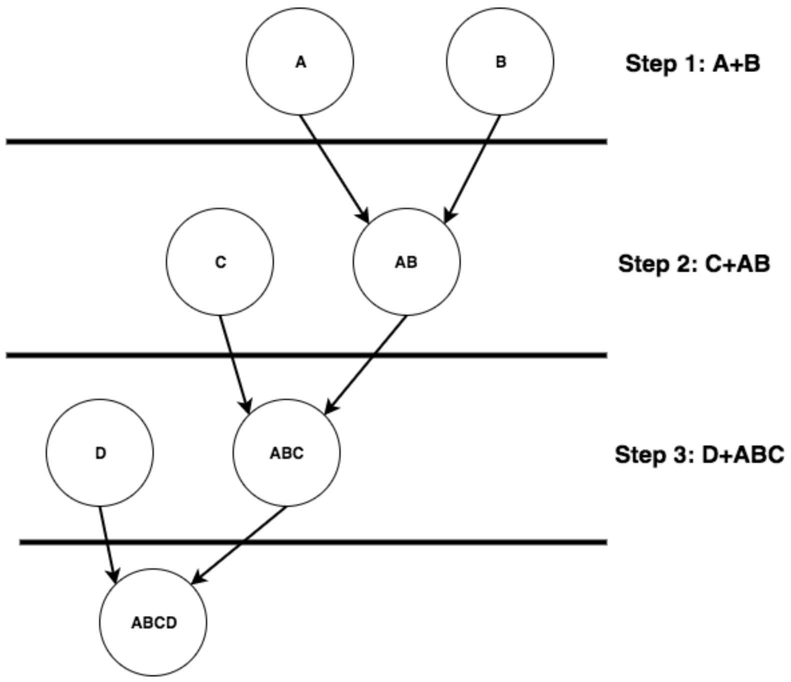


Figure 5.1: Example of the ladder integration strategy for sources A,B,C and D

5.2 Integration Functions

Data sources required for integration are selected by a user. These sources are then passed as a list to the system and integrated using the ladder strategy [8]. That is, the first two sources in the list are integrated, then the next source is selected and integrated into the previously integrated sources. This process continues until there are no more sources to integrate. This results in a process involving $n-1$ integration steps where n is the number of sources to integrate. A conceptual diagram of the ladder strategy for data sources named `A`, `B`, `C` and `D` can be seen in Figure 5.1.
It is important to note that the order in which the sources are selected does not influence the integration process. That is, for all possible permutations of that set, there exists only one possible StarGraph configuration. This is due to the presence of the ontology which influences the integration approach. An empirical analysis and discussion of this property is presented in section 5.3.1.
Our assumption is that the user selects a set of data sources which they would like to merge into a single data mart. The StarGraph behaviour (functionality) described in chapter 4 has an extended behaviour to manage integration. There are seven main functions which comprise the integration process. These functions detail all steps involved in integrating two web datasets, from the initial selection of the sources to the construction and materialisation of the multi-source mart. These functions are:

- `TermMap`: annotate data sources with canonical terms.
- `TypeMap`: annotate data source with abstract types.
- `MetaModelCheck`: ensure a data source is compliant with the metamodel.
- `GranularityCheck`: ensure the data sources set to be integrated are of the same grain.
- `DetermineStrategy`: determine an integration strategy.
- `GenerateTargetSchema` and `GenerateMappings`: Construct the target schema, and corresponding mappings to extract and integrate data.
- `Materialise`: populate or update the constellation.

5.2.1 The `TermMap` function

The `TermMap` function annotates the graph with canonical terms for attributes found within the data source and is necessary in order to resolve heterogeneities in data sources. A common use for `TermMap` resides in the *Geo* dimension with the numerous ways different country names can be represented. For example, the terms '`IRL`' and '`Éire`' are not related using any standard string matching algorithm. However, both terms refer to the country Ireland.

Algorithm 5.1 takes two parameters: the name of the required node, and the context of the graph. The context of the graph is the source (unique-identifier) of the data source from which the StarGraph was created. This is required as different data sources may use similar terms within different contexts. For example, one source may use the term 'Ireland' to refer to the *sale* of pigs in Ireland, while another may use the same term to refer to the *production* of pigs in Ireland.

Algorithm 5.1 The TermMap function

```
1: function termMap(graph)
2:     for node in graph.nodes do
3:         term = getTermForNode(n.name,graph.context)
4:         node.addTerm(term)
5:     end for
6:     return
7: end function
```

Using the Bord Bia example, the node with the name '`time`' is renamed to '`week`' as in this context, the data values are dates representing weeks and do not contain time values. A diagram of this graph after the `TermMap` process can be seen in Figure. 5.2 with the annotated terms between '(' and ')'.

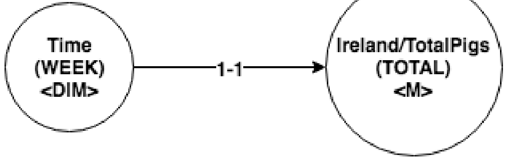


Figure 5.2: `bb_1` StarGraph after application of the `TermMap` function.

For the `aim_1` dataset, the application of the `TermMap` function removed some ambiguities, such as the lack of a name for the measures, and provided more context in relation to terms such as‘East’ and ‘West’ referring specifically to ‘East-Canada’ and ‘West-Canada’ respectively. The `aim_1` StarGraph returned from the `TermMap` function can be seen in Figure 5.3.

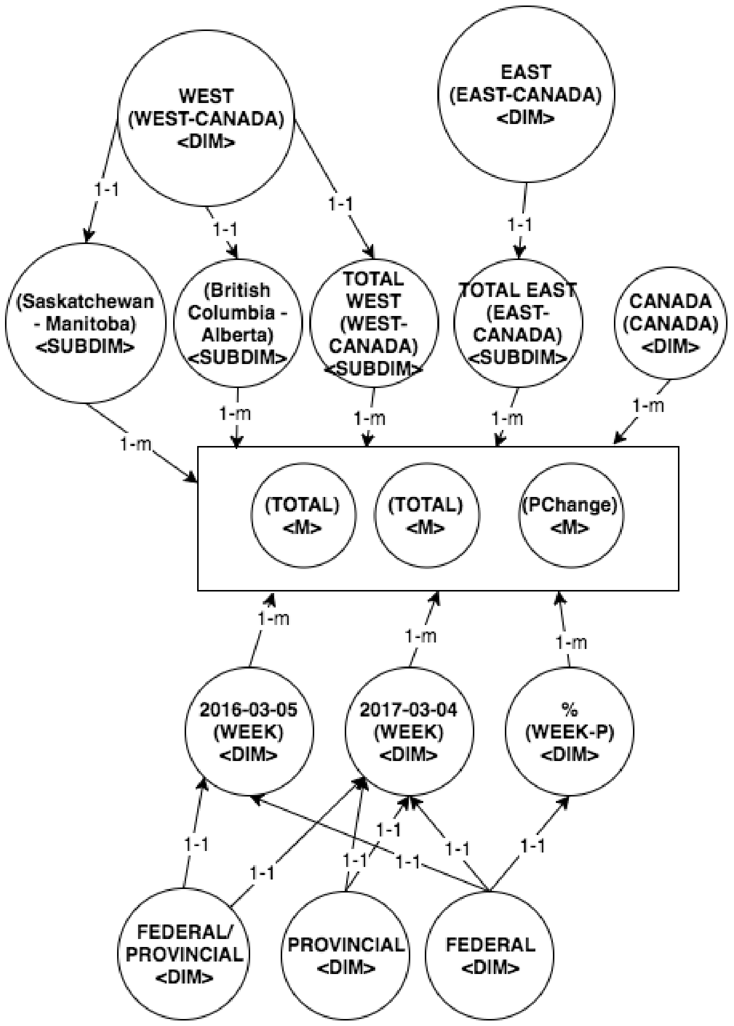


Figure 5.3: `aim_1` StarGraph after application of the `TermMap` function.

This process extends the mappings for a StarGraph by adding a property `term` to each node denoting the canonical term obtained from the ontology. An example of this for the `aim_1` data source can be seen in Example 5.2.1. In this example, we can see the addition of a `term` property for each dimension, attribute, subdimension and measure, denoting the canonical term obtained from the ontology.

Example 5.2.1. Mappings after `TermMap` for `aim_1`.

```
{
    name:"aim_1",
    type:"HTML",
    source:"http://aimis-simia.agr.gc.ca/..."
    dimensions:[
        {
          name:"West",
          type:"object",
          term:"WEST-CANADA"
         src:"//table[1]/tbody/tr[3]/th[1]",
          attributes:[],
          subdimensions:[...]
        },
        {
            name:"Federal/Provincial",
            type:"object",
            term:"FEDERAL-PROVINCIAL",
            src:"//table[1]/tbody/tr[2]/th[2]",
            attributes:[],
            subdimensions:[...]
        },
    ... ...],
```

5.2.2 The `TypeMap` function

There still exists the issue of resolving concepts which should be treated as the same type, even though they appear to different (e.g. 'Ireland' , 'France' are two values of the same *Country* type). Type mapping assigns canonical *types* to each node by capturing the abstract concepts which link these nodes in order to further facilitate semantic integration. These types are obtained from the ontology. There are five types specified as being the minimum requirement for an automated integration as defined by the metamodel (section 3.3.3). However, the user may extend the metamodel by defining their own types within the ontology in addition to base types.

Algorithm 5.2 The TypeMap function

```
1: function typeMap(graph)
2:     for node in graph.nodes do
3:         type = getTypeForNode(n.term)
4:         node.addType(type)
5:     end for
6:     return
7: end function
```

The `TypeMap` function can be seen in Algorithm 5.2. This function call `getTypeForNode` on line 3 searches the ontology for the type of the node. The function takes the term previously applied to a node and performs a lookup operation in the ontology to return the canonical type associated with that term.

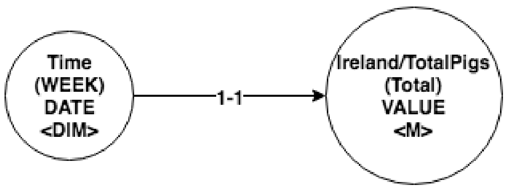
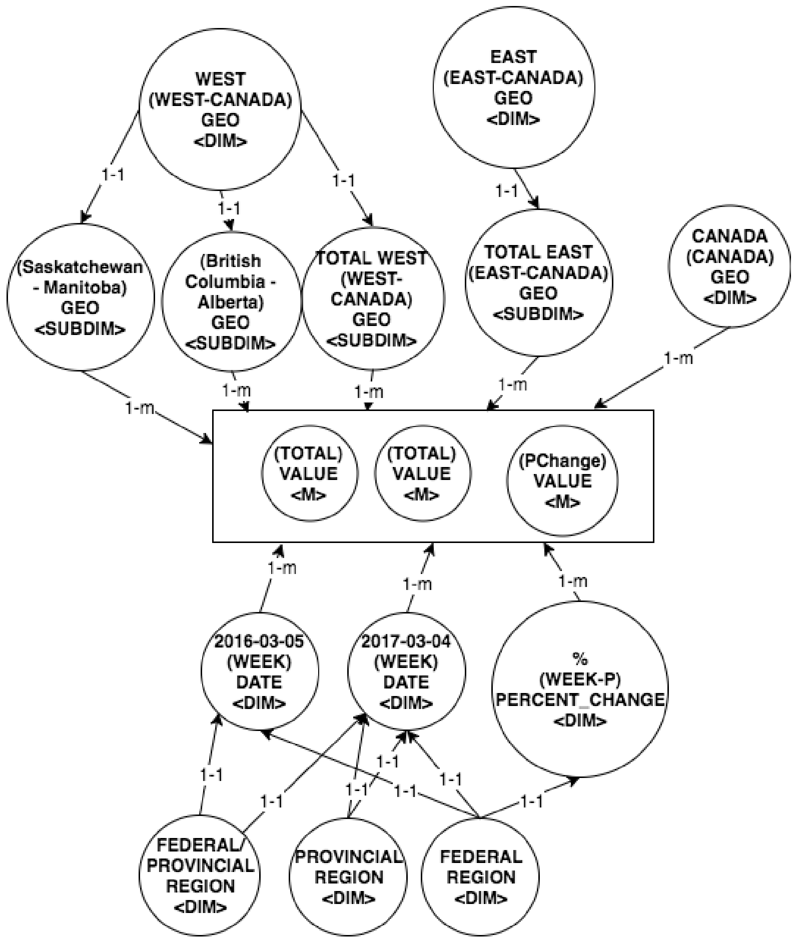


Figure 5.4: `bb_1` StarGraph after application of the `TypeMap` function

For the Bord Bia dataset, the applied types were both metamodel types: `Date` and `Value`. The `bb_1` StarGraph produced by the `TypeMap` function can be seen in Figure. 5.4. Similarly for the `aim_1` StarGraph, there were three Metamodel attributes: `Date`, `Geo` and `Metric`. In addition, one type which was not part of the metamodel, but a part of the ontology, `Region`, was annotated to the nodes `Federal`, `Provincial` and `Federal/Provincial`. The `aim_1` StarGraph returned by the `TypeMap` function can be seen in Figure. 5.5.
Similar to the `TermMap` function, this function extends the mappings by adding a property `onttype` (ontlogy type) denoting the type of the node as obtained from the ontology. An example of these mappings for the `aim_1` dataset can be seen in Example 5.2.2.



WEST (WEST-CANADA) GEO <DIM>
EAST (EAST-CANADA) GEO <DIM>
1-1
(Saskatchewan - Manitoba) GEO <SUBDIM>
(British Columbia - Alberta) GEO <SUBDIM>
TOTAL WEST (WEST-CANADA) GEO <SUBDIM>
TOTAL EAST (EAST-CANADA) GEO <SUBDIM>
CANADA (CANADA) GEO <DIM>
1-m
(TOTAL) VALUE <M>
(TOTAL) VALUE <M>
(PChange) VALUE <M>
2016-03-05 (WEEK) DATE <DIM>
2017-03-04 (WEEK) DATE <DIM>
% (WEEK-P) PERCENT_CHANGE <DIM>
FEDERAL/ PROVINCIAL REGION <DIM>
PROVINCIAL REGION <DIM>
FEDERAL REGION <DIM>

Figure 5.5: aim_1 StarGraph after application of the TypeMap function

Example 5.2.2. Mappings after TypeMap for aim_1.

```
{
    name:"aim_1",
    type:"HTML",
    source:"http://aimis-simia.agr.gc.ca/..."
    dimensions:[
        {
          name:"West",
          type:"object",
          term:"WEST-CANADA",
          onttype:"GEO",
         src:"//table[1]/tbody/tr[3]/th[1]",
          attributes:[],
          subdimensions:[...]
        },
        {
            name:"Federal/Provincial",
            type:"object",
            term:"FEDERAL-PROVINCIAL",
            onttype:"REGION",
            src:"//table[1]/tbody/tr[2]/th[2]",
            attributes:[],
            subdimensions:[...]},...],
```

5.2.3 The `MetamodelCheck` function

The `MetamodelCheck` function ensures that a measure has sufficient information to complete the integration process. For every measure, the function confirms that the measure has connections to nodes with the types `Date`, `Geo`, `Item`, `Metric` and `Unit`. These types are required for each measure as multiple measures within a data source may represent different metrics. For example, one measure may provide a total count for a given date, while another may provide cumulative totals. However, without the necessary information `Item`, `Metric` and `Unit` for the measure, it is impossible to infer what the measure represents. The system examines each measure to determine if these attributes are present and, if not, prompts the user to supply the missing value.

For example, the `aim_1` dataset contains two measures: the total number of pigs slaughtered in a given week, and the percentage change of this number compared to the same week of the previous year. For this data source, the type `Item` refers to the high level concept being analysed, in this instance, the value for `Item` is `Pigs`. `Metric` refers to the metric used for the measure in question, in this instance, the value of `Metric` is `Slaughter`. However, the type `Unit` is different for each measure, with the number of pigs slaughtered having a `Unit` value of `Total` and the percentage change measure having the `Unit` value `PCHANGE_WEEK_YEAR` referring to the percentage change of the value for the same week in the previous year.

Conversely, the `b_1` dataset contains a single measure, the total number of pigs slaughtered per week and as such, has an `Item` value of `Pigs`, a `Metric` value of `Slaughter` and a `Unit` value of `Total`. By comparing both sets of metamodel attributes, it can be determined that the sole measure presented in `b_1` is semantically identical to the `Total` measure in `aim_1` and thus, the system can integrate both into the same fact.

However, metadata such as `Date`, `Geo`, `Item`, `Metric` and `Unit` may not be present within the data source and this final process requires user assistance to complete. The benefit of the `MetamodelCheck` function is that it highlights the precise location where these user-assisted inputs are required.

The function examines all nodes which *connect* to a measure (either direct relationships or through transitive closure). The types of these nodes returned by the `TermMap` function are then analysed and placed within a set.

Once the set has been constructed, it is examined in order to determine if *all* required types are present in the set. If there are missing attributes, a `MISSING_ATTR` event is triggered. This event prompts a user to provide values for the missing metamodel attributes.

Once these values have been provided by a user, they are stored in the metabase and will be used by any future integrations involving this data source.

Algorithm 5.3 outlines the process for the metamodel check function. The function examines the measure found within a StarGraph. The function call on line 4 `getRelatedNodes` is shorthand for loading all nodes which share an edge with a measure, either directly or through transitive closure. Once a list of related nodes are loaded, this list is then examined to see if there exists a node in this set for each metamodel attribute using the `contains(⟨type⟩)` function.

If no such type exists, the `missing_attr` event is triggered which prompts a user for a value.

It is important to note, even though `Value` is a metamodel attribute, this function does not check for the existence of these types. This is because each measure is assigned the type `Value` as they contain the value of the measure of interest.

Algorithm 5.3 The MetamodelCheck function

```
1: function METAMODELCHECK(graph)
2:     measures = graph.measures
3:     for m in measures do
4:         relNodes = m.getRelatedNodes
5:         if !relNodes.contains(Date) then
6:             missing_attr(Date)
7:         end if
8:         if !relNodes.contains(Geo) then
9:             missing_attr(Geo)
10:        end if
11:        if !relNodes.contains(Item) then
12:            missing_attr(Item)
13:        end if
14:        if !relNodes.contains(Metric) then
15:            missing_attr(Metric)
16:        end if
17:        if !relNodes.contains(Units) then
18:            missing_attr(Units)
19:        end if
20:    end for
21:    return
22: end function
```

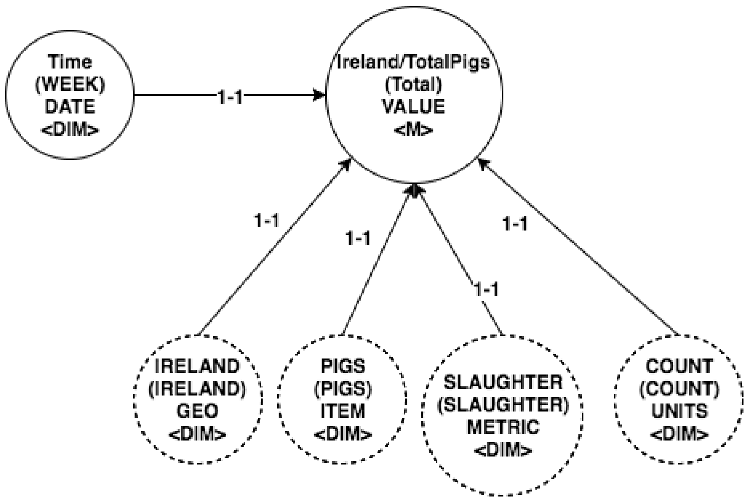
Once all metamodel attributes for all measures between both data sources are found, the next stage is to examine these metamodel attributes to determine if they are semantically compatible.

The `b_1` dataset, consisting of only two values required multiple metamodel attributes to be entered by a user. These attributes were values for the `Item`, `Metric`, `Units` and `Geo` attributes of the metamodel.

Figure. 5.6 shows the `b_1` StarGraph with the addition of these attributes. They are represented by graph nodes with dotted lines.

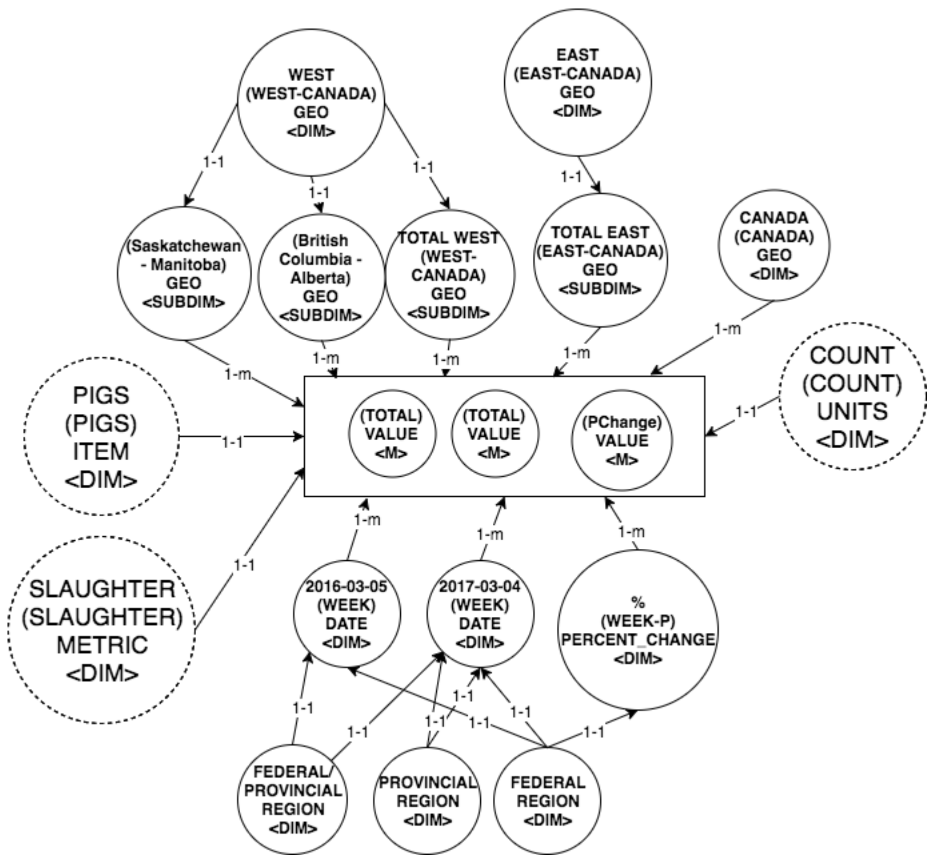
The `aim_1` dataset also required a user to enter metamodel attributes, these were attributes for the `Item`, `Metric` and `Units` attributes.

As this function acts as a check to see if desirable properties are present, and if not, add them. If all desirable properties are present there, no change is made to the underlying mappings. However, if a user supplies missing metamodel values, these are incorporated into the mappings.



Time (WEEK) DATE <DIM>
Ireland/TotalPigs (Total) VALUE <M>
1-1
IRELAND (IRELAND) GEO <DIM>
PIGS (PIGS) ITEM <DIM>
SLAUGHTER (SLAUGHTER) METRIC <DIM>
COUNT (COUNT) UNITS <DIM>

Figure 5.6: b_1 StarGraph after metamodel check



WEST (WEST-CANADA) GEO <DIM>
EAST (EAST-CANADA) GEO <DIM>
(Saskatchewan - Manitoba) GEO <SUBDIM>
(British Columbia - Alberta) GEO <SUBDIM>
TOTAL WEST (WEST-CANADA) GEO <SUBDIM>
TOTAL EAST (EAST-CANADA) GEO <SUBDIM>
CANADA (CANADA) GEO <DIM>
PIGS (PIGS) ITEM <DIM>
COUNT (COUNT) UNITS <DIM>
SLAUGHTER (SLAUGHTER) METRIC <DIM>
(TOTAL) VALUE <M>
(TOTAL) VALUE <M>
(PChange) VALUE <M>
2016-03-05 (WEEK) DATE <DIM>
2017-03-04 (WEEK) DATE <DIM>
% (WEEK-P) PERCENT_CHANGE <DIM>
FEDERAL/ PROVINCIAL REGION <DIM>
PROVINCIAL REGION <DIM>
FEDERAL REGION <DIM>
1-1
1-m

Figure 5.7: aim_1 StarGraph after metamodel check

A sample of the new mappings produced by the aim_1 StarGraph after the MetamodelCheck function can be seen in Example. 5.2.3.

The only difference between these and previous mappings are the presence of these new metamodel values. In addition, the `src` properties for these values is the static value supplied by the user.

Example 5.2.3. Mappings after `MetamodelCheck` for `aim_1`.

```
{
     name:"aim_1",
     type:"HTML",
     source:"http://aimis-simia.agr.gc.ca/..."
     dimensions:[
         {
           name:"West",
           type:"object",
           term:"WEST-CANADA",
           onttype:"GEO",
          src:"//table[1]/tbody/tr[3]/th[1]",
           attributes:[],
           subdimensions:[...]
         },
         {
            name:"SLAUGHTER",
            type:"String",
            term:"SLAUGTER",
            onttype:"METRIC",
            src:"SLAUGHTER"
         },
         {
             name:"Federal/Provincial",
             type:"object",
             term:"FEDERAL-PROVINCIAL",
             onttype:"REGION",
             src:"//table[1]/tbody/tr[2]/th[2]",
             attributes:[],
             subdimensions:[...]},...],
```

5.2.4 The `GranularityCheck` function

The role of the `GranularityCheck` function is to examine the two StarGraphs for integration to determine if their dimensions have the same granularity. This ensures that any integration is semantically correct. In the event that both graphs contain the same hierarchical dimension, they are checked to see if they both reside on the same level in the hierarchy.

This function resolves conflicts which can arise when integrating data from, for example, the date dimension which is hierarchical in nature. For example, one data source may be monthly while the other is daily. In this event, data cannot be directly compared. These events trigger a `GRAIN_MISMATCH` event. The system cannot directly integrate the two sources on this level as it requires the finer grained source to undergo a `rollup` operation so that both StarGraphs occupy the same level in the appropriate dimension hierarchy.

5.2.5 The `DetermineStrategy` function

The role of the `DetermineStrategy` function is to examine the metamodel attributes of data sources and determine how they should be integrated using four steps.

Step 1: Construct Dependency Sets. The first step constructs dependency sets for each measure in both data sources. Each set contains a measure and all metamodel attributes found for the measure. For the source `b_1`, there is only one dependency set shown in Example 5.2.4.

Example 5.2.4. Dependency set for `b_1`.
⟨*WEEK, GEO, PIGS, SLAUGHTER, COUNT, VALUE* ⟩

As there are two measures in `aim_1`, there are two dependency sets, shown in Example 5.2.5. In the remainder of this discussion, we will refer to the set in `b_1` as `b_1_a` and the sets in `aim_1` as `aim_1_a` and `aim_1_b`.

Example 5.2.5. Dependency sets for `aim_1`.
⟨*WEEK, GEO, PIGS, SLAUGHTER, COUNT, VALUE* ⟩,
⟨*WEEK, GEO, PIGS,SLAUGHTER ,PERCENT_CHANGE, VALUE* ⟩

Step 2: Create Combination Table. The next step is to examine dependency sets to determine how they may be integrated. The process examines each combination of dependency sets, at each stage calling the `combine` function (Algorithm 5.4). There are three possible outcomes to this function: `row-append`, `column-append` and `no-integration`, which represent the integration strategies. The output is a table detailing how each combination of dependency sets may be integrated.
The `row-append` method is used for sets which are identical, meaning that a direct integration can take place. The `column-append` method is used for sets which only differ based on their `Metric` and `Unit` values. In these instances, the facts are based around the same concept, but differ in the metrics and units used. The column-append strategy integrates data where two sources contain the same values for the metamodel values `Date`, `Geo` and `Item`. This strategy overcomes differences between

Table 5.1: Combine function results: aim_1 and b_1

Set_1	Set_1	Method
b_1_a	aim_1_a	row-append
b_1_a	aim_1_b	column-append
aim_1_a	aim_1_b	column-append

units of measurement, and combines multiple measures into a single fact based on their similar metamodel attributes. Finally the no-integration method is reserved for sources for which comparisons cannot be determined. Specifically, if they contain different Date, Geo or Item values, then integration cannot take place.

Algorithm 5.4 The Combine function

```
1: function COMBINE(set_a, set_b)
2:     if IDENTICAL(set_a,set_b) then
3:         return row-append
4:     else if DIFFERENCE(set_a,set_b) = ⟨METRIC, UNITS⟩ or ⟨METRIC⟩ or
   ⟨UNITS⟩ then
5:         return column-append
6:     else
7:         return no-integration
8:     end if
9: end function
```

Algorithm 5.4 shows the combine function. The call identical on line 2 determines if both sets are completely identical $((set\_a \cup set\_b) \setminus (set\_a \cap set\_b) = \emptyset)$. The call difference on line 4 calculates the difference $((set\_a \cup set\_b) \setminus (set\_a \cap set\_b))$ between two sets. Once all combinations have been calculated, a table of all combinations and their respective methods are returned. Table 5.1 details the results of the combine function for the data sources aim_1 and b_1. The sets b_1_a and aim_1_a were deemed identical and as such the row-append strategy is used. For the remaining combinations, the column-append strategy is used as the sets only differ based on their Metric and Unit values.

Step 3: Create Combination Graph. While the results of the previous step show how each pair of dependency sets may be integrated, there remains the issue of integrations which may occur through transitive closure. In order to overcome this, a combination graph is constructed from the table. A combination graph is an undirected graph, where each node in the graph represents a dependency set, and the edges between nodes determine how they should be combined. As such we shall call this graph a `combination graph`.

Figure 5.8 represents the combination graph for the sources `aim_1` and `b_1`. From the graph we can see the nodes `aim_1_a` and `b_1_a` are linked through the `row-append` method and `aim_1_b` is linked to both using the `column-append` method.

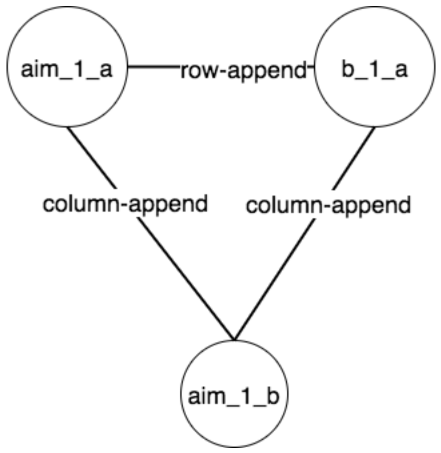


Figure 5.8: Combination graph for `aim_1` and `b_1`

Step 4: Collapse Combination Graph. The next step is to collapse the graph. At the end of this process, the number of nodes in the graph relate to the number of facts which shall be constructed in the Constellation. The graph is collapsed by merging nodes in the graph. The function first examines all nodes which share a `row-append` edge and combines them into a single node. This new node inherits all the edges obtained from the previous nodes. This results in a graph similar to the one shown in Figure 5.9. In this figure, we can see that the nodes `aim_1_a` and `b_1_a` have been merged to produce a single node. This node holds two `column-append` edges to the node `aim_1_b`.

At this stage, it is possible to have a node which holds two edges to the collapsed node. The next step of this process is to resolve these edges. It achieves this by selecting the highest possible edge from a hierarchy. Table 5.2 outlines the possible combinations of edges, and the resulting edge which is chosen. From this table we can see that the `column-append` edge is selected. This results in the graph shown in Figure 5.10.

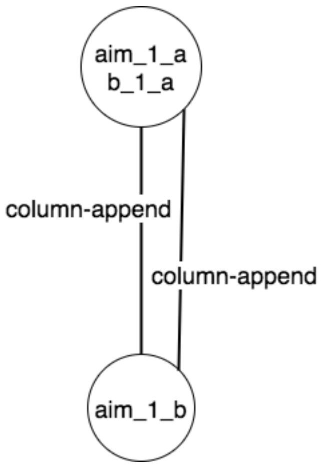


Figure 5.9: Combination graph after merging row-append edges

Table 5.2: Integration Strategy Based on Edges

Edge 1	**Edge 2**	**Final Strategy**
row-append	row-append	row-append
row-append	column-append	row-append
row-append	no-integration	row-append
column-append	column-append	column-append
column-append	no-integration	column-append
no-integration	no-integration	no-integration

This process repeats for every set of `row-append` edges until there are none present in the graph. Once there are no `row-append` edges, the process or merging nodes continues for all `column-append` edges. At the end of this process, there are two possibilities. There is a single unified node, which means that all sources can be combined into a single unified fact, or there are a series of merged nodes, which all hold `no-integration` links to each other. As shown by the single node in Figure 5.11.

The steps taken to collapse the graph are captured and constitute the integration strategy, producing a set of integration steps shown in Table 5.3. Figure 5.12 outlines, on a high-level, the Constellation created from the StarGraphs `aim_1` and `b_1`. This highlights metamodel values (i.e. the values for `Date`, `Geo`,`Item`,`Metric` and `Unit`) found within each data source, and their interactions which constitute a *fact*. The blank dotted nodes indicate different measures, and the dotted node `Region` is an optional dimension obtained from the `aim_1` StarGraph.

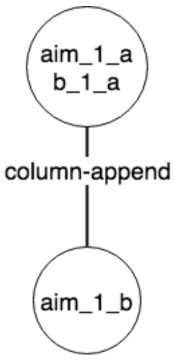


Figure 5.10: Closure graph after combining edges

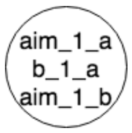


Figure 5.11: Closure graph after merging column append edges

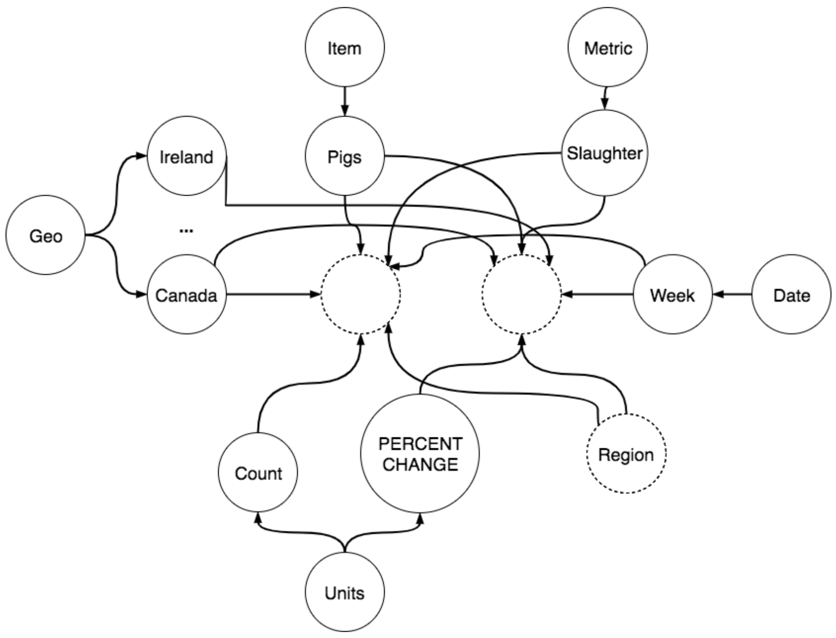


Figure 5.12: Constellation created from `aim_1` and `b_1` StarGraphs

Table 5.3: Integration Steps produced by the collapsed graph

Step	Source_1	Source_2	Method	Join
1	{`aim_1_a`}	{`b_1_a`}	row-append	all

2	{`aim_1_a`}
	{`b_1_a`}
		{`aim_1_b`}	column-append	DATE, GEO, ITEM

Once this function finishes, it provides the Constellation and the integrations determined to the Materialise function to construct the mappings.

The GenerateTargetSchema and GenerateMappings functions

The `GenerateTargetSchema` and `GenerateMappings` functions for a Constellation is similar to that of a StarGraph, with the main differences being the number of sources to be accessed in the data lake and the complexity of the mapping rules. The `GenerateTargetSchema` function creates a JSON file describing the target schema for the integrated sources in readiness for subsequent materialisation. Example 5.2.6 shows the target schema generated for the constellation created from the `aim_1` and `b_1` StarGraphs. The target schema lists all metamodel and optional attributes for a Constellation per data source under the `source_meta` property. The "facts" property lists the individual sources that constitute a fact and their metamodel attributes. Example 5.2.7 details the mappings produced by the Constellation.

Example 5.2.6. Sample target Schema `aim_1` and `bb_1` Constellation.

```
{
    "name" : "aim\_1,bb\_1",
    "source_meta" : {
        "aim\_1" : {
            "DATE" : [
                {
                    name:"2016-03-05",
                    type:"WEEK",
                    src:"//table/tbody/tr[2]/th"
                },
                {
                     name:"2016-03-05",
                    type:"WEEK",
                     src:"//table/tbody/tr[3]/th"
                }
            ],
            "GEO" : [
                {
                    name:"CANADA",
                    type"Geo",
                    src:""//table/tbody/tr[6]/th""
                },...
            ],
            "ITEM" : [
            {
                name:"PIGS",
                type:"Item",
                src:"STATIC"
            }
            ],..
        }
    },
    "facts" : [
        {
            "sources" : [ "aim\_1","b\_1"
            ],
            "META_VAL" : [
                {
                    "attr" : "DATE",
                    "val" : "WEEK"
                },
                {
                    "attr" : "GEO",
                },
                {
                    "attr" : "METRIC",
                    "val" : "COUNT"
                },
                {
                    "attr":"ITEM",
                    "val":"PIG"
                },
                {
                    "attr":"UNIT",
                    "val":"Count"
                }
            ],
            "OPTIONAL":[
                {
                    "attr":"Region"
                }
            ]
        },...
        ]
}
```

Example 5.2.7. Sample mappings for `aim_1` **and** `bb_1` **Constellation.**

```
("//table[1]/tbody/tr[3]/th[1]") -> West-CANADA,
("//table[1]/tbody/tr[3]/th[2]")->West-CANADA.su...,
("//table[1]/tbody/tr[4]/th[2]")-> West.subdimensions["Saskatchewan ..,
("//table[1]/tbody/tr[5]/th[2]")-> WestWest-CANADA.subdimen...
("//table[1]/tbody/tr[6]/th[1]")-> East-CANADA,
...,
(//table/tbody/tr/td[1])->Date,
(//table/tbody/tr/td[2])->Total
```

5.2.6 The `Materialise` function

The role of the `Materialise` function is to use the target schema and mappings generated by the previous functions to produce a populated integrated data mart. Similar to a StarGraph, the same methodology used to transform a dataset into an Annotated Tuple Set (ATS), is first applied on all sources required for population. In order to overcome terminology issues, each tuple set must undergo term mapping, to ensure canonical terms are used within the source data. For the `aim_1` dataset, the ATS before term mapping can be seen in Example 4.3.1 and the ATS after term mapping can be seen in Example 5.2.8. The difference between the two lies in the property `name` which has been term mapped from 'West' to 'West-Canada'

Example 5.2.8. Annotated tuple set for aim_1 after term mapping.

```
    ({name:Federal, src:"...", val:"Federal"},
    {name:"2016-03-05", src:"...",
    val:"2016-03-05"},
    {name:"West-Canada",src:"...", val:"West"},
    {name:"Total West",src:"...",
    val"Total West"},
    {name:"Total",src:"...",val:"226,171"}),
    ...
```

Once an ATS has been created for each data source and term mapping has finished, the next step is to extract the dependency sets identified by the `DetermineStrategy` function. This produces a series of named annotated tuple sets as shown in Example 5.2.9.

Example 5.2.9. Named ATS for aim_1.

```
  { name:"aim\_1\_a",
   ats:[{name:Federal, src:"...", val:"Federal"},
    {name:"2016-03-05", src:"...",
    val:"2016-03-05"},
    {name:"West-Canada",src:"...", val:"West"},
    {name:"Total West",src:"...",
    val"Total West"},
    {name:"Total",src:"...",val:"226,171"},
    ...]
    }
    {
      name:"aim\_1\_b",
      ats:[{name:Federal, src:"...", val:"Federal"},
    {name:"2016-03-05", src:"...",
    val:"2016-03-05"},
    {name:"West-Canada",src:"...", val:"West"},
    {name:"Total West",src:"...",
    val"Total West"},
    {name:"PChange",src:"...",val:"5"},
    ...]
    }
```

After these sets have been extracted, the steps used to collapse the combination graph are executed step by step. If two dependency sets have a `row-append` strategy, they are simply appended one after the other, as shown in Example 5.2.10. If two sets are `column-append` then they are integrated based on the common join attributes identified in Table 5.3. Once all steps have been executed, the data mart is populated.

Example 5.2.10. Named ATS for aim_1.

```
  { name:"aim\_1\_a, b\_1\_a",
   ats:[{name:Federal, src:"...", val:"Federal"},
    {name:"2016-03-05", src:"...",
    val:"2016-03-05"},
    {name:"West-Canada",src:"...", val:"West"},
    {name:"Total West",src:"...",
    val"Total West"},
    {name:"Total",src:"...",val:"226,171"},
    ...,
     {name:"Date", src:"...", val:"2017-01-07"},
    {name:"Total",src:"...",val:"51,051"})
    ]
    }
    {
      name:"aim\_1\_b",
      ats:[{name:Federal, src:"...", val:"Federal"},
    {name:"2016-03-05", src:"...",
    val:"2016-03-05"},
    {name:"West-Canada",src:"...", val:"West"},
    {name:"Total West",src:"...",
    val"Total West"},
    {name:"PChange",src:"...",val:"5"},
    ...]
    }
```

5.3 Integration Data Flows and Mappings

In order to fully illustrate what takes place during the creation of a Constellation, we now present a case study integration, showing the required schema transformations. We then present a discussion on the construction of the constellations for each of the three case studies that are used for our ongoing evaluation.

5.3.1 Examining Graph Convergence

As part of this integration case study, we will also show that the order of which sources are integrated does not change the final schema configuration for the sources used in our case studies. A data mart was constructed from five sources using different integration orders to demonstrate this fact. We chose to use five sources as the number of potential orderings of a set of sources n is $n!$. For five sources, this presents us with 120 different possible configurations which still result in the same Constellation structure. Of course, intermediate steps will produce different graph structures which will converge once all sources have been processed.

The five sources chosen were `aim_1`, `b_1`, `p_1`, `bp_1` and `c_1` from Case Study 1 (section 3.4.1). For the purpose of this experiment, all missing metamodel attributes were previously entered. Figure 5.13 outlines the resulting constellation, detailing located metamodel values and facts, and how they are joined at each step. The facts are represented as nodes with dotted circles and the edges indicate the joining attributes which constitute each fact. Due to the large number of possible permutations, Table 5.4 presents the first integration step of each permutation and selected integration steps. The column `Step` indicates at which step of the integration process, that output represents. As we use a ladder strategy there are $n - 1$ steps to integrate n sources.

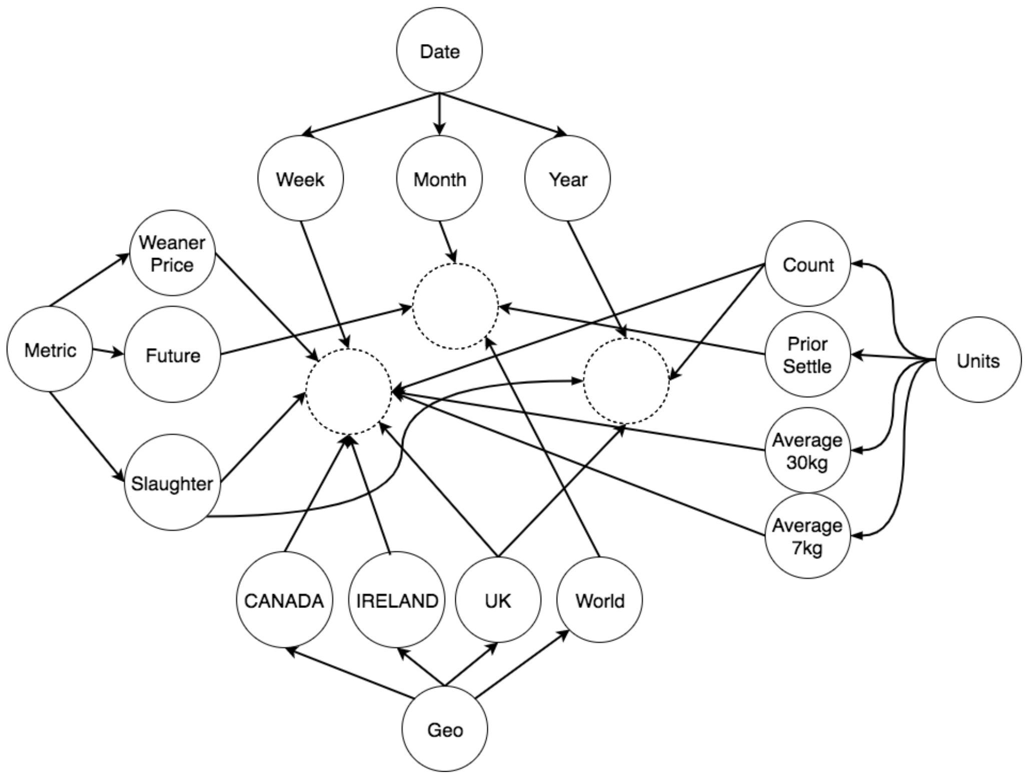


Figure 5.13: Constellation created from the five experimental data sources

In Table 5.4, we can see that the source `aim_1` and `b_1` were joined based on their `Date`, `Item`, `Metric` and `Unit` metamodel values, with the presence (or absence) of these values indicating the common attributes found between two data sources. As we can see from the table, the source `c_1` only joined on the `Item` attribute for all data sources. This is because `c_1` lists future pig prices monthly worldwide. The last two rows in the table indicate how a graph will *always* converge. The second last row to integrates the `aim_1` dataset with all others, and joins on the `Date`, `Item`,`Metric` and `Unit` metamodel values. This high degree of integration is due to the fact that the source `b_1` was already within the constellation. The same can be said for the last row of the table which attempts to integrate `b_1` into a constellation

Table 5.4: Small subset of Integration steps from 120 possible configurations

Step	Source_1	Source_2	JOIN_ON
1	{aim_1}	{b_1}	DATE, ITEM, METRIC, UNIT
1	{aim_1}	{p_1}	ITEM, METRIC, UNIT
1	{aim_1}	{bp_1}	DATE, ITEM
1	{aim_1}	{c_1}	ITEM
1	{b_1}	{p_1}	ITEM, METRIC, UNIT
1	{b_1}	{bp_1}	DATE, ITEM
1	{b_1}	{c_1}	ITEM
1	{p_1}	{bp_1}	ITEM, METRIC, UNIT
1	{p_1}	{c_1}	ITEM
1	{bp_1}	{c_1}	ITEM
2	{aim_1},{bp_1}	{p_1}	ITEM, METRIC, UNIT
2	{b_1},{c_1}	{aim_1}	DATE, ITEM, METRIC, UNIT
3	{bp_1},{p_1},{c_1}	{b_1}	ITEM, METRIC, UNIT
3	{c_1},{b_1},{bp_1}	{p_1}	ITEM, METRIC, UNIT
4	{aim_1},{b_1},{p_1},{bp_1}	{c_1}	ITEM
4	{aim_1},{b_1},{p_1},{c_1}	{bp_1}	ITEM, METRIC, UNIT
4	{aim_1},{b_1},{bp_1}, {c_1}	{p_1}	ITEM, METRIC, UNIT
4	{b_1},{p_1},{bp_1}, {c_1}	{aim_1}	DATE, ITEM, METRIC, UNIT
4	{aim_1},{p_1},{bp_1}, {c_1}	{b_1}	DATE, ITEM, METRIC, UNIT

which contains `aim_1`. As this represents only an summary, we will now present a step by step approach for two integration combinations.

5.3.1.1 Configuration 1

Table 5.5: Integration steps for Approach 1

Step	Source_1	Source_2	JOIN_ON
1	{aim_1}	{b_1}	DATE, ITEM, METRIC, UNIT
2	{aim_1}, {b_1}	{p_1}	ITEM, METRIC, UNIT
3	{aim_1}, {b_1}, {p_1}	{bp_1}	DATE, ITEM, METRIC, UNIT
4	{aim_1}, {b_1}, {p_1}, {bp_1}	{c_1}	ITEM

This approach integrated the five sources in the following order: ⟨`aim_1`, `b_1`, `p_1`, `bp_1`, `c_1`⟩, with table 5.5 showing the steps involved. The first step integrated `aim_1` and `b_1`. There was a high degree of common metamodel attributes between the two sources, resulting in them being integrated on the values `Date`, `Item`, `Metric`, `Unit`. The second stage integrated `p_1` with this constellation where: `p_1` joined to both aim_1 and b_1 on the Item, Metric and Unit metamodel values but not Date as it is monthly while aim_1 and b_1 are weekly. The third step integrated the source bp_1. This source joined aim_1 and b_1 on the Date and Item values, and joined p_1 on the Item, Metric and Units values. The final step integrated the c_1 source. This source integrated on the Item value with all other sources, and on nothing else.

5.3.1.2 Configuration 2

This approach integrated the five sources in the following order: ⟨bp_1, c_1, aim_1, p_1, b_1⟩ and Table 5.6 shows the steps involved. The first step integrated the sources bp_1 and c_1 on the Item value. The second step integrated the source aim_1 with bp_1 and c_1. It integrated on bp_1 on the Date and Item values and on c_1 on the Item value. The third step integrated the source p_1. This source integrated on bp_1 on the Item, Metric and Unit values, c_1 on the Item value and aim_1 on the Item, Metric and Unit values.The final stage integrated the source b_1. This source integrated on bp_1 on the Date and Item values, on c_1 on the Item value, on aim_1 on the Date, Item, Metric and Unit values and finally on p_1 on the Item, Metric and Unit values.

Table 5.6: Integration steps for Approach 2

Step	Source_1	Source_2	JOIN_ON
1	{bp_1}	{c_1}	ITEM
2	{bp_1}, {c_1}	{aim_1}	DATE, ITEM
3	{bp_1}, {c_1}, {aim_1}	{p_1}	ITEM, METRIC, UNIT
4	{bp_1}, {c_1}, {aim_1}, {p_1}	{b_1}	DATE, ITEM, METRIC, UNIT

Summary. These experiments demonstrate that for the sources used, the integration order does not affect the constellation. This is due to the presence of the metamodel and ontology. The metamodel values and the term and type map processes, ensure that there is only one way two individual sources may be integrate and as such, there is only one final possible configuration of a Constellations structure regardless of the order of integration.

5.3.2 Constellation construction for Case Study 1

Case Study 1 (section 3.4.1) attempts to construct a pig prices data mart from 13 separate sources. Table 5.7 outlines the 12 integration steps taken in order to construct this data mart. `Step` is the integration step, `Source_1` and `Source_2` detail what sources are being integrated at each stage, `Cons` is shorthand for Constellation. `Join on` details on what metamodel attributes the integration joined on and `User Supplied` indicates metamodel attributes which were supplied by a user for this integration step.

Table 5.7: Integration steps for Case Study 1

Step	Source_1	Source_2	Join on	User Supplied
1	{aim_1}	{aim_2}	GEO, ITEM, METRIC, UNIT	ITEM, METRIC, UNIT
2	Cons	{b_1}	DATE, ITEM, METRIC, UNIT	GEO, ITEM, METRIC, UNIT
3	Cons	{b_2}	ALL	GEO, ITEM, METRIC, UNIT
4	Cons	{p_1}	DATE, ITEM, METRIC, UNIT	ITEM, METRIC
5	Cons	{p_2}	ALL	GEO, ITEM
6	Cons	{p_3}	ALL	GEO, METRIC
7	Cons	{bp_1}	ALL	ITEM, UNIT
8	Cons	{bp_2}	ALL	ITEM, UNIT
9	Cons	{c_1}	DATE, ITEM	GEO, UNIT
10	Cons	{c_2}	ALL	GEO, UNIT
11	Cons	{imf}	DATE	METRIC, UNIT
12	Cons	{usda}	ALL	NONE

Step 1 attempted to integrate the `aim_1` and `aim_2` datasets. These sources shared `Geo`, `Metric`, `Item` and `Units` values but failed to integrate into the same fact as `aim_1` is weekly and `aim_2` is yearly. Step 2 integrated the `b_1` dataset with the Constellation. This source integrated with `aim_1` on the `Date`, `Item`, `Metric` and `Units` values. Step 3 integrated `b_2` however as this data source is identical to `b_1` in structure, a full integration occurred. Step 4 integrated `p_1` with the Constellation, this source joined on the `Date`, `Item`, `Metric` and `Unit` values. Steps 5 and 6 integrated `p_2` and `p_3` both sources integrated on all values. Step 7 and 8 integrated the `bp_1` and `bp_2` soures. These joined on all attributes. Step 9 integrated the `c_1` source. This only joined on the `Date` and `Item` values as it presents future prices, which have previously been unseen by the Constellation. Step 10 integrated `c_2`, this proceeded to fully integrate with `c_1`. Step 11 added the `imf` data source. This source only joined on the `Date` attribute, as it provides exchange rate reports, it added a new `Item`, `Metric` and `Unit` values to the Constellation. Step 12 added the `usda` data source. This source fully integrated into the constellation with no user supplied metamodel values.

Overall, this produced a reasonably complex data mart, consisting of three facts as determined by the differing Date granularities. The produced constellation can be seen in Figure 5.14. Due to the size of the mappings and target schema produced it is provided in Appendix E

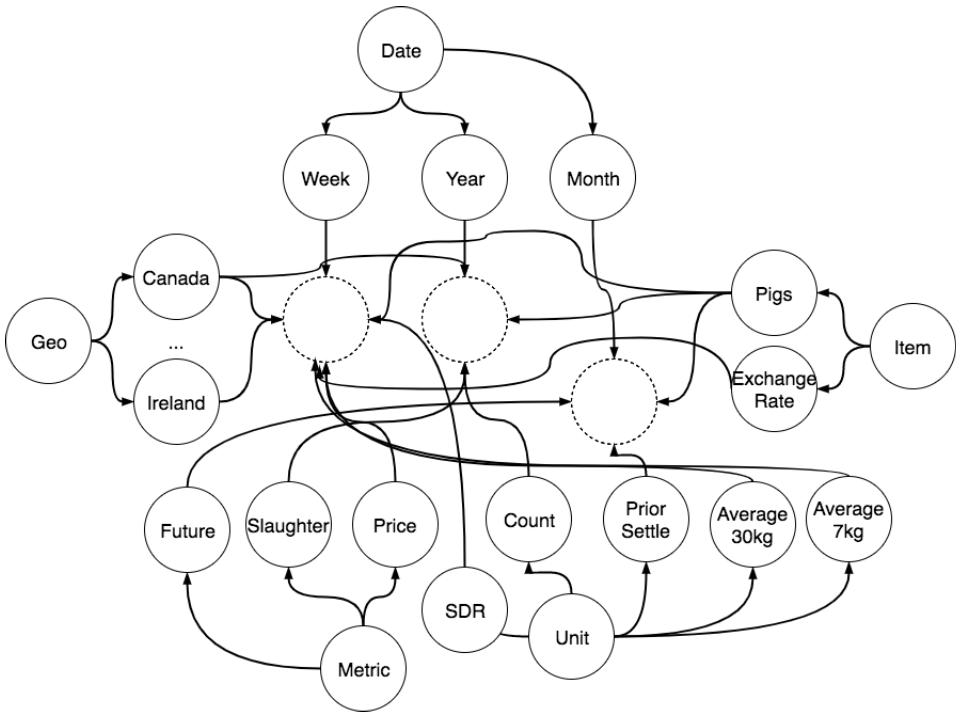


Figure 5.14: Case Study 1: Final Constellation

5.3.3 Constellation construction for Case Study 2

Case Study 2 (section 3.4.2) constructs a data mart from 9 separate sources. All CSV sources were incredibly sparse, consisting of a Date dimension and a measure, requiring a user to enter multiple metamodel attributes. Table 5.8 details the integration steps for the construction of this data mart. Most data sources were semantically similar, detailing the weekly price of a vegetable oil worldwide in USD. As such, once metamodel attributes were supplied these all proceeded to integrate on the `Date`, `Geo`, `Metric` and `Units` values (Steps 1-7). However, the final step for the source `gdt` only integrated on the `Geo`, `Metric` and `Units` values as it is a daily data source, while the others are weekly. This discrepancy resulted in the construction of two fact tables with shared dimensions `Geo`, `Metric` and `Units`, one for the weekly sources and another for the daily source.

Table 5.8: Integration steps for Case Study 2

Step	Source_1	Source_2	Join on	User Supplied
1	{`pp`}	{`pr`}	DATE, GEO, METRIC, UNITS	GEO, METRIC, ITEM, UNITS
2	Cons	{`ps`}	DATE, GEO, METRIC, UNITS	GEO, METRIC, ITEM, UNITS
3	Cons	{`psu`}	DATE, GEO, METRIC, UNITS	GEO, METRIC, ITEM, UNITS
4	Cons	{`ren`}	DATE, GEO, METRIC, UNITS	GEO, METRIC, ITEM, UNITS
5	Cons	{`cn`}	DATE, GEO, METRIC, UNITS	GEO, METRIC, ITEM, UNITS
6	Cons	{`po`}	DATE, GEO, METRIC, UNITS	GEO, METRIC, ITEM, UNITS
7	Cons	{`so`}	DATE, GEO, METRIC, UNITS	GEO, METRIC, ITEM, UNITS
8	Cons	{`gdt`}	UNITS, GEO, METRIC	GEO, METRIC, UNITS

Figure 5.15 details the Constellation constructed from this integration. Due to the size of the mappings and target schema produced it is shown in Appendix F.

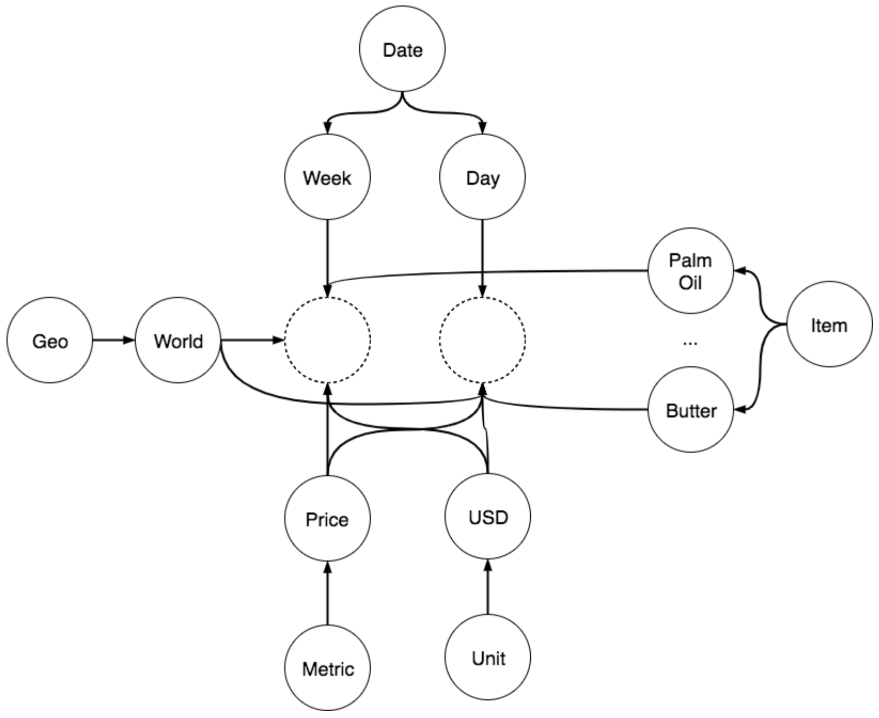


Figure 5.15: Case Study 2: Final Constellation

5.3.4 Constellation construction for Case Study 3

Case Study 3 (section 3.4.3) constructs a data mart from five sources, all centred around dairy production and deliveries. Table 5.9 outlines the integration steps for this data source. All sources, despite structure differences due to HTML, contained many metamodel attributes, requiring the user to only enter the `Geo` and `Item` values. As all sources were of the same date granularity (weekly) only one fact was produced. Figure 5.16 shows the Constellation produced by this Case Study. Due to the size of the mappings and schema produced they are shown in Appendix G .

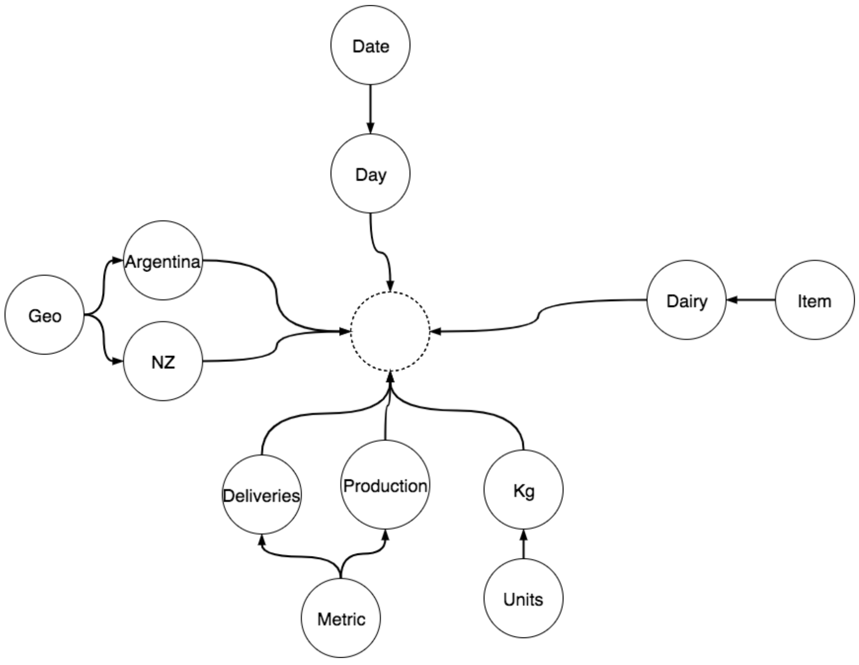


Figure 5.16: Case Study 3: Final Constellation

Table 5.9: Integration steps for Case Study 3

Step	Source_1	Source_2	Join on	User Supplied
1	{amd}	{wdp}	DATE, ITEM, METRIC, UNITS	GEO, ITEM
2	Cons	{usda_2}	DATE, ITEM, METRIC, UNITS	NONE
3	Cons	{mpg}	DATE, ITEM, METRIC, UNITS	GEO
4	Cons	{nzmp}	DATE, ITEM, METRIC, UNITS	GEO, ITEM

5.4 Summary

In this chapter, we detailed our approach for integrating heterogeneous data sources through the use of our ontology assisted `termMap` and `typeMap` functions. These functions coupled with the metamodel provide a means of both structurally and semantically integrating data sources. Using the metamodel values found within a data source, we detail how an integration strategy may be determined and how to materialise a Constellation. Finally, our evaluation examines the integration steps for three case studies and shows how the integration order does not affect the final outcome due to our ontology-assisted integration approach. However, despite the ability to construct integrated data marts, some issues centred around optimisation and redundancy in data sources still remain. These issues will be addressed in the following chapter.

Chapter 6

Deleting Redundant Mappings

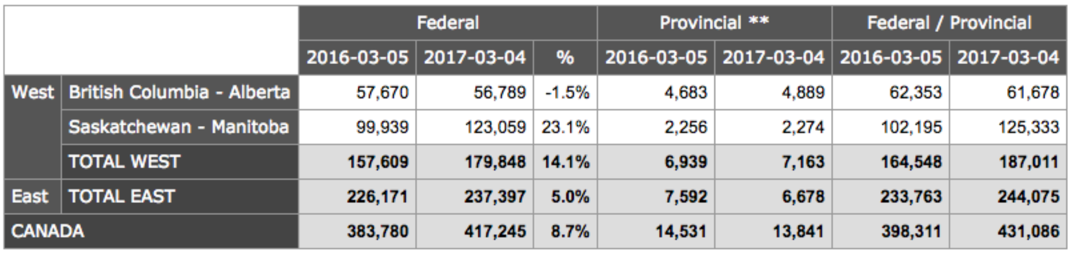
While the StarGraph system is capable of constructing a data mart from previously unseen sources, there still remains issues of redundancy within the mart. These issues, while not interfering with the usefulness of a data mart, are undesirable as they lead to increased population times for data marts. In this chapter, we will identify and classify the types of redundancies found within the source data in sections 6.1 to 6.4. In section 6.5, we present the results of a series of experiments which implemented these optimisations to demonstrate the efficiencies that were achieved.

6.1 HTML formatting: Blank Value

A `blank value` redundancy is a redundancy where blank attributes are present within a constructed data mart. Blank value redundancies occur in HTML data sources, resulting from the incorrect use of HTML tags to dictate page layout. For example, consider the `aim_1` source shown in Figure 6.1, which is a HTML table. Note the whitespace at the top-left of the table. Such space is required in order to accurately convey information to the user. However, in order to achieve such a table, a developer must edit the tables structure by providing blank `td` and `th` elements.

Unfortunately, these elements are also required to convey data within a table. As such, a StarGraph would identify these elements and construct nodes within a Star-

Graph for them. Such nodes act in a similar manner to other StarGraph nodes, their datatype is assumed to be a String, and they are classified as either Dimensions or Dimension attributes depending on their location within the table. As they have a place within the HTML table, mappings are constructed for them and subsequently, they are populated during the materialisation process. As these elements serve only to direct the visual layout of a table to a user, semantically they serve no purpose within a StarGraph or data mart. Additionally, their removal results in fewer mappings which ultimately reduces the population time.



		Federal			Provincial **		Federal / Provincial	
		2016-03-05	2017-03-04	%	2016-03-05	2017-03-04	2016-03-05	2017-03-04
West	British Columbia - Alberta	57,670	56,789	-1.5%	4,683	4,889	62,353	61,678
	Saskatchewan - Manitoba	99,939	123,059	23.1%	2,256	2,274	102,195	125,333
	TOTAL WEST	157,609	179,848	14.1%	6,939	7,163	164,548	187,011
East	TOTAL EAST	226,171	237,397	5.0%	7,592	6,678	233,763	244,075
CANADA		383,780	417,245	8.7%	14,531	13,841	398,311	431,086

Figure 6.1: Sample of the `aim_1` data source

Detecting these redundancies is generally straightforward. When constructing a StarGraph, a check on a node to determine if it is empty would suffice in detecting these nodes. The system must ensure that when determining if these nodes are blank, it may be because they direct layout, or because they represent a *missing value* within a dataset. A value which serves to direct layout, holds no functional dependencies to other nodes, while nodes representing a missing value must be captured by the StapGraph as appropriate.

Within a StarGraph, a node is functionally dependant on other nodes if it has parent nodes. Algorithm 6.1 checks to see if a node is a `blank-value` redundancy. By adding this check as a classification within the `ClassifyNode` function (Section 4.2.2.2), these nodes can be classified as `containers` and subsequently removed.

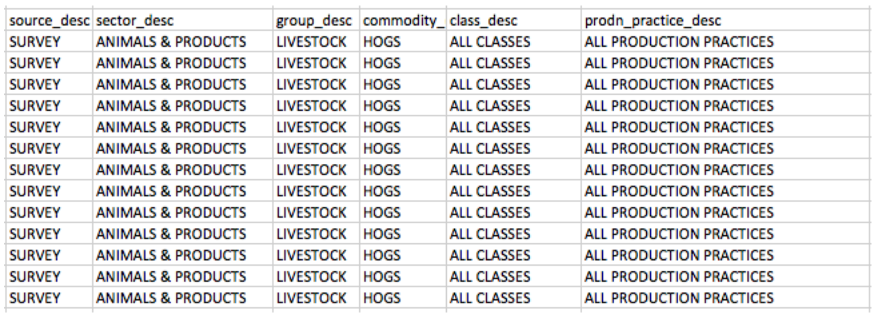
Algorithm 6.1 Algorithm to detect blank value redundancies

```
1: function isBlankValue(node)
2:     return isEmpty(node.name) AND !hasParents(node)
3: end function
```

Case Study Analysis for Blanks. This redundancy only occurs within HTML data sources. Of the 84 usable agri data sources, 46 (54%) were HTML. Of these 46 sources, the `blank value` redundancy was present within 5 sources (10.8%).

6.2 CSV Formatting: Repeating Value

A repeating value redundancy appears within `CSV` datasets. This is a result of the data being presented in a flattened format. While other formats may convey relationships through structure, CSV must use redundancy. This appears in the data as repeating values for a column. For example, the `usda` dataset has a high degree of redundancy across dimensional values. Figure 6.2 shows a small sample of the `usda` data source. Note the repeating values for all columns.



source_desc	sector_desc	group_desc	commodity_	class_desc	prodn_practice_desc
SURVEY	ANIMALS & PRODUCTS	LIVESTOCK	HOGS	ALL CLASSES	ALL PRODUCTION PRACTICES
SURVEY	ANIMALS & PRODUCTS	LIVESTOCK	HOGS	ALL CLASSES	ALL PRODUCTION PRACTICES
SURVEY	ANIMALS & PRODUCTS	LIVESTOCK	HOGS	ALL CLASSES	ALL PRODUCTION PRACTICES
SURVEY	ANIMALS & PRODUCTS	LIVESTOCK	HOGS	ALL CLASSES	ALL PRODUCTION PRACTICES
SURVEY	ANIMALS & PRODUCTS	LIVESTOCK	HOGS	ALL CLASSES	ALL PRODUCTION PRACTICES
SURVEY	ANIMALS & PRODUCTS	LIVESTOCK	HOGS	ALL CLASSES	ALL PRODUCTION PRACTICES
SURVEY	ANIMALS & PRODUCTS	LIVESTOCK	HOGS	ALL CLASSES	ALL PRODUCTION PRACTICES
SURVEY	ANIMALS & PRODUCTS	LIVESTOCK	HOGS	ALL CLASSES	ALL PRODUCTION PRACTICES
SURVEY	ANIMALS & PRODUCTS	LIVESTOCK	HOGS	ALL CLASSES	ALL PRODUCTION PRACTICES
SURVEY	ANIMALS & PRODUCTS	LIVESTOCK	HOGS	ALL CLASSES	ALL PRODUCTION PRACTICES
SURVEY	ANIMALS & PRODUCTS	LIVESTOCK	HOGS	ALL CLASSES	ALL PRODUCTION PRACTICES
SURVEY	ANIMALS & PRODUCTS	LIVESTOCK	HOGS	ALL CLASSES	ALL PRODUCTION PRACTICES
SURVEY	ANIMALS & PRODUCTS	LIVESTOCK	HOGS	ALL CLASSES	ALL PRODUCTION PRACTICES

Figure 6.2: Sample of `usda` data source

While this redundancy does not interfere with the semantic structure of a data mart, it must be addressed in order to optimise the mappings of a StarGraph (or Constellation) by determining if the value for a column is the same across all rows. If this is the case, this value can be stored and read only once instead of n times (where n is the number of rows within the file). In order to detect this redundancy, a Set must be constructed containing the values for a column. If the size of the constructed set is 1, then there exists a single value for every row within the dataset and such, this column exhibits the `repeating value` redundancy.
In order to detect this redundancy, the entire file must be scanned, with the effect that no optimisation can be achieved for the first copy of this stream. However, subsequent versions of this file within the data lake can make use of this optimisa-

tion as the redundancies have previously been identified. During population, this optimisation reduces the number of mappings applied to an Annotated Tuple Set, replacing the `Get` commands with a Static value within the target schema. If there is more than one repeating value within a CSV file, the functionality to detect this redundancy can be further extended through the use of predictive algorithms such as decision trees as discussed in our future work 8.2.3.

Case Study Analysis for CSV Redundancies. Of the 84 usable sources, 16 (19%) were CSV files. Of these 16 files, 3 contained the repeating value redundancy (18.7%).

6.3 StarGraph Integration: Duplicate Node

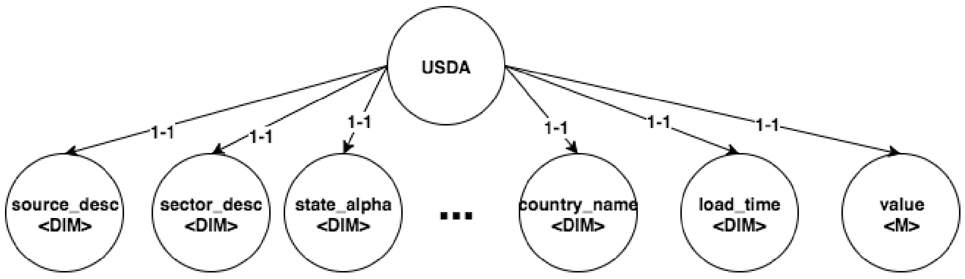


Figure 6.3: Sample of the `usda` StarGraph

A `duplicate node` redundancy occurs when a data set contains two nodes which are structurally similar. For example, the `usda` dataset contains two attributes `state alpha` and `country name`. The value for `state alpha` is a single repeating value `US` and the value for `country name` is also a single value `United States`. These repeating values can be determined using the optimisation described in Section 6.2. However, there exists a structural redundancy within the StarGraph. As both values simply relate to the country name, only one node is required in order to construct a data mart. As these values are semantically similar, the `termMap` and `typeMap` functions (Section 5.2.2 and Section 5.2.1) can be used to determine if these two nodes are *identical*.

Figure 6.3 shows the StarGraph constructed from the `usda` data source. The two nodes `state_alpha` and `country_name` are semantically identical. By applying the termMap function we can see that they resolve to the same term United States (Figure 6.4).

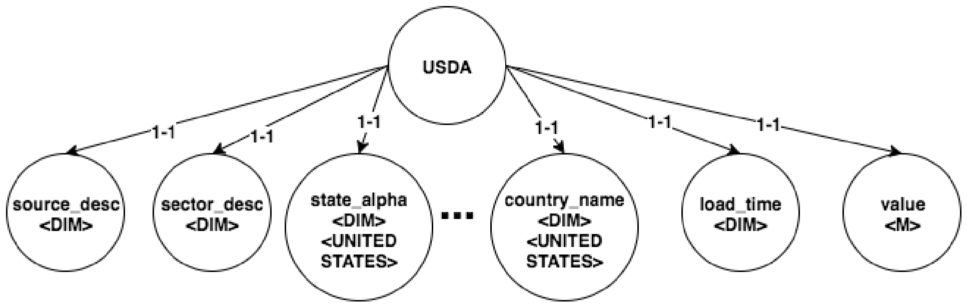


Figure 6.4: usda StarGraph after termMap function

However, even though both nodes resolve to the same term, they may represent different data types. As such, the typeMap function must be applied. Figure 6.5 shows the usda StarGraph after the application of the typeMap function. From this illustration, we can see that both state_alpha and country_name nodes contain the same term and type and as such are structurally identical.

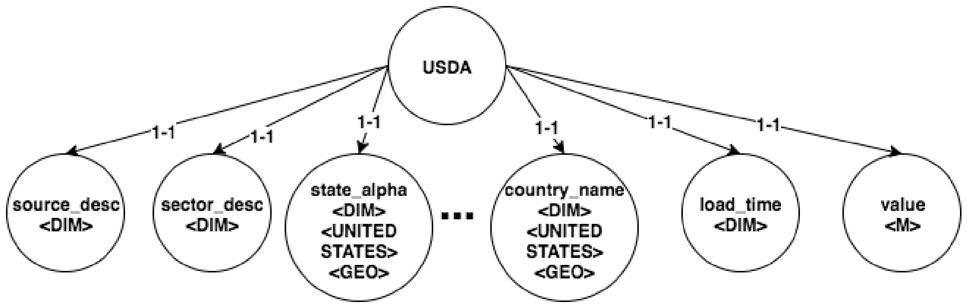


Figure 6.5: usda StarGraph after typeMap function

Case Study Analysis for Duplicate Node. From all CSV data sources used in this evaluation, one file out of 16 exhibited this redundancy.

6.4 Valueless Node

Valueless node redundancies are items within a StarGraph (or Constellation) which should not be there as they provide no value, or are unusable within the Data Mart. An example of this kind of redundancy lies in the imf dataset (Figure 6.6). Within this StarGraph, the nodes ReportName, Disclaimer and USER_SELECTIONS (and all of it's sub-nodes) should not be present within a data mart. This is because all of these nodes contain information about the query which was executed on the IMF website to obtain this data source.

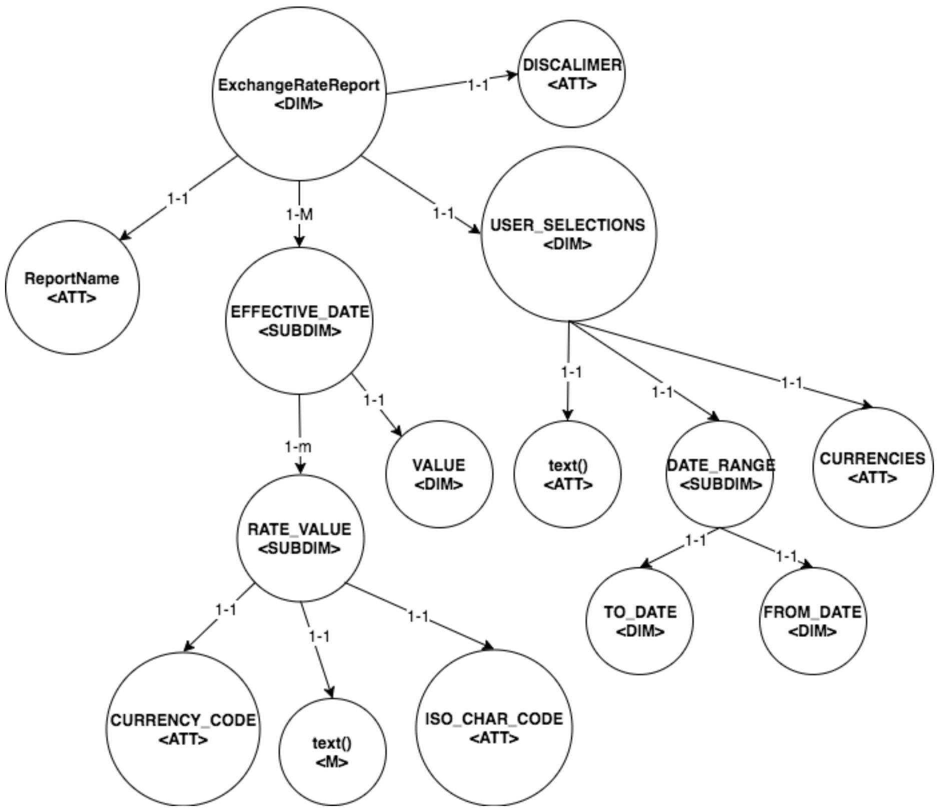


Figure 6.6: `imf` dataset with semantic redundancies

However, as all of these nodes contain information, they are considered valid dimensions by the StarGraph. Ultimately, the data contained within these nodes seems identical to valid data as far as our StarGraph system is concerned. As such, the only meaningful way to determine if these nodes are redundant is for a user to specify them as of having *No Value*. This can be achieved in two ways. Firstly, the user may edit the StarGraph mapping and structure to remove these nodes manually. Secondly, the addition of a redundant type to the lightweight ontology which could be used to assign these nodes with a redundant type during the `typeMap` function (Section 5.2.2).

6.5 Optimisation Analysis

As the end goal of each of these optimisations is to either remove nodes, or reduce reads to source datasets, these optimisations ultimately reduce the population time of a Constellation or StarGraph by reducing the number of required mappings. As such, for all data sources identified with these redundancies, we will examine their population times with and without these optimisations in order to determine their impact on the population times of a data mart.

6.5.1 Blank Value Optimisation

Table 6.1 outlines the evaluation of *Blank Value* redundancies found for HTML data sources. The column `Source` refers to the experimental ID given to the data source; `Redundancies` refers to the number of redundancies found within the source (the number of blank `th` and `td` cells); Pop_a is the population time in milliseconds without optimisations present; and Pop_b is the population time in milliseconds with the optimisations; and finally, `Difference` is the difference between the population times in milliseconds.

Table 6.1: Results for Blank Value Optimisation

Source	**Redundancies**	Pop_a	Pop_b	**Difference**
`aim_1`	1	446	445	1
`bb`	4	109	99	10
`adhb_1`	48	364	277	87
`adhb_2`	1	199	194	5
`mlk`	3	61	59	2

From table 6.1, we can see while that the optimisations reduce the population times for the data sources, their effect is small. However, we are using comparatively small datasets and as the number of redundancies increases, the savings posed by the optimisation increases, with the `adhb_1` dataset seeing the largest reduction in population times going from 364ms to 277ms.

6.5.2 Repeating Value Optimisation

Table 6.2 outlines the population times of all sources which have *Repeating Value* redundancies. It shows the experimental id of the source (`Source`); the number of columns within the source containing repeating values `Redundancies`; `Rows` is the number of rows in the CSV file; population time in milliseconds of the source without optimisations Pop_a; population time once optimisations are applied Pop_b; and the difference in milliseconds gained by the optimisation.

Table 6.2: Results for Repeating Value Optimisation

Source	**Redundancies**	**Rows**	Pop_a	Pop_b	**Difference**
`usda_1`	27	1812	797	236	561
`usda_2`	25	167	74	32	42
`statcan`	1	520	43	40	3

The effect on population time by these redundancies is a measure of both the number of redundancies within a file (columns) and by the number of instances of the data (rows).

From this we can see that the `usda_1` dataset containing both the highest number of redundant columns and the highest number of rows, stands to benefit the most from these optimisations, having a reduction in population time of 561 milliseconds. While the `usda_2` source saw a reduction of 42 milliseconds with the `statcan` dataset having a reduction of 3 milliseconds.

6.5.3 Duplicate Node Optimisation

The single data source which contained a duplicate node redundancy additionally had numerous repeating value redundancies examined previously. As such, we will now examine the effect the duplicate node redundancy has on population times, and will present the population time for this source with both the repeating value and duplicate node redundancies removed.

Table 6.3 examines the effect in population times between various redundancies found within the `usda_1` source. `Optimisation` refers to the optimisations applied to the population of the source. Pop_a is the time without optimisations, Pop_b is the time taken, with optimisations applied, and `Difference` is the difference in milliseconds between the two.

Without optimisations the `usda_1` source has a population time of 797 milliseconds. The `Duplicate Node` optimisation combined two nodes into one, resulting in a saving of 53 milliseconds. Combining the optimisations for `duplicate node` and `repeating value` changes the population time from 797 milliseconds to 230ms. This provides a reduction of 567 milliseconds, which provides a 6 milliseconds increase on top of the benefits gained by the `repeating value` optimisation alone.

Table 6.3: Results of Duplicate Node Optimisation for `usda` source

Source	**Optimisation**	Pop_a	Pop_b	**Difference**
`usda_1`	None	797	797	0
`usda_1`	Duplicate Node	797	744	53
`usda_1`	Duplicate Node Repeating Value	797	230	567

6.5.4 Valueless Node Optimisation

Table 6.4 details the optimisation gained by the `imf` dataset by removing unnecessary nodes. For the single source containing unnecessary nodes there was a small benefit of 7 milliseconds. The benefit in this instance was small, as these nodes only appear once per file and not per instance as seen in CSV files.

Table 6.4: Results for Valueless Node Optimisation

Source	**Redundancies**	Pop_a	Pop_b	**Difference**
`imf`	8	20	13	7

6.6 Summary

In this chapter, we identified and classified a number of redundancies that were encountered when constructing StarGraphs from the 84 agri data sources used in our evaluation case studies. We presented methodologies which are used to detect and remove each of these redundancies. Our analysis showed increased efficiency in all cases, with a reduction in data mart population times once optimisations were used to remove redundancies. While in some cases, the benefits are quite small, for a number of sources, savings of up to half a second can be made by applying these optimisations.

At this stage, our methodology has been presented in full. We have also presented analyses in chapter 4 to illustrate the feasibility of our approach in terms of the numbers of data sources that were usable in data marts: i.e. those with extractable dimensions and fact metadata. In this chapter, we showed how efficiencies in data mart population could be achieved. The final part of this dissertation is a complete evaluation of our approach using a series of case studies and the assistance of agri domain users.

Chapter 7

Evaluation

The goal of the StarGraph system is to construct a data mart from web data to meet specific user (or business) requirements. In order to evaluate our approach, it is necessary to determine the effectiveness of the StarGraph system in building data marts in a series of real-world settings. The three case studies presented in chapter 3 provide the user requirements from an agri collaborator that forms part of this research. We evaluate these three data marts with regards to their construction quality and population time. In all three case studies, we compare manually created data marts with data marts constructed using the StarGraph system. In sections 7.1 to 7.3, we provide a detailed analysis and discussion for the three separate data marts and conclude with a comparison of results across all case studies in section 7.4.

7.1 Case Study 1: Price Prediction Data Mart

This first case study requires the construction of a data mart to facilitate analyses on pig market trends and prices in order to predict future pricing. This case study uses all 13 data sources outlined in Table 3.1 to construct a data mart to be used to predict the price of pigs on the global market. It contains the number of pig slaughters and prices per date and location. The dataset `imf` is required in order to resolve different currencies. From the classifications of StarGraphs presented in Table 4.1 13 data sources specified by the end user, 8 sources were classified as `Full` while the remaining 5 were classified as `Partial`.

7.1.1 User Defined Integration.

The manual data mart was created by editing the mapping files of each generated StarGraph to influence the integration process. Four measures were identified: the number of pigs slaughtered `slaughter`, the price of pigs `price`, the milk futures quotes `milk-future` and the corn futures `corn-future` quotes. In addition there were three main dimensions, `Date`, `Geo` and `Currency`.

The dimensions `Date` and `Geo` are dimensional hierarchies containing various levels of granularity. For the `Date` dimension, the data sources provided were either monthly or weekly. For the `Geo` dimension, the hierarchies were based on area. For example, the `USDA` source provided a breakdown of slaughtering by state, while the `Bord Bia` source lists the number of slaughters as a whole.

Figure 7.1 details the composition of this data mart in a relational format as it would be displayed to a user. Internally, this structure is a Constellation but in order to convey the structure of this data mart, it is displayed as a traditional entity relationship diagram. When constructing this data mart, all prices have been converted to Euro and stored in the `price_eur` column, this has been provided by the `imf` dataset, which lists exchange rates.

A high-level overview of the Constellation can be seen in Fig 7.1, with details of dimensions and facts and the links between them. Solid nodes indicate a dimension while dotted nodes indicate a fact. Solid edges detail the links between dimensions and facts, while dotted edges indicate hierarchical relationships within a dimension.

7.1.2 Non-Assisted Integration

The integration process is outlined in Table 7.1 and contains all the three approaches. `Source_1` and `Source_2` indicate the sources used at different stages of the integration process and for `Source_1`, the value `Cons` represents the current Constellation. `Issues` highlights problems which appeared during the non-assisted integration and `OntologyAssist` indicates the rule or process utilised by the ontology to resolve the issue. The `User Supplied` column indicates the metamodel values a user was

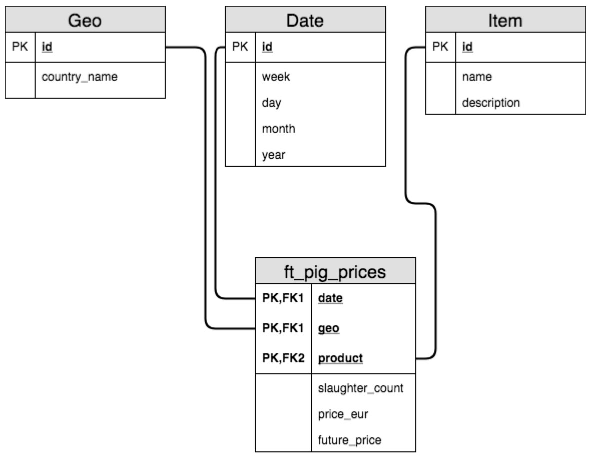


Figure 7.1: User defined Strategy (Case Study 1)

required to supply in order for the integration process to complete. For example, `GEO` indicates that the user was prompted to provide a value for the GEO attribute, while `NONE` indicates that no user intervention was required.

The initial Constellation (Step 1) was created from the datasets `aimis_1` and `aimis_2`. There was a high degree of integration at Step 1. This is because structurally `aim_1` and `aim_2` were very similar. However, granularity for integrated data proved a problem. Both of these sources contained a measure called `%` which denotes the percentage change between two dates. However, for one source the percentage meant the percentage change from the previous *week*, and for the second source, it meant the percentage change for a *year*. Despite these both being correctly identified as measures, the ontology is needed to resolve these issues of granularity.

Step 2 integrated the `b_1` data. Here, there was a single node to be integrated with the previous data, based on the `date` dimension. However, as there was no matching node in the Constellation for the measure, it was not combined with an existing one and instead occupies its own column in the fact table. This measure should have integrated with the previous two sources in addition to the Date dimension as they all relate to the sales and production of pigs. However, without a form of abstraction (supplied by an ontology) to semantically link the two measures they remain separate. Step 3 included the StarGraph created from `b_2` into the Constellation. Similar to `b_1`, this source was integrated based on the `date` dimension as no suitable candidate was found for measure integration. In other words, the dimensional hierarchy was enriched but no new facts were added.

Step 4 integrated `p_1` into the Constellation. Once again, a matching `date` candidate was found which saw a large reduction in the graph (as this source is largely time-series data). However, other dimensions which failed to integrate were country identifiers (e.g. "Austria"). Again, an extension to the ontology to indicate a type hierarchy would see a large degree of integration produced (and subsequently a lower number of nodes & edges). As this data was initially modelled as a matrix, a large number of rows were produced in the fact table associated with the measure which could not be integrated with the existing Constellation. Steps 5 & 6 integrates `p_2` and `p_3`. These sources were simple tables matching years to a measure. As such the data was integrated based on the date dimension and the measure was added to the fact table.

Step 7 integrated `bp_1` with the Graph partially integrated on the countries listed in `p_1` as one dimension specified a country. As expected, without the ontology, which contains a full dimensional hierarchy, some countries failed to be integrated due to differing tags (e.g."Great Britain" and "Northern Ireland" combined would be synonymous with the tag "United Kingdom"). Step 8 integrated the `bp_2` data source. This data source was identical in structure to `bp_1`. As such, there was a 1-1 integration between this source and the Constellation with all measures additionally being merged with those provided by `bp_1`.

Step 9 integrated `c_1` with date providing the only common attribute for integration. This is due to the fact that the data source `c_1` refers to future prices. However, a large number of measures were found within this data source, and as such have been added to the fact data joined on the date dimension. Step 10 integrated `c_2` and, similar to Step 9, the structural similarity between `c_1` and `c_2` facilitated a 1-1 mapping between all nodes. Step 11 sought to integrate currency conversion data from the `imf` data source. The data was successfully integrated on the date dimension, while the new currencies occupied new dimensions and the rates were included as measures within the fact table. Finally, Step 12 integrated the `usda` data, once again using the date dimension. The data was highly dimensional, adding in

30 previously unseen dimensions and 8 measures.

This process generated a large data mart containing 66 dimensions and 7 fact tables and thus, a final illustration cannot be displayed. There was a large degree of redundancy in this data mart as most items failed to integrate due to lacking semantics (e.g. country names, similar products).

7.1.3 Ontology-Assisted Integration

In this section, we describe how a close-to-automatic process for data mart constructed was achieved. However, at various points in the process, it was necessary for the user to update the ontology (through a system prompt) so that integration could complete and to ensure a fully automated integration for the same sources in future data marts.

For the initial Constellation, both a `Date` and `Geo` dimension were found for both data sources. Two measures were found for each source, but there was no defined `Item`, `Metric` or `Units` attributes. As both of these sources were the same structurally and semantically, they were fully integrated. However, the process prompted the user for input in both cases.

The next source `b_1`, was very sparse, containing only two attributes: date and measure. In this instance, the integration processes stopped again to prompt a user for input for the dimensions `Geo` and the attributes `Item`, `Metric` and `Units`, as the ontology again could not provide the precise level of detail. Once supplied, integration was completed using the Date and Geo dimensions, and along the measure by the item dimension. This is because both measures are of the product `Pig` but have different metric and unit attributes.

The next source - `b_2` - was again missing a `Geo` dimension and `Item`, `Metric` and `Units` attributes. Once provided, the system proceeded to integrate this source on the Date and Geo dimensions, and integrated the measure with the `aimis_1` and `aimis_2` sources, as they were identical. The source `p_1` found `Date` and `Geo` dimensions, but failed to find the attributes `Item`, `Metric`. However a `Units` attribute was found (kg per head). The source `p_2` found a `Date` and `Units` attributes, but failed to find a `Geo` and `Item` attributes. Once again, a user prompt updated the

Table 7.1: Non-Assisted Integration Issues for Case Study 1

Step	Issues	OntologyAssist	User Supplied
1	GRAIN_MISMATCH	GRAIN_CHECK	ITEM, METRIC, UNIT
2	MISSING_ATTR	METAMODEL_CHECK	GEO, ITEM, METRIC, UNIT
3	MISSING_ATTR	METAMODEL_CHECK	GEO, ITEM, METRIC, UNIT
4	TERM-TYPE_MISMATCH	TERM-TYPE_MAP	ITEM, METRIC
5	MISSING_ATTR	METAMODEL_CHECK	GEO, ITEM
6	MISSING_ATTR	METAMODEL_CHECK	GEO, METRIC
7	TERM-TYPE_MISMATCH	TERM-TYPE_MAP	ITEM, UNIT
8	TERM-TYPE_MISMATCH	TERM-TYPE_MAP	ITEM, UNIT
9	MISSING_ATTR	METAMODEL_CHECK	GEO, UNIT
10	MISSING_ATTR	METAMODEL_CHECK	GEO, UNIT
11	TERM-TYPE_MISMATCH	TERM-TYPE_MAP	METRIC, UNIT
12	TERM-TYPE_MISMATCH	TERM-TYPE_MAP	NONE

ontology and integration was completed.

The source `p_3` found a `Date` attribute and two `Unit` attributes. However, there was no `Metric` or `Geo` dimension. The source `bp_1` found all required attributes except an `Item` and a `Unit` attribute. Once provided by a user, it was integrated into an existing `Metric` and `Unit` dimension.

The source `bp_2` was also missing the `Item` and `Unit` attributes and, after user prompting, integration with `bp_1` was successful.

The source `c_1` provided several new `Metric` attributes. However, a `Unit` was not listed and the `Geo` dimension was not found. `c_2` was also missing `Unit` attributes and a `Geo` dimension. However, with a user prompt and ontology update, the integration was completed. The final integration was different as the data source provided quotes about corn, and every `Item` thus far referred to pigs. Thus, the system integrated on the `Date` and `Geo` dimensions, which was correct.

The source `imf` found a series of `Item` attributes and a `Date` attribute. However, it failed to find a `Metric` or `Unit` attribute for each measure. Once again, this data was of an entirely new domain, currency conversion rates, and as such was integrated on the `Date` and `Geo` dimensions, after the ontology was updated. The final source `usda` found all required attributes and was integrated automatically.

Rendered as a view, this Constellation can be seen in Figure 7.2. This Constellation contained seven dimensions and four facts. In this data mart, most dimensions contain only two attributes: *name* and *key*. For example, the `Units` dimension contains one attribute `name` which stores units of measurement for a measure (e.g. `kg`).

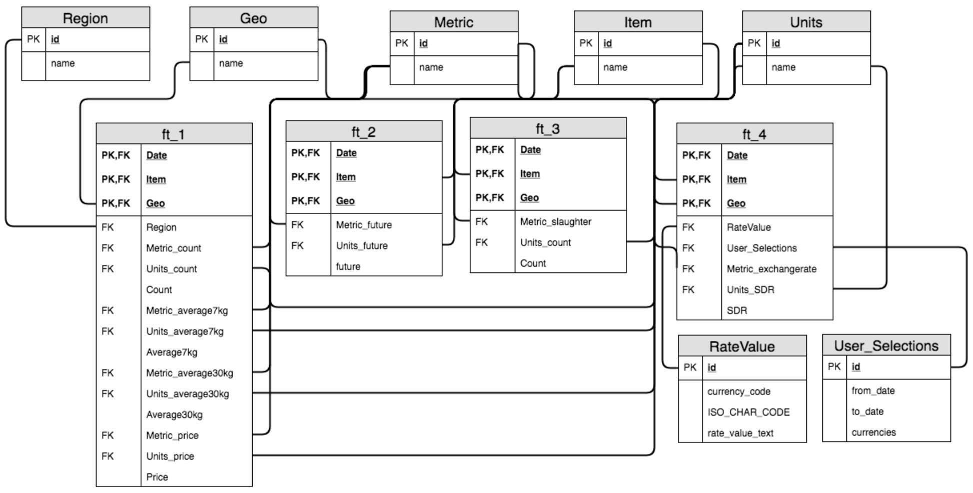


Figure 7.2: Case Study 1 Final Schema: A User View

7.1.4 Summary

Case study 1 required the integration of 13 data sources to construct the data mart. In terms of the user-defined approach, not only do they have to have an in depth understanding of the data they are capturing, it was also necessary to manually design and create the Data Mart (e.g. construct the schema, add constraints, write queries to insert data). This raised the same issues as were seen during both non-assisted and ontology-assisted integration strategies, in terms of having to determine the correct grain in hierarchy dimensions and in detecting the correct attribute for integration. Manual integration was estimated to take between 8 and 10 hours for what was a reasonably complex data mart (13 separate sources), while the automated approach required 118ms and populated with a batch update in 1.1 seconds. The manual data mart had a faster population time of 0.9 seconds.

It is worth highlighting the reasons as to why manual construction takes such a long time as these are generally the same issues that are resolved during ontology-assisted integration. This process also highlights how the ontology is used to aid integration and how rules are updated when issues with the structure of data sources are uncovered.

Step 1 integrates two sources into a Constellation (`Cons`) and from there, proceeds to integrate more sources into this constellation. At a high level, there are three main issues. The first is `GRAIN_MISMATCH`, which occurs when two data source are of differing levels of granularity. For example, the Date dimension has one source that provides weekly data and the other providing monthly data. Similarly for the Geo dimension, some data sources indicate individual countries while others provide statistics globally. The approach is to create a separate fact for each level of granularity. Later, a ROLLUP operation can be employed to join facts.

The second issue is labelled generically as `MISSING_ATTR`. This indicates that the data source did not contain enough information to correctly (semantically) integrate facts. In short, it means that the data source did not contain all of the attributes specified in the metamodel. The approach in this instance is for the ontology to prompt a user for input on these values. The most common reason for this issue to occur is that the data is sparse in terms of dimensions and attributes. When this occurs, it is impossible to infer these values so the system defers to a user to provide the missing contextual information.

The final issue is labelled as `TERM-TYPE_MISMATCH` and revolves around two problems. The first being different terminologies used across data sources, particularly across the `Geo` dimension (for example 'US' vs 'America'). These issues are resolved by the ontology during the term mapping phase, where both possibilities are resolved into a single canonical term. The second issue arises from a lack of type information in both sources. This issue arises when attempting to resolve different attributes and dimensions which are of the same abstract type. For example one source may have a dimension called 'France' and another 'Australia'. Without an ontology linking these two concepts under the common theme 'Country', they cannot be integrated. For this case study, term and type mapping resolved integration problems for the Geo and product dimensions, while the metamodel was used to enforce semantic integration by prompting a user for the `metric` and `units` attributes.

For the remaining two case studies, we will provide a more abbreviated discussion as the issues are identical across case studies. However, it is important to demonstrate the generic nature of our work and the wider applicability.

7.2 Case Study 2: Price Comparison Data Mart

This case study requires the construction of a data mart to allow the analyst to compare the trend in the price of butter with the price of vegetable oil. It requires the 9 sources shown in Table 3.2 to construct the appropriate data mart. Of the 9 sources, 8 were classified as `Full` StarGraphs and `GlobalDairyTrade` being classified as `Partial`. Data sources such as `PPOIL_USD` provide historical data up to the current week.

7.2.1 User Defined Integration

All of the data sources provided for this case study provide the same information, a product and a price at datetime t. However, some attributes such as product name are taken from the name of the source (e.g. `PPOIL_USD` refers to the product Palm Oil in USD). Figure 7.3 details the structure of this Constellation when presented as a view to a user. The Constellation consists of five dimensions and a single fact table containing the measures, the value (price) of a product, and the percentage change of the price compares to the previous week.

7.2.2 Non-Assisted Integration

The sources used in this case study are very similar with most containing only two items: a date and measure called *Value*. Figure 7.4 details the structure of this Constellation when rendered as a view to a user. There are ten dimensions and a single fact in the final data mart. Nine of the dimensions found are product names obtained from the `gdt` source. However, the user defined approach contains a single dimension called `Item` which refers to the product name. Additionally, the two measures `RenCas change` and `%` both refer to the percentage change from the previous week and failed to integrate due to their different names. These are two

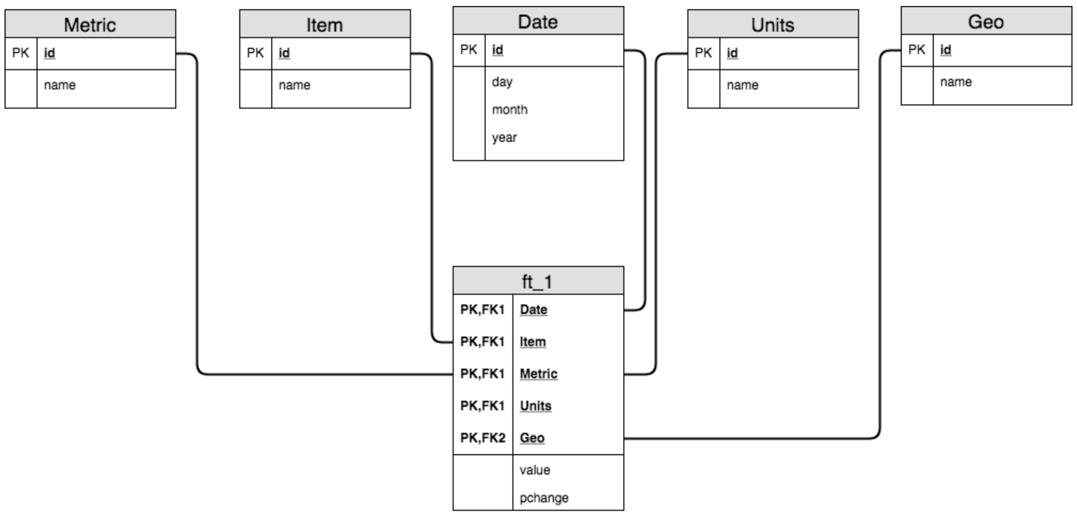


Figure 7.3: User defined Constellation for Case Study 2

significant issues which motivated the need for some form of ontology.

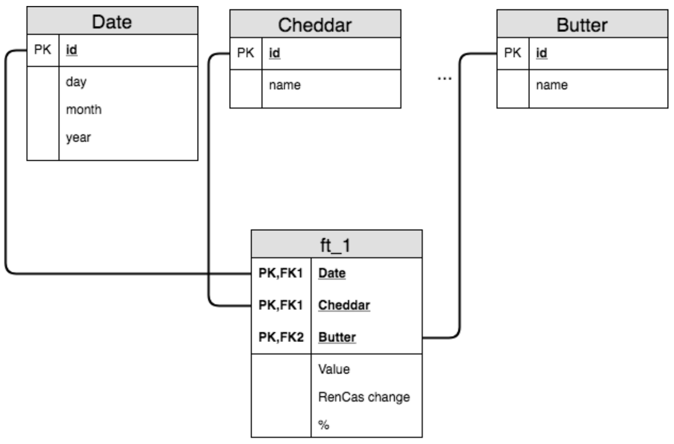


Figure 7.4: Non assisted approach for Case Study 2

7.2.3 Ontology-assisted integration.

Term mapping and type mapping were correctly able to identify all attributes necessary for the GTD data source. However, due to the sparse nature of the CSV data sources, user input was required to determine the Type, Metric and Units for these data sources. Once provided, all data sources shared a date dimension and all CSV sources shared the same metric and units dimensions.

Table 7.2: Integration Issues for Case Study 2

Step	Issues	OntologyAssist	User Supplied
1	TERM-TYPE_MISMATCH, MISSING_ATTR	TERM-TYPE_MAP, METAMODEL_CHECK	GEO, METRIC, ITEM, UNITS
2	TERM-TYPE_MISMATCH, MISSING_ATTR	TERM-TYPE_MAP, METAMODEL_CHECK	GEO, METRIC, ITEM, UNITS
3	TERM-TYPE_MISMATCH, MISSING_ATTR	TERM-TYPE_MAP, METAMODEL_CHECK	GEO, METRIC, ITEM, UNITS
4	MISSING_ATTR	METAMODEL_CHECK	GEO, METRIC, ITEM, UNITS
5	TERM-TYPE_MISMATCH, MISSING_ATTR, GRAIN_MISMATCH	TERM-TYPE_MAP, METAMODEL_CHECK, GRAIN_CHECK	GEO, METRIC, ITEM, UNITS
6	TERM-TYPE_MISMATCH, MISSING_ATTR	TERM-TYPE_MAP, METAMODEL_CHECK	GEO, METRIC, ITEM, UNITS
7	TERM-TYPE_MISMATCH, MISSING_ATTR	TERM-TYPE_MAP, METAMODEL_CHECK	GEO, METRIC, ITEM, UNITS
8	TERM-TYPE_MISMATCH	TERM-TYPE_MAP	GEO, METRIC, UNITS

7.2.4 Comparison

Table 7.2 shows: the issues found in the non-assisted integration; the ontology rules applied to mitigate these issues for the ontology-assisted integration; and a marker denoting whether user intervention was needed at a particular integration step. Most of the sources for this data mart were structurally identical, containing only a Date and a measure. The non-assisted integration approach integrated on both attributes. This was the incorrect approach, even though these sources are structurally identical, they are semantically different.

What was different in this case study, and was due to the sparse nature of sources, was the lack of information required to deliver proper (semantic) integration. These issues are overcome through a combination of the ontology (`TERM-TYPE_MAP`) and user intervention through the ontology's `METAMODEL_CHECK` function. The user supplies the necessary metamodel values for most of these sources and in general, these contained only a date and measure.

This approach produced a structure identical to the user defined approach. However, this is due to the large number of missing metamodel attributes within the data sources which required user intervention to supply these missing values.

Summary. Case study 2 used nine data sources in the construction of the data mart. The user defined approach and the ontology-assisted approach suffered from the same problem: the sparseness of the majority of the data sets. However, as most of the datasets were identical in structure, there was a significant saving in development time when manually constructing the data mart. We estimate that this manual approach completed in about 5 hours and took 17 seconds to perform a population. A large amount of time was spent understanding the data due to its sparse nature. Conversely, the automatic approach had a time of 102ms and populated the data mart in 23 seconds.

7.3 Case Study 3: Milk Production Data Mart

This case study required a data mart to examine year on year changes for milk production and deliveries and required the 5 data sources outlined in Table 3.3. Two were classified as `Full` and three as `Partial` StarGraphs. For this case study, we again used a summary table (Table 7.3) to provide a brief overview of the different integration strategies and the issues that arose.

7.3.1 User Defined and Non-Assisted Integration.

Similar to the first case study, this data mart requires use of information which is not visible to the StarGraph. For instance, terms with names such as `Kg/$` imply that this attribute is a measure; it is created from two units. Additionally, the knowledge a designer has such as `New Zealand, Germany == Country` allow the schema designer to create generic dimensions such as Type, Country and Units. The reason for the units dimension is that unless one explicitly identifies those units used for a specific measure, they cannot be directly compared.

Due to that fact that some sources were csv files with two attributes `date` and `value`, this provided a direct mapping for integration. One other source also provided a date dimension and as such the attributes named `butter`, for example, integrated on this dimension. However, for the remaining source, the only attribute to integrate

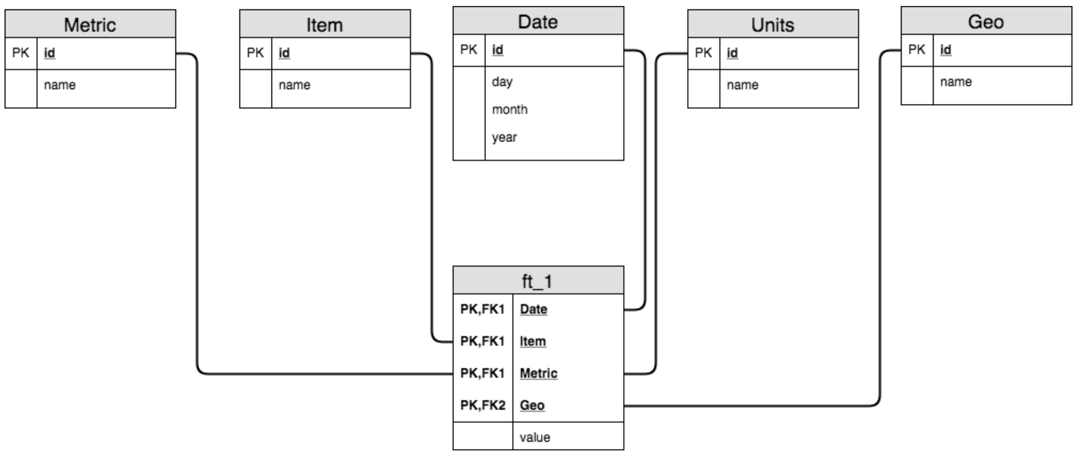


Figure 7.5: User defined Strategy (Case Study 3)

on was the name of the measure. Despite the fact that it provided usable facts, the data content was incorrect in data mart usage. This required an entry in the ontology to *prevent* this aspect to the integration.
The differences between user-defined (Fig. 7.5) and non-assisted (Fig. 7.6) are primarily due to abstractions of which the analyst in the user-defined approach had knowledge. In general, all facts present in the automatic approach are combined into a single fact entity. This is accomplished through the use of a `Type` dimension. However, without a suitable ontology to inform the automatic approach that these facts can be combined, they will remain separated.

7.3.2 Ontology assisted integration

All attributes required for an integration approach were found within both csv files. However, there was no `geo` dimension found for the Argentina and NZ milk production tables. Finally, for the Milk production in Germany data, no `date` or `geo` dimensions were located. This required the user prompt for dimensions and an ontology update before the integration process could complete. This produced a single fact table with five dimensions, one for each metamodel value, similar to the user defined approach 7.5.

Comparison. While case study 3 constructed a data mart from just 5 sources, the levels of missing metadata and heterogeneity in the data, resulted in the most

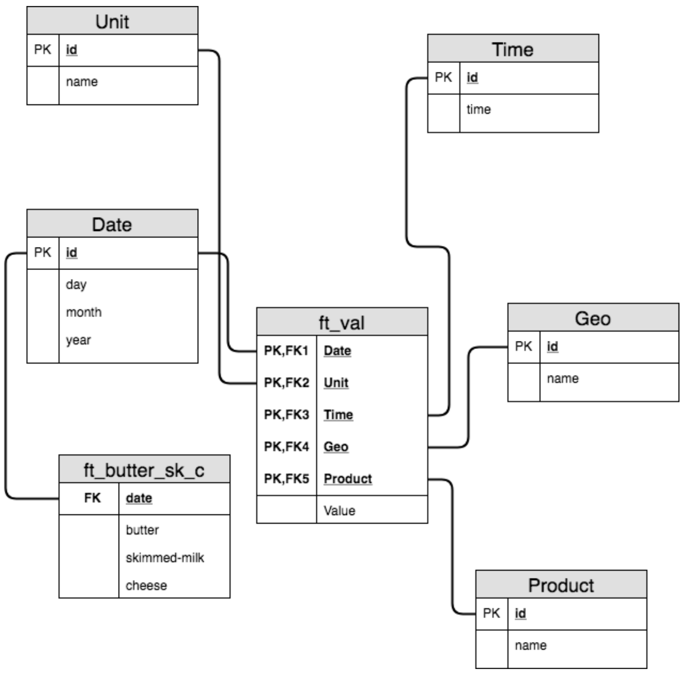


Figure 7.6: Non-assisted Strategy (Case Study 3)

Table 7.3: Integration Issues for Case Study 3

Step	Join on	Issues	OntologyAssist	User Supplied
1	DATE	TERM-TYPE_MISMATCH	TERM-TYPE_MAP	GEO, ITEM
2	DATE	TERM-TYPE_MISMATCH	TERM-TYPE_MAP	NONE
3	DATE	TERM-TYPE_MISMATCH	TERM-TYPE_MAP	GEO
4	DATE	TERM-TYPE_MISMATCH	TERM-TYPE_MAP	GEO, ITEM

difficult integration effort of all three case studies. Regarding the user-defined approach, in addition to the time taken to understand the data and design/implement a data mart, additional time was spent individually examining the markup specific to each HTML source so that the correct data could be extracted. We estimate this process took in the region of 10 hours while the automatic approach completed in approx. 110ms.

With regards to population time, the automatic approach was 11 seconds slower than the manual approach (74s to 63s). Table 7.3 outlines the issues found during the non-assisted approach, the ontology rule applied to overcome this issue, and whether or not user intervention was required at an integration step. For all sources the only issues found were those of term and type mapping. Once again, the ontology

Table 7.4: Analysis of all case study approaches

Name	Source	Meta	Ins	Time	Struct	Sem	Dims	Facts
cs_1_user _defined	13	24	262	8hrs	✓	✓	3	1
cs_1_non _assisted	13	143	262	N/A	✓	✗	66	7
cs_1_ontology _assisted	13	24	262	11m	✓	✓	7	4
cs_2_user _defined	9	16	4016	5hrs	✓	✓	5	1
cs_2_non _assisted	9	28	4016	N/A	✓	✓	10	1
cs_2_ontology _assisted	9	16	4016	8m	✓	✓	5	1
cs_3_user _defined	5	17	19709	10hrs	✓	✓	5	1
cs_3_non _assisted	5	60	19709	N/A	✗	✗	5	2
cs_3_ontology _assisted	5	17	19709	3m	✓	✓	5	1

uses these to assign canonical terms to each data source, and provides a layer of abstraction between the data sources so that they can be semantically linked.

7.4 Overall Summary

The goal of our evaluation was to examine the differences between a manual integration approach and our automated approach, both to determine the value of our methodology and to identify potential improvements to our process. Table 7.4 allows us to compare the results of all three use cases under both a manual and automatic approach. Column `Name` relates to the case study and the approach used: the columns `Source` and `Ins` refer to the number of data sources involved in the integration, and the number of instances for each; `Meta` refers to the number of attributes found in the multidimensional schema once integration has been completed; `Time` is the time taken to construct the final data mart; `Struct` (Structure) relates to the usability of the data (yes/no); `Sem` (Semantics) refers to the correctness of data (yes/no); and finally `Dim` refers to the number of dimensions within the constructed data mart, while `Facts` refers to the number of facts within the mart.

The `Time` column illustrates the savings in time using our approach. The 3 manual approaches required between 5 and 10 hours approximately while the automated approach was between 3 and 11 minutes. In the earlier discussion on case studies, the automated time was reported as between 110ms and 7s. However, here we include 1 minute for each user prompt and response (60 seconds was the longest time recorded). This demonstrates a clear benefit of the StarGraph approach.

In terms of Metadata, the manual approach detected and removed more redundancies than the automatically generated approach. This is due in a large part to the domain expert's knowledge of the dataset compared to the automatic approach. While the semi-automatic approach had similar results to the automatic approach, this is undoubtedly due to the semi-automatic approach requesting information from the user as needed. This results in marginally slower times for batch updates as recorded in the case study discussions. The degree of redundant data determines the difference in data loading times and is our current area of research in terms of improving the ontology.

When examining the dimensions and facts constructed for each data mart, the manual approach provides the best schema for large datasets. With case study 1 containing three dimensions compared to the 7 used for the ontology assisted approach. However, in this instance, only two operations are required in order to exhibit the same functionality as the user defined approach; the removal of redundancies, and a ROLLUP operation to consolidate the two facts into the same date dimension.

Case studies 2 and 3 contain identical marts for both their user defined and non assisted approaches. For case study 2, this is because of the large number of missing metamodel values which required user intervention. For the user defined approach, these values are also required, resulting in an identical data mart. For case study 3, the same result is obtained due to the minimal amount of metamodel values supplied namely `Geo` and `Item`. In this instance, as the majority of metamodel values are found within each data source, a similar data mart is constructed from both the user defined and ontology-assisted approaches.

In all cases, the non-assisted approach has the worst performance, with the worst instance being for case study 1 where 66 dimensions and 7 facts are constructed while a user can represent the same information through three dimensions and a single fact. This shows that some form of ontology coupled with user intervention is required in order to construct data marts as the ontology assisted approaches for sparse data are identical to the user defined methods.

While the user-defined approach is guaranteed to produce the best schema, it is important to note the `Time` taken to construct each data mart. The time taken to construct a mart manually varied from 5-10 hours, depending on the number of sources and their structure. While the time taken to construct a mart using the ontology-assisted approach lies within 3-11 minutes.

Chapter 8

Conclusions

In this final chapter, we will provide a brief summary of the work presented in this dissertation in section 8.1 before discussing possibilities for extending this research in section 8.2.

8.1 Thesis Overview

This dissertation began with discussing the importance of data mining and machine learning in an environment where very large volumes of data are created daily. However, activities such as data mining cannot be performed without a layer of data management to clean, integrate, process and make available the necessary datasets. To that extent, large and costly data flow processes such as Extract-Transform-Load are necessary to extract data from disparate information sources, and generate ready-for-analyses datasets. Furthermore, the traditional approach to this type of processing is not sufficiently robust in the face of streaming data or data sources that can change format in an uncontrolled fashion. In chapter 1, we motivated the hypothesis and goals of this research: a new method for data management is needed if we are to *warehouse* data streams. As our work encompasses multiple fields, the discussion on related research in chapter 2 included methodologies for determining facts, dimensions and measures from unseen data; means in which a data cube and data mart can be represented outside of an RDBMS; ETL systems which seek to combine traditional relational data with web data; and finally, systems which leverage ontologies in order to construct ETL systems for data marts.

Chapter 3 was used to outline our system architecture: the processes and data stores required to provide a means of constructing integrated web data marts. This chapter also provided details of the data sources and case studies which served as running examples throughout chapter 4 and chapter 5. Chapter 4 presented the StarGraph data model and the methodology used to construct a data mart from an unseen data source. In addition, this chapter demonstrates how a StarGraph can be automatically populated from a Data Lake, and examined StarGraphs constructed from 84 previously unseen data sources, some of which constitute the data sources presented by the case studies in chapter 3. A data mart is constructed to meet a set of business requirements and generally involves combining data from multiple sources. Chapter 5 describes the processes involved in integrating two or more StarGraphs constructed from web data to produce a Constellation. Through the use of a lightweight ontology, StarGraphs were integrated to produce semantically correct data marts. In addition, this chapter details how data marts are populated and examines the structure of the Constellations produced by the data sources presented within each case study. Chapter 6 addressed the issue of redundancies found within the data sources which slowed down materialisation times. These redundancies, while not interfering with the *usability* of a data mart constructed from a StarGraph or Constellation provide opportunities to reduce the time taken to populate data sources. This chapter described the properties of each redundancy type and provided a mechanism to detect and remove the redundancy prior to population.

Chapter 7 described our evaluation, which sought to validate the effectiveness of our automatically constructed data marts. We compared their structure, population times and semantics to data marts constructed by designers and domain experts using the same sources. While the evaluation showed that the ETL processes constructed by users have more efficient population times, due to differences in structure, the StarGraph data marts were verified by collaborators from the agri domain and thus, *usable* for the requirements for which they were specified. As part of the evaluation, we were provided with 120 unseen data sources of which

84 were directly usable as data mart components, using the StarGraph model and transformation methodology.
In summary, the main contribution of this research was the specification and deployment of the StarGraph model to automatically construct data marts from previously unseen data sources. Using a semi-automatic process requiring a lightweight ontology and minimal user intervention, we demonstrated how two (or more) StarGraphs can be integrated semantically to produce an integrated data mart.

8.2 Future Work

In this final section of the dissertation, we will explore possibilities for future work arising from this research.

8.2.1 Robust Query Mechanisms

This research focused on the automated construction of data marts from web data sources. As such, the constructed data marts produce the *all cube (*)* query format, meaning a full materialisation of the data mart. In many cases, the entire data cube is not required by the analyst. By providing mechanisms which enable full OLAP functionality on the data mart, an analyst may pose full OLAP queries to our data marts. While this represents a small extension to our work, this would provide the analyst with a means of filtering the *all* cube (e.g. slice, dice) which are desirable features for an analyst. Such mechanisms for querying graph models has been presented in chapter 2 with [80] presenting a data warehouse model for storing and querying graphs.

8.2.2 On-Demand ETL

Traditional ETL systems employ a rigid approach to batch updating due to the high volumes of data which are transferred to the data warehouse. While we present a more lightweight approach of data marts constructed outside the main warehouse environment, we still require the same batch updating of data marts. ETL on-the-fly differs from traditional ETL, where a data mart is previously constructed and queried by a user. ETL on-the-fly constructs a data mart from data sources only at the point at which the query posed by a user. This mechanism provides the system which the ability to process and deal with partially materialised data marts. It works best when used with a data lake and thus, our current system provides a solid platform for this type of extension.

8.2.3 Prescanning Data Sources

At present, a StarGraph is constructed from a data source, prior to redundancies being found. By providing a process which can examine data *prior* to StarGraph construction, redundancies can be detected and removed from the source data. In addition, machine learning and data mining algorithms may be used on the data lake to scan data sources. These algorithms may further optimise the redundancies presented in chapter 6. For example, the use of predictive algorithms such as decision trees can optimise the population process by identifying duplicate values within the data lake, removing the need to populate multiple values within a data source.

8.2.4 Cloud based ETL

At present our system runs in a local environment. However, as the data lake increases in size through day-to-day use, the time taken to query data from the lake and populate it in a local environment may become unrealistic. By storing the data lake in a distributed environment using technologies such as Hadoop [5] and a suitable streaming platform such as Spark [78], this would provide large scale ETL, using many sources in a distributed environment. As a result, this could overcome the problems encountered by a local ETL system as the number of sources and size of the data lake increases.

8.2.5 A Different Approach to Experiments

In chapter 7 the time taken to design and implement the manual approach for each case study was obtained by the author implementing a manual ETL process. A more different set of experiments to fully test the systems capabilities could be implemented by utilising a larger set of data sources across domains and a number of developers. Each developer would be given the task of constructing a data mart corresponding to a user query from a series of these data sources. While designing and implementing the data mart the developer would be asked to take note of the time taken to:

- Gain an understanding of the data
- Determine a suitable schema for the data mart
- Implement the ETL process to populate the data mart
- The population time from source to warehouse for the data mart.

By then constructing the same data marts using the automated system we can then gather more data about the performance of the system compared to a manual approach.

Bibliography

[1] Transport Infrastructure Ireland [Online]. `https://www.tiitraffic.ie/travel_times/`. (Accessed 10/05/2018).

[2] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[3] Agriculture and Horticulture Development Board. `http://pork.ahdb.org.uk/`. (Accessed 10/05/2018).

[4] AIMIS - Agriculture and Agri-Food Canada. `http://www5.agr.gc.ca/eng/home/`. (Accessed 10/05/2018).

[5] Apache Software Foundation. Hadoop. `https://hadoop.apache.org`. (Accessed 10/05/2018).

[6] Maha Azabou, Kaïs Khrouf, Jamel Feki, Chantal Soulé-Dupuy, and Nathalie Vallès. Diamond multidimensional model and aggregation operators for document OLAP. In *Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on*, pages 363–373. IEEE, 2015.

[7] Liang Bai, Songyang Lao, Weiming Zhang, Gareth JF Jones, and Alan F Smeaton. Video semantic content analysis framework based on ontology combined mpeg-7. In *International Workshop on Adaptive Multimedia Retrieval*, pages 237–250. Springer, 2007.

[8] Carlo Batini, Maurizio Lenzerini, and Shamkant B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.

[9] Sonia Bergamaschi, Francesco Guerra, Mirko Orsini, Claudio Sartori, and Maurizio Vincini. A semantic approach to ETL technologies. *Data & Knowledge Engineering*, 70(8):717–731, 2011.

[10] Alain Berro, Imen Megdiche-Bousarsar, and Olivier Teste. Graph-based ETL processes for warehousing statistical open data. 2015.

[11] Kevin Beyer and Raghu Ramakrishnan. Bottom-up computation of sparse and iceberg cube. In *ACM Sigmod Record*, volume 28, pages 359–370. ACM, 1999.

[12] Bord Bia. `http://www.bordbia.ie/Pages/Default.aspx`. (Accessed 10/05/2018).

[13] Omar Boussaid, Riadh Ben Messaoud, Rémy Choquet, and Stéphane Anthoard. X-warehousing: an XML-based approach for warehousing complex data. In *ADBIS*, volume 6, pages 39–54. Springer, 2006.

[14] T. Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159, RFC Editor, March 2014. `http://www.rfc-editor.org/rfc/rfc7159.txt` (Accessed 10/05/2018).

[15] Tim Bray, Michael Sperberg-McQueen, and Jean Paoli. XML 1.0 recommendation. W3C recommendation, W3C, February 1998. `http://www.w3.org/TR/1998/REC-xml-19980210` (Accessed 10/05/2018).

[16] Donald Burleson. *New Developments In Oracle Data Warehousing*, 2004. Available Online: `http://www.dba-oracle.com/oracle_news/2004_4_22_burleson.htm` (Accessed 10/05/2018).

[17] Chris Campbell. Top five differences between data lakes and data warehouses. January 2015. Available Online:

https://www.blue-granite.com/blog/bid/402596/
top-five-differences-between-data-lakes-and-data-warehouses
(Accessed 10/05/2018).

[18] Don Chamberlin, Jerome Simeon, Michael Kay, Scott Boag, Jonathan Robie, Anders Berglund, and Mary Fernandez. XML path language (XPath) 2.0 (second edition). W3C recommendation, W3C, December 2010. http://www.w3.org/TR/2010/REC-xpath20-20101214/ (Accessed 10/05/2018).

[19] Clal. https://www.clal.it/en/index.php. (Accessed 10/05/2018).

[20] CME Group. https://www.cmegroup.com/. (Accessed 10/05/2018).

[21] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[22] Edgar F Codd, Sharon B Codd, and Clynch T Salley. Providing olap (on-line analytical processing) to user-analysts: An IT mandate. *Codd and Date*, 32, 1993.

[23] Dairy World. http://www.milk.de/. (Accessed 10/05/2018).

[24] Marc Demarest. The politics of data warehousing. http://www.hevanet.com/demarest/marc/dwpol.html, 1997. (Accessed 12/03/2018).

[25] ECMA. *ECMA-404–The JSON Data Interchange Syntax*. ECMA (European Association for Standardizing Information and Communication Systems), Geneva, Switzerland, Oct 2013.

[26] Mohamed Medhat Gaber. Advances in data stream mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):79–85, 2012.

[27] GlobalDairyTrade. https://www.globaldairytrade.info/. (Accessed 10/05/2018).

[28] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery*, 1(1):29–53, 1997.

[29] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques.* Elsevier, 2011.

[30] John A Hartigan and JA Hartigan. *Clustering algorithms*, volume 209. Wiley New York, 1975.

[31] Wilhelm Hasselbring. Top-Down vs. Bottom-Up Engineering of Federated Information Systems. In *Engineering Federated Information Systems, Proceedings of the 2nd Workshop EFIS'99, Kühlungsborn, Germany, May 5-7, 1999*, pages 131–138, 1999.

[32] Wilhelm Hasselbring. Information system integration. *Communications of the ACM*, 43(6):32–38, 2000.

[33] Wilhelm Hasselbring. Web data integration for e-commerce applications. *Ieee Multimedia*, 9(1):16–25, 2002.

[34] Victoria J. Hodge and Jim Austin. A survey of outlier detection methodologies. *Artif. Intell. Rev.*, 22(2):85–126, 2004.

[35] Wolfgang Hümmer, Andreas Bauer, and Gunnar Harde. Xcube: Xml for data warehouses. In *Proceedings of the 6th ACM International Workshop on Data Warehousing and OLAP*, DOLAP '03, pages 33–40, New York, NY, USA, 2003. ACM.

[36] Bill Inmon. The data warehouse budget. *DM Review Magazine, January*, 1997.

[37] William H. Inmon. *Building the Data Warehouse.* John Wiley & Sons, Inc., New York, NY, USA, 1992.

[38] William H Inmon. *Building the data warehouse.* John wiley & sons, 2005.

[39] International Monetary Fund.[Online].
`http://www.imf.org/external/index.htm`. (Accessed 10/05/2018).

[40] Mikael R Jensen, Thomas H Møller, and Torben Bach Pedersen. Specifying olap cubes on xml data. In *Scientific and Statistical Database Management, 2001. SSDBM 2001. Proceedings. Thirteenth International Conference on*, pages 101–112. IEEE, 2001.

[41] Alexandros Karakasidis, Panos Vassiliadis, and Evaggelia Pitoura. ETL queues for active data warehousing. In *Proceedings of the 2nd international workshop on Information quality in information systems*, pages 28–39. ACM, 2005.

[42] Ralph Kimball and Margy Ross. The data warehouse toolkit: The complete guide to dimensional modeling. 2002.

[43] Sasivimol Kittivoravitkul and Peter McBrien. Integrating unnormalised semi-structured data sources. In *CAiSE*, pages 460–474. Springer, 2005.

[44] Mark Levene and George Loizou. Why is the snowflake schema a good data warehouse design? *Information Systems*, 28(3):225–240, 2003.

[45] Hans Peter Luhn. A business intelligence system. *IBM Journal of Research and Development*, 2(4):314–319, 1958.

[46] Svetlana Mansmann, Nafees Ur Rehman, Andreas Weiler, and Marc H Scholl. Discovering olap dimensions in semi-structured data. *Information Systems*, 44:120–133, 2014.

[47] Adriana Marotta, Regina Motz, and Raul Ruggia. Managing source schema evolution in web warehouses. *Journal of the Brazilian Computer Society*, 8(2):20–31, 2002.

[48] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[49] DuyHoa Ngo and Zohra Bellahsene. Yam++: A multi-strategy based approach for ontology matching task. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 421–425. Springer, 2012.

[50] Tapio Niemi, Marko Niinimäki, Jyrki Nummenmaa, and Peter Thanisch. Constructing an OLAP cube from distributed XML data. In *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP*, pages 22–27. ACM, 2002.

[51] Marko Niinimäki and Tapio Niemi. An ETL process for OLAP using RDF/OWL ontologies. *J. Data Semantics*, 13:97–119, 2009.

[52] Martin F O'Connor, Kenneth Conroy, Mark Roantree, Alan F Smeaton, and Niall M Moyna. Querying XML data streams from wireless sensor networks: An evaluation of query engines. In *Research Challenges in Information Science, 2009. RCIS 2009. Third International Conference on*, pages 23–30. IEEE, 2009.

[53] Josh Parenteau, Neil Chandler, Rita L. Sallam, Dogulas Laney, and Alan D. Duncan. Predicts 2015: Power shift in business intelligence and analytics will fuel disruption. Technical report, Gartner, November 2014. `https://www.gartner.com/doc/2920717/predicts--power-shift-business` (Accessed 10/05/2018).

[54] Neoklis Polyzotis, Spiros Skiadopoulos, Panos Vassiliadis, Alkis Simitsis, and Nils-Erik Frantzell. Supporting streaming updates in an active data warehouse. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 476–485. IEEE, 2007.

[55] Geneviève Pujolle, Franck Ravat, Olivier Teste, Ronan Tournier, and Gilles Zurfluh. Multidimensional database design from document-centric XML documents. *Data Warehousing and Knowledge Discovery*, pages 51–65, 2011.

[56] Quandl. `https://www.quandl.com/`. (Accessed 10/05/2018).

[57] Franck Ravat, Jiefu Song, and Olivier Teste. Designing multidimensional cubes from warehoused data and linked open data. In *Research Challenges in Information Science (RCIS), 2016 IEEE Tenth International Conference on*, pages 1–12. IEEE, 2016.

[58] Antonio Regalado. The data made me do it. *MIT Technology Review*, May 2013. `https://www.technologyreview.com/s/514346/the-data-made-me-do-it/` (Accessed 10/05/2018).

[59] Oscar Romero, Alkis Simitsis, and Alberto Abelló. Gem: requirement-driven generation of etl and multidimensional conceptual designs. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 80–95. Springer, 2011.

[60] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[61] Michael Scriney, Suzanne McCarthy, Andrew McCarren, Paolo Cappellari, and Mark Roantree. Automating data mart construction from semi-structured data sources. 2018. To appear in the Computer Journal, Oxford University Press, 2018.

[62] Michael Scriney, Martin F. O'Connor, and Mark Roantree. Generating cubes from smart city web data. In *Proceedings of the Australasian Computer Science Week Multiconference, ACSW 2017, Geelong, Australia, January 31 - February 3, 2017*, pages 49:1–49:8, 2017.

[63] Michael Scriney, Martin F. O'Connor, and Mark Roantree. Integrating online data for smart city data marts. In *Data Analytics - 31st British International Conference on Databases, BICOD 2017, London, UK, July 10-12, 2017, Proceedings*, pages 23–35, 2017.

[64] Michael Scriney and Mark Roantree. Efficient cube construction for smart city data. In *Proceedings of the Workshops of the EDBT/ICDT 2016 Joint*

Conference, EDBT/ICDT Workshops 2016, Bordeaux, France, March 15, 2016., 2016.

[65] Khouri Selma, Boukhari IlyèS, Bellatreche Ladjel, Sardet Eric, Jean Stéphane, and Baron Michael. Ontology-based structured web data warehouses for sustainable interoperability: requirement modeling, design methodology and tool. *Computers in Industry*, 63(8):799–812, 2012.

[66] Rick Sherman. *Business intelligence guidebook: From data integration to analytics*. Newnes, 2014.

[67] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

[68] Yannis Sismanis, Antonios Deligiannakis, Nick Roussopoulos, and Yannis Kotidis. Dwarf: Shrinking the petacube. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 464–475. ACM, 2002.

[69] Dimitrios Skoutas and Alkis Simitsis. Designing etl processes using semantic web technologies. In *Proceedings of the 9th ACM international workshop on Data warehousing and OLAP*, pages 67–74. ACM, 2006.

[70] Dimitrios Skoutas and Alkis Simitsis. Ontology-based conceptual design of ETL processes for both structured and semi-structured data. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 3(4):1–24, 2007.

[71] Alan F. Smeaton. The sensor web: Unpredictable, noisy and loaded with errors. In *2010 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2010, Toronto, Canada, August 31 - September 3, 2010, Main Conference Proceedings*, page 3, 2010.

[72] Paul Thomas. Business intelligence and analytics: A new perspective. `https://www.provenir.com/2017/08/rise-business-intelligence-analytics-using/`, 2017. (Accessed 10/05/2018).

[73] Juan Trujillo, Sergio Luján-Mora, and Il-Yeol Song. Applying UML and XML for designing and interchanging information for data warehouses and OLAP applications. *Journal of Database Management (JDM)*, 15(1):41–72, 2004.

[74] USDA. `https://quickstats.nass.usda.gov/`. (Accessed 10/05/2018).

[75] Jeffrey Scott Vitter, Min Wang, and Bala Iyer. Data cube approximation and histograms via wavelets. In *Proceedings of the seventh international conference on Information and knowledge management*, pages 96–104. ACM, 1998.

[76] Yuxiang Wang, Aibo Song, and Junzhou Luo. A mapreducemerge-based data cube construction method. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, pages 1–6. IEEE, 2010.

[77] Barbara H Wixom and Hugh J Watson. An empirical investigation of the factors affecting data warehousing success. *MIS quarterly*, pages 17–41, 2001.

[78] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[79] Zhuolun Zhang and Sufen Wang. A framework model study for ontology-driven etl processes. In *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM'08. 4th International Conference on*, pages 1–4. IEEE, 2008.

[80] Peixiang Zhao, Xiaolei Li, Dong Xin, and Jiawei Han. Graph cube: on warehousing and OLAP multidimensional networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 853–864. ACM, 2011.

Appendices

Appendix A

Schema and Mappings for aim_1 StarGraph

```
{
    name:"aim_1",
    type:"HTML",
    source:"http://aimis-simia.agr.gc.ca/..."
    dimensions:[
        {
          name:"West",
          attributes:[],
          subdimensions:[
            {
                name:"British Columbia - Alberta",
                 attributes:[],
                 subdimensions:[]
            },
            {
                name:"Saskatchewan - Manitoba",
                attributes:[],
                subdimensions:[]
            },
            {
                name:"TOTAL WEST",
                attributes:[],
                subdimensions:[]
            }]
        },
        {
          name:"East",
          attributes:[],
          subdimensions:[
            {
                name:"TOTAL EAST",
                attributes:[],
                subdimensions:[]
            }
          ]
        },
        {
           name:"Canada",
                attributes:[],
                subdimensions:[]
             },
             {
                 name:"Federal/Provincial",
                 attributes:[],
                 subdimensions:[]
             },
             {
                name:"Federal",
                 attributes:[],
                 subdimensions:[]
             },
             {
                name:"Provincial**",
                 attributes:[],
                 subdimensions:[]
             },
             {
                name:"Date",
                attributes:[],
                subdimensions:[]
             }
         ],
         measures:[
             {
                 name:"",
             },
             {
                name:"%"
         ,   }]
         facts:[
             {
                 dimensions:["West","East","CANADA","Federal","Provincial"
                 ,"Federal/Provincial**","Date"],
                 measures:["","%"]
             }]}
("//table[1]/tbody/tr[3]/th[1]") -> West,
("//table[1]/tbody/tr[3]/th[2]")-> West.subdimensions["British Columbia - Alberta"]
("//table[1]/tbody/tr[4]/th[2]")-> West.subdimensions["Saskatchewan - Manitoba"]
("//table[1]/tbody/tr[5]/th[2]")-> West.subdimensions["TOTAL WEST"]
("//table[1]/tbody/tr[6]/th[1]")-> East
("//table[1]/tbody/tr[6]/th[1]")-> East.subdimensions["TOTAL EAST"]
("//table[1]/tbody/tr[7]/th[1]")-> Canada
("//table[1]/tbody/tr[1]/th[1]")-> Federal
("//table[1]/tbody/tr[1]/th[2]")-> Provincial**
("//table[1]/tbody/tr[1]/th[3]")-> Federal/Provincial
("//table[1]/tbody/tr[2]/th[1]")-> Date
("//table[1]/tbody/tr[2]/th[2]")-> Date
("//table[1]/tbody/tr[2]/th[3]")-> %
("//table[1]/tbody/tr[2]/th[4]")-> Date
("//table[1]/tbody/tr[2]/th[5]")-> Date
("//table[1]/tbody/tr[2]/th[6]")-> Date
("//table[1]/tbody/tr[2]/th[7]")-> Date
("//table[1]/tbody/tr[3-7]/td[1,2,4,5,6,7]")->measures[""]
("//table[1]/tbody/tr[3-7]/td[3]")->measures["%"]
```

Appendix B

Schema and Mappings for `imf` StarGraph

```
{
    "name" : "imf",
    "type" : "XML",
     source:"http://www.imf.org/external/...",
    dimensions : [
    {
        "name" : "ExchangeRateReport",
        "type" : "Object",
        "src" : "//ExchangeRateReport",
        "attributes":[{
            "name":"ReportName",
            "type":"String",
            "src":"//ExchangeRateReport/ReportName/text()"
        },
        {
           "name":"Disclaimer",
            "type":"String",
            "src":"//ExchangeRateReport/Disclaimer/text()"
        }],
        "subdimensions":[
           {
            name:"USER SELECTIONS",
            type:"Object",
            src:"//ExchangeRateReport//USER_SELECTIONS",
            attributes:[
                {
                  name:"text()",
                  src:"//ExchangeRateReport//USER_SELECTIONS/text()",
                  type:"String",
                },
                {
                  name:"currencies",
                  src:"//ExchangeRateReport//USER_SELECTIONS/currencies",
                  type:"String"
                }
            ],
            subdimensions:[
                    {
                       name:"DATE_RANGE",
                       type:"object",
                       src:"//ExchangeRateReport//USER_SELECTIONS/date_range",
                       subdimensions:[],
                       attributes:[{
                         name:"from_date",
                         type:"string",
                         src:"//ExchangeRateReport//USER_SELECTIONS/
                         date_range/from_Date"
                       },
                       {
                         name:"to_date",
                         type:"string",
                         src:"//ExchangeRateReport//USER_SELECTIONS
                         /date_range/to_Date"
                       }]
                    }
                ]
               },
               {
                 name:"Rate_value",
                 type:"object",
                 src:"//ExchangeRateReport/EffectiveDate/RateValue[*]",
                 attributes:[
                 {
                    name:"CURRENCY_CODE",
                    type:"string",
                    src:"//ExchangeRateReport/EffectiveDate/RateValue[*]
                    @CURRENCY_CODE"
                 },
                 {
                    name:"ISO_CHAR_CODE",
                    type:"string",
                    src:"//ExchangeRateReport/EffectiveDate/RateValue[*]
                    @ISO_CHAR_CODE"
                 }],
                 subdimensions:[]
               }
             ]
        },..
        ],
        measures : [
        {
            "name" : "RateValue",
            "type" : "MEASURE",
            "src" : "//ExchangeRateReport/EffectiveDate/RateValue[*]/text()"
        },..
        ],
        facts:[{
            dimensions:["ExchangeRateReport","UserSelections,....],
            measures:["RateValue/text()"]
        }]
    }
(//ExchangeRateReport)->ExchangeRateReport
(//ExchangeRateReport/ReportName/text())->ExchangeRateReport[‘‘ReportName"]
(//ExchangeRateReport/Disclaimer/text())->ExchangeRateReport[‘‘Discalimer"]
(//ExchangeRateReport/User_Selections) ->
   ExchangeRateReport.subdimensions[‘‘User_Selections"]
(//ExchangeRateReport/User_Selections/Currencies/text()) ->
   ExchangeRateReport.subdimensions[‘‘User_Selections"].Currencies
```

```
(//ExchangeRateReport/User_Selections/DATE_RANGE) ->
   ExchangeRateReport.subdimensions[‘‘User_Selections"].subdimensions["DATE_RANGE"]
(//ExchangeRateReport/User_Selections/DATE_RANGE/FROM_DATE) ->
   ExchangeRateReport.subdimensions[‘‘User_Selections"]
      .subdimensions["DATE_RANGE"].from_date
(//ExchangeRateReport/User_Selections/DATE_RANGE/TO_DATE) ->
   ExchangeRateReport.subdimensions[‘‘User_Selections"]
      .subdimensions["DATE_RANGE"].to_date
(//ExchangeRateReport/EffectiveDate/RateValue[@CURRENCY_CODE])->
   RateValue.currency_code
(//ExchangeRateReport/EffectiveDate/RateValue[@ISO_CHAR_CODE])->
   RateValue.iso_char_code
(//ExchangeRateReport/EffectiveDate/RateValue/text())->RateValue
```

Appendix C

Schema and Mappings for `tii` StarGraph

```
{
  "name" : "tii",
  "type" : "JSON",
   source:"dataproxy.mtcc.ie/..",
  dimensions : [
  {
      "name" : "M7_EastBound",
      "type" : "Object",
      "src" : "M7_EastBound",
      "attributes":[],
      "subdimensions":[
          {
              name:"",
              "type":"Object",
              "src":"M7_EastBound.data[*]",
              "attributes":[{
                  "name":"from_name",
                  "type":"String",
                  "src":"M7_EastBound.data[*].from_name"
              },
              {
                  "name":"to_name",
                  "type":"String",
                  "src":"M7_EastBound.data[*].to_name"
              },
               {
                  "name":"status",
                  "type":"String",
                  "src":"M7_EastBound.data[*].status"
              }]
          }
      ]
  },
  {
      "name" : "M4_WestBound",
      "type" : "Object",
      "src" : "M4_WestBound",
      "attributes":[],
      "subdimensions":[
          {
              name:"",
              "type":"Object",
              "src":"M4_WestBound.data[*]",
              "attributes":[{
                  "name":"from_name",
                  "type":"String",
                  "src":"M4_WestBound.data[*].from_name"
              },
              {
                  "name":"to_name",
                  "type":"String",
                  "src":"M4_WestBound.data[*].to_name"
              },
               {
                  "name":"status",
                  "type":"String",
                  "src":"M4_WestBound.data[*].status"
              }]
          }
      ]
  },
  {
      "name" : "M3_EastBound",
      "type" : "Object",
      "src" : "M3_EastBound",
      "attributes":[],
      "subdimensions":[
          {
              name:"",
              "type":"Object",
              "src":"M3_EastBound.data[*]",
              "attributes":[{
                  "name":"from_name",
                  "type":"String",
                  "src":"M3_EastBound.data[*].from_name"
              },
              {
                  "name":"to_name",
                  "type":"String",
                  "src":"M3_EastBound.data[*].to_name"
              },
               {
                  "name":"status",
                  "type":"String",
                  "src":"M3_EastBound.data[*].status"
              }]
          }
      ]
  },
   {
      "name" : "M7_WestBound",
      "type" : "Object",
      "src" : "M7_WestBound",
      "attributes":[],
      "subdimensions":[
          {
              name:"",
              "type":"Object",
              "src":"M7_WestBound.data[*]",
              "attributes":[{
                  "name":"from_name",
                  "type":"String",
                      "src":"M7_WestBound.data[*].from_name"
                  },
                  {
                      "name":"to_name",
                      "type":"String",
                      "src":"M7_WestBound.data[*].to_name"
                  },
                   {
                      "name":"status",
                      "type":"String",
                      "src":"M7_WestBound.data[*].status"
                  }]
              }
          ]
      },
      {
          "name" : "M1_NorthBound",
          "type" : "Object",
          "src" : "M1_NorthBound",
          "attributes":[],
          "subdimensions":[
              {
                  name:"",
                  "type":"Object",
                  "src":"M1_NorthBound.data[*]",
                  "attributes":[{
                      "name":"from_name",
                      "type":"String",
                      "src":"M1_NorthBound.data[*].from_name"
                  },
                  {
                      "name":"to_name",
                      "type":"String",
                      "src":"M1_NorthBound.data[*].to_name"
                  },
                   {
                      "name":"status",
                      "type":"String",
                      "src":"M1_NorthBound.data[*].status"
                  }]
              }
          ]
      },
      {
          "name" : "M2_SouthBound",
          "type" : "Object",
          "src" : "M2_SouthBound",
          "attributes":[],
          "subdimensions":[
              {
                  name:"",
                  "type":"Object",
                  "src":"M2_SouthBound.data[*]",
                  "attributes":[{
                      "name":"from_name",
                      "type":"String",
                      "src":"M2_SouthBound.data[*].from_name"
                  },
                  {
                      "name":"to_name",
                      "type":"String",
                      "src":"M2_SouthBound.data[*].to_name"
                  },
                   {
```

```
                "name":"status",
                "type":"String",
                "src":"M2_SouthBound.data[*].status"
            }]
        }
    ]
},
{
    "name" : "M4_EastBound",
    "type" : "Object",
    "src" : "M4_EastBound",
    "attributes":[],
    "subdimensions":[
        {
            name:"",
            "type":"Object",
            "src":"M4_EastBound.data[*]",
            "attributes":[{
                "name":"from_name",
                "type":"String",
                "src":"M4_EastBound.data[*].from_name"
            },
            {
                "name":"to_name",
                "type":"String",
                "src":"M4_EastBound.data[*].to_name"
            },
             {
                "name":"status",
                "type":"String",
                "src":"M4_EastBound.data[*].status"
            }]
        }
    ]
},
{
    "name" : "M1_SouthBound",
    "type" : "Object",
    "src" : "M1_SouthBound",
    "attributes":[],
    "subdimensions":[
        {
            name:"",
            "type":"Object",
            "src":"M1_SouthBound.data[*]",
            "attributes":[{
                "name":"from_name",
                "type":"String",
                "src":"M1_SouthBound.data[*].from_name"
            },
            {
                "name":"to_name",
                "type":"String",
                "src":"M1_SouthBound.data[*].to_name"
            },
             {
                "name":"status",
                "type":"String",
                "src":"M1_SouthBound.data[*].status"
            }]
        }
    ]
},
 {
    "name" : "M50_SouthBound",
    "type" : "Object",
    "src" : "M50_SouthBound",
    "attributes":[],
    "subdimensions":[
        {
            name:"",
            "type":"Object",
            "src":"M50_SouthBound.data[*]",
            "attributes":[{
                "name":"from_name",
                "type":"String",
                "src":"M50_SouthBound.data[*].from_name"
            },
            {
                "name":"to_name",
                "type":"String",
                "src":"M50_SouthBound.data[*].to_name"
            },
             {
                "name":"status",
                "type":"String",
                "src":"M50_SouthBound.data[*].status"
            }]
        }
    ]
},
{
    "name" : "M50_NorthBound",
    "type" : "Object",
    "src" : "M50_NorthBound",
    "attributes":[],
    "subdimensions":[
        {
            name:"",
            "type":"Object",
            "src":"M50_NorthBound.data[*]",
            "attributes":[{
                "name":"from_name",
                "type":"String",
                "src":"M50_NorthBound.data[*].from_name"
            },
            {
                "name":"to_name",
                "type":"String",
                "src":"M50_NorthBound.data[*].to_name"
            },
             {
                "name":"status",
                "type":"String",
                "src":"M50_NorthBound.data[*].status"
            }]
        }
    ]
},
],
measures : [
{
    "name" : "current_travel_time",
    "type" : "MEASURE",
    "src" : "*.data[*].current_travel_time"
},
{
    "name" : "free_flow_travel_time",
                "type" : "MEASURE",
                "src" : "*.data[*].free_flow_travel_time"
            },
            {
                "name" : "distance",
                "type" : "MEASURE",
                "src" : "*.data[*].distance"
            }
            ],
            facts:[{
                dimensions:["M7_EastBound","M4_WestBound","M3_EastBound",
                "M7_WestBound","M1_NorthBound","M2_SouthBound","M4_EastBound",
                "M1_SouthBound","M50_SouthBound","M50_NorthBound"],
                measures:["current_travel_time", "distance","free_flow_travel_time"]
            }]
        }
    (M7_eastBound)->M7_EastBound
    (M7_eastBound.data[*].status)->M7_EastBound.subdimsnions[""].status
    (M7_eastBound.data[*].from_name)->M7_EastBound.subdimsnions[""].from_name
    (M7_eastBound.data[*].to_name)->M7_EastBound.subdimsnions[""].to_name
    (M7_eastBound.data[*].distance)->Distance
    (M7_eastBound.data[*].current_travel_time)->current_travel_time
    (M7_eastBound.data[*].free_flow_travel_time)->free_flow_travel_time
    (M4_westBound)->M4_westBound
    (M4_westBound.data[*].status)->M4_westBound.subdimsnions[""].status
    (M4_westBound.data[*].from_name)->M4_westBound.subdimsnions[""].from_name
    (M4_westBound.data[*].to_name)->M4_westBound.subdimsnions[""].to_name
    (M4_westBound.data[*].distance)->Distance
    (M4_westBound.data[*].current_travel_time)->current_travel_time
    (M4_westBound.data[*].free_flow_travel_time)->free_flow_travel_time
    (M3_eastBound)->M3_eastBound
    (M3_eastBound.data[*].status)->M3_eastBound.subdimsnions[""].status
    (M3_eastBound.data[*].from_name)->M3_eastBound.subdimsnions[""].from_name
    (M3_eastBound.data[*].to_name)->M3_eastBound.subdimsnions[""].to_name
    (M3_eastBound.data[*].distance)->Distance
    (M3_eastBound.data[*].current_travel_time)->current_travel_time
    (M3_eastBound.data[*].free_flow_travel_time)->free_flow_travel_time
    (M7_westBound)->M7_westBound
    (M7_westBound.data[*].status)->M7_westBound.subdimsnions[""].status
    (M7_westBound.data[*].from_name)->M7_westBound.subdimsnions[""].from_name
    (M7_westBound.data[*].to_name)->M7_westBound.subdimsnions[""].to_name
    (M7_westBound.data[*].distance)->Distance
    (M7_westBound.data[*].current_travel_time)->current_travel_time
    (M7_westBound.data[*].free_flow_travel_time)->free_flow_travel_time
    (M1_northBound)->M1_northBound
    (M1_northBound.data[*].status)->M1_northBound.subdimsnions[""].status
    (M1_northBound.data[*].from_name)->M1_northBound.subdimsnions[""].from_name
    (M1_northBound.data[*].to_name)->M1_northBound.subdimsnions[""].to_name
    (M1_northBound.data[*].distance)->Distance
    (M1_northBound.data[*].current_travel_time)->current_travel_time
    (M1_northBound.data[*].free_flow_travel_time)->free_flow_travel_time
    (M2_southBound)->M2_southBound
    (M2_southBound.data[*].status)->M2_southBound.subdimsnions[""].status
    (M2_southBound.data[*].from_name)->M2_southBound.subdimsnions[""].from_name
    (M2_southBound.data[*].to_name)->M2_southBound.subdimsnions[""].to_name
    (M2_southBound.data[*].distance)->Distance
    (M2_southBound.data[*].current_travel_time)->current_travel_time
    (M2_southBound.data[*].free_flow_travel_time)->free_flow_travel_time
    (M4_eastBound)->M4_eastBound
    (M4_eastBound.data[*].status)->M4_eastBound.subdimsnions[""].status
    (M4_eastBound.data[*].from_name)->M4_eastBound.subdimsnions[""].from_name
    (M4_eastBound.data[*].to_name)->M4_eastBound.subdimsnions[""].to_name
    (M4_eastBound.data[*].distance)->Distance
    (M4_eastBound.data[*].current_travel_time)->current_travel_time
```

```
(M4_eastBound.data[*].free_flow_travel_time)->free_flow_travel_time
(M1_southBound)->M1_southBound
(M1_southBound.data[*].status)->M1_southBound.subdimsnions[""].status
(M1_southBound.data[*].from_name)->M1_southBound.subdimsnions[""].from_name
(M1_southBound.data[*].to_name)->M1_southBound.subdimsnions[""].to_name
(M1_southBound.data[*].distance)->Distance
(M1_southBound.data[*].current_travel_time)->current_travel_time
(M1_southBound.data[*].free_flow_travel_time)->free_flow_travel_time
(M50_northBound)->M50_northBound
(M50_northBound.data[*].status)->M50_northBound.subdimsnions[""].status
(M50_northBound.data[*].from_name)->M50_northBound.subdimsnions[""].from_name
(M50_northBound.data[*].to_name)->M50_northBound.subdimsnions[""].to_name
(M50_northBound.data[*].distance)->Distance
(M50_northBound.data[*].current_travel_time)->current_travel_time
(M50_northBound.data[*].free_flow_travel_time)->free_flow_travel_time
(M50_southBound)->M50_southBound
(M50_southBound.data[*].status)->M50_southBound.subdimsnions[""].status
(M50_southBound.data[*].from_name)->M50_southBound.subdimsnions[""].from_name
(M50_southBound.data[*].to_name)->M50_southBound.subdimsnions[""].to_name
(M50_southBound.data[*].distance)->Distance
(M50_southBound.data[*].current_travel_time)->current_travel_time
(M50_southBound.data[*].free_flow_travel_time)->free_flow_travel_time
```

Appendix D

Schema and Mappings for usda StarGraph

```
{
    "_id" : ObjectId("59e397eff0f2229021f27d52"),
    "name" : "download(1).csv",
    "type" : "CSV",
    "source" : "C:Users\\mscri\\Desktop\\exp_sources\\Agri-data\\download(1).csv",
    "dimensions" : [
        {
            "name" : "source_desc",
            "type" : "STRING",
            "src" : "[0]"
        },
        {
            "name" : "sector_desc",
            "type" : "STRING",
            "src" : "[1]"
        },
        {
            "name" : "group_desc",
            "type" : "STRING",
            "src" : "[2]"
        },
        {
            "name" : "commodity_desc",
            "type" : "STRING",
            "src" : "[3]"
        },
        {
            "name" : "class_desc",
            "type" : "STRING",
            "src" : "[4]"
        },
        {
            "name" : "prodn_practice_desc",
            "type" : "STRING",
            "src" : "[5]"
        },
        {
            "name" : "util_practice_desc",
    "type" : "STRING",
    "src" : "[6]"
},
{
    "name" : "stasticcat_desc",
    "type" : "STRING",
    "src" : "[7]"
},
{
    "name" : "unit_desc",
    "type" : "STRING",
    "src" : "[8]"
},
{
    "name" : "short_desc",
    "type" : "STRING",
    "src" : "[9]"
},
{
    "name" : "domain_desc",
    "type" : "STRING",
    "src" : "[10]"
},
{
    "name" : "domaincat_desc",
    "type" : "STRING",
    "src" : "[11]"
},
{
    "name" : "agg_level_desc",
    "type" : "STRING",
    "src" : "[12]"
},
{
    "name" : "state_alpha",
    "type" : "STRING",
    "src" : "[15]"
},
{
    "name" : "state_name",
    "type" : "STRING",
    "src" : "[16]"
},
{
    "name" : "asd_code",
    "type" : "STRING",
    "src" : "[17]"
},
{
    "name" : "asd_desc",
    "type" : "STRING",
    "src" : "[18]"
},
{
    "name" : "county_ansi",
    "type" : "STRING",
    "src" : "[19]"
},
{
    "name" : "county_code",
    "type" : "STRING",
    "src" : "[20]"
},
{
        "name" : "counry_name",
        "type" : "STRING",
        "src" : "[21]"
    },
    {
        "name" : "region_desc",
        "type" : "STRING",
        "src" : "[22]"
    },
    {
        "name" : "zip_5",
        "type" : "STRING",
        "src" : "[23]"
    },
    {
        "name" : "watershed_desc",
        "type" : "STRING",
        "src" : "[25]"
    },
    {
        "name" : "congr_district_code",
        "type" : "STRING",
        "src" : "[26]"
    },
    {
        "name" : "country_name",
        "type" : "STRING",
        "src" : "[28]"
    },
    {
        "name" : "location_desc",
        "type" : "STRING",
        "src" : "[29]"
    },
    {
        "name" : "year",
        "type" : "DATE",
        "src" : "[30]"
    },
    {
        "name" : "freq_desc",
        "type" : "STRING",
        "src" : "[31]"
    },
    {
        "name" : "reference_period_desc",
        "type" : "STRING",
        "src" : "[34]"
    },
    {
        "name" : "week_ending",
        "type" : "STRING",
        "src" : "[35]"
    },
    {
        "name" : "load_time",
        "type" : "DATE",
        "src" : "[36]"
    },
    {
        "name" : "CV (%)",
        "type" : "STRING",
        "src" : "[38]"
        }
    ],
    "measures" : [
        {
            "name" : "state_ansi",
            "type" : "MEASURE",
            "src" : "[13]"
        },
        {
            "name" : "state_fips_code",
            "type" : "MEASURE",
            "src" : "[14]"
        },
        {
            "name" : "watershed_code",
            "type" : "MEASURE",
            "src" : "[24]"
        },
        {
            "name" : "country_code",
            "type" : "MEASURE",
            "src" : "[27]"
        },
        {
            "name" : "begin_code",
            "type" : "MEASURE",
            "src" : "[32]"
        },
        {
            "name" : "end_code",
            "type" : "STRING",
            "src" : "[33]"
        },
        {
            "name" : "Value",
            "type" : "MEASURE",
            "src" : "[37]"
        }
    ]
}
```

Appendix E

Schema and mappings for Case Study 1

```
{
   "name":"pig_mart",
   "source_meta":{
       "aim_1":{
            "DATE":{
                "type":"DATE",
                "from":"source:,
                "src":"//table[1]/tbody/tr[2]/th[1,2]"
            },
            "GEO":{
                "type":"String",
                "from":"Source",
                "src":"//table[1]/tbody/tr[7]/th[1]"
            },
            "ITEM":{
                "type":"String:,
                "from":"static",
                "src":"PIGS"
            },
            "METRIC":{
                "type":"String",
                "from":"Static",
                "src":"Slaughterings"
            },
            "UNITS":{
                "type":"String",
                "from":"static",
                "src":"Count"
            }
       },
       "aim_2":{
            "DATE":{
                "type":"DATE",
                "from":"source:,
                "src":"//table[2]/tbody/tr[2]/th[1,2]"
            },
            "GEO":{
                "type":"String",
            "from":"Source",
            "src":"//table[2]/tbody/tr[7]/th[1]"
        },
        "ITEM":{
            "type":"String:,
            "from":"static",
            "src":"PIGS"
        },
        "METRIC":{
            "type":"String",
            "from":"Static",
            "src":"Slaughterings"
        },
        "UNITS":{
            "type":"String",
            "from":"static",
            "src":"Count"
        }
},
"b_1":{
        "DATE":{
            "type":"DATE",
            "from":"source:,
            "src":"//table[1]/tbody/tr[1]/th[1]"
        },
        "GEO":{
            "type":"String",
            "from":"static",
            "src":"Ireland"
        },
        "ITEM":{
            "type":"String:,
            "from":"static",
            "src":"PIGS"
        },
        "METRIC":{
            "type":"String",
            "from":"Static",
            "src":"Slaughterings"
        },
        "UNITS":{
            "type":"String",
            "from":"static",
            "src":"Count"
        }
},
"b_2":{
         "DATE":{
            "type":"DATE",
            "from":"source:,
            "src":"//table[1]/tbody/tr[1]/th[1]"
        },
        "GEO":{
            "type":"String",
            "from":"static",
            "src":"Ireland"
        },
        "ITEM":{
            "type":"String:,
            "from":"static",
            "src":"PIGS"
        },
        "METRIC":{
            "type":"String",
            "from":"Static",
         "src":"Price"
     },
     "UNITS":{
         "type":"String",
         "from":"static",
         "src":"EUR_CENT"
     }
},
"p_1":{
      "DATE":{
         "type":"DATE",
         "from":"source:,
         "src":"//table[1]/tr[1-*]/th[1]"
     },
     "GEO":{
         "type":"String",
         "from":"static",
         "src":"GB"
     },
     "ITEM":{
         "type":"String:,
         "from":"static",
         "src":"PIGS"
     },
     "METRIC":{
         "type":"String",
         "from":"Static",
         "src":"Price"
     },
     "UNITS":{
         "type":"String",
         "from":"static",
         "src":"EUR_CENT"
     }
},
"p_2":{
     "DATE":{
         "type":"DATE",
         "from":"source:,
         "src":"//table[1]/tr[1-*]/th[1]"
     },
     "GEO":{
         "type":"String",
         "from":"static",
         "src":"GB"
     },
     "ITEM":{
         "type":"String:,
         "from":"static",
         "src":"PIGS"
     },
     "METRIC":{
         "type":"String",
         "from":"Static",
         "src":"Price"
     },
     "UNITS":{
         "type":"String",
         "from":"static",
         "src":"EUR_CENT"
     }
},
"p_3":{
     "DATE":{
         "type":"DATE",
            "from":"source:,
            "src":"//table[1]/tr[1-*]/th[1]"
        },
        "GEO":{
            "type":"String",
            "from":"static",
            "src":"GB"
        },
        "ITEM":{
            "type":"String:,
            "from":"static",
            "src":"PIGS"
        },
        "METRIC":{
            "type":"String",
            "from":"Static",
            "src":"Price"
        },
        "UNITS":{
            "type":"String",
            "from":"static",
            "src":"EUR_CENT"
        }
    },
    "bp_1":{
        "DATE":{
            "type":"DATE",
            "from":"source:,
            "src":"//table[1]/tr[1]/th[1]"
        },
        "GEO":{
            "type":"String",
            "from":"static",
            "src":"GB"
        },
        "ITEM":{
            "type":"String:,
            "from":"static",
            "src":"PIGS"
        },
        "METRIC":{
            "type":"String",
            "from":"Static",
            "src":"Price"
        },
        "UNITS":{
            "type":"String",
            "from":"static",
            "src":"GBP"
        }
    },
    "bp_2":{
         "DATE":{
            "type":"DATE",
            "from":"source:,
            "src":"//table[1]/tr[1]/th[1]"
        },
        "GEO":{
            "type":"String",
            "from":"static",
            "src":"GB"
        },
        "ITEM":{
            "type":"String:,
            "from":"static",
            "src":"PIGS"
```

```
     },
     "METRIC":{
         "type":"String",
         "from":"Static",
         "src":"Price"
     },
     "UNITS":{
         "type":"String",
         "from":"static",
         "src":"GBP"
     }
},
"c_1":{
      "DATE":{
         "type":"DATE",
         "from":"source:,
         "src":"//table[1]/tr[1]/th[2]"
     },
     "GEO":{
         "type":"String",
         "from":"static",
         "src":"WORLD"
     },
     "ITEM":{
         "type":"String:,
         "from":"static",
         "src":"PIGS"
     },
     "METRIC":{
         "type":"String",
         "from":"Static",
         "src":"FUTURE"
     },
     "UNITS":{
         "type":"String",
         "from":"static",
         "src":"QUOTE"
     }
},
"c_2":{
     "DATE":{
         "type":"DATE",
         "from":"source:,
         "src":"//table[1]/tr[1]/th[2]"
     },
     "GEO":{
         "type":"String",
         "from":"static",
         "src":"WORLD"
     },
     "ITEM":{
         "type":"String:,
         "from":"static",
         "src":"PIGS"
     },
     "METRIC":{
         "type":"String",
         "from":"Static",
         "src":"FUTURE"
     },
     "UNITS":{
         "type":"String",
         "from":"static",
         "src":"QUOTE"
     }
},
        "imf":{
            "DATE":{
                "type":"DATE",
                "from":"source:,
                "src":"//ExchangeRateReport/User_Selections/DATE_RANGE"
            },
            "GEO":{
                "type":"String",
                "from":"static",
                "src":"WORLD"
            },
            "ITEM":{
                "type":"String:,
                "from":"static",
                "src":"Currency"
            },
            "METRIC":{
                "type":"String",
                "from":"Static",
                "src":"Exchange Rate"
            },
            "UNITS":{
                "type":"String",
                "from":"static",
                "src":"SDR"
            }
        },
        "usda":{
            "DATE":{
                "type":"DATE",
                "from":"source:,
                "src":"//ExchangeRateReport/User_Selections/DATE_RANGE"
            },
            "GEO":{
                "type":"String",
                "from":"source",
                "src":"[21]"
            },
            "ITEM":{
                "type":"String:,
                "from":"source",
                "src":"[3]"
            },
            "METRIC":{
                "type":"String",
                "from":"source",
                "src":"[7]"
            },
            "UNITS":{
                "type":"String",
                "from":"source",
                "src":"[8]"
            }
        }
    },
    "facts":[
        {
            "sources":[
                "aim_1",
                "aim_2",
                "b_1",
                "b_2",
                "p_1",
                "p_2",
                "p_3",
                "bp_1",
                "bp_2",
                "usda"
            ],
        },
        {
            "sources":["c_1","c_2"]
        },
        {
            sources:["imf"]
        }
    ]
}
```

Appendix F

Schema and mappings for Case Study 2

```
{
"_id" : ObjectId("59e880840e4b9b1a17316d57"),
"name" : "oil_data_mart",
"source_meta" : {
    "PPOIL_USD" : {
        "DATE" : {
            "type" : "DATE",
            "from" : "SOURCE",
            "src" : "[0]"
        },
        "GEO" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "AMERICA"
        },
        "ITEM" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "PALM OIL"
        },
        "METRIC" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "PRICE"
        },
        "UNITS" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "USD"
        },
        "VALUE" : {
            "type" : "MEASURE",
            "from" : "SOURCE",
            "src" : "[1]"
        }
    },
    "PROIL_USD" : {
        "DATE" : {
            "type" : "DATE",
            "from" : "SOURCE",
            "src" : "[0]"
        },
        "GEO" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "AMERICA"
        },
        "ITEM" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "RAPESEED OIL"
        },
        "METRIC" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "PRICE"
        },
        "UNITS" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "USD"
        },
        "VALUE" : {
            "type" : "MEASURE",
            "from" : "SOURCE",
            "src" : "[1]"
        }
    },
    "PSOIL_USD" : {
        "DATE" : {
            "type" : "DATE",
            "from" : "SOURCE",
            "src" : "[0]"
        },
        "GEO" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "AMERICA"
        },
        "ITEM" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "SOYBEAN OIL"
        },
        "METRIC" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "PRICE"
        },
        "UNITS" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "USD"
        },
        "VALUE" : {
            "type" : "MEASURE",
            "from" : "SOURCE",
            "src" : "[1]"
        }
    },
    "PSUNO_USD" : {
        "DATE" : {
            "type" : "DATE",
            "from" : "SOURCE",
            "src" : "[0]"
        },
        "GEO" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "AMERICA"
        },
        "ITEM" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "SUNFLOWER OIL"
        },
        "METRIC" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "PRICE"
        },
        "UNITS" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "USD"
        },
        "VALUE" : {
            "type" : "MEASURE",
            "from" : "SOURCE",
            "src" : "[1]"
        }
    },
    "REN_SU" : {
        "DATE" : {
            "type" : "DATE",
            "from" : "SOURCE",
            "src" : "[0]"
        },
        "GEO" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "AMERICA"
        },
        "ITEM" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "SUNFLOWER OIL"
        },
        "METRIC_1" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "PRICE"
        },
        "UNITS_1" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "USD"
        },
        "VALUE_1" : {
            "type" : "MEASURE",
            "from" : "SOURCE",
            "src" : "[1]"
        },
        "METRIC_2" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "PRICE"
        },
        "UNITS_2" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "PERCENT_CHANGE"
        },
        "VALUE_2" : {
            "type" : "MEASURE",
            "from" : "SOURCE",
            "src" : "[2]"
        }
    },
    "WLD_COCONUT_OIL" : {
        "DATE" : {
            "type" : "DATE",
            "from" : "SOURCE",
            "src" : "[0]"
        },
        "GEO" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "WORLD"
        },
        "ITEM" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "COCONUT OIL"
        },
        "METRIC" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "PRICE"
        },
        "UNITS" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "USD"
        },
        "VALUE" : {
            "type" : "MEASURE",
            "from" : "SOURCE",
            "src" : "[1]"
        }
    },
    "WLD_PALM_OIL" : {
        "DATE" : {
            "type" : "DATE",
            "from" : "SOURCE",
            "src" : "[0]"
        },
        "GEO" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "WORLD"
        },
        "ITEM" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "PALM OIL"
        },
        "METRIC" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "PRICE"
        },
        "UNITS" : {
            "type" : "STRING",
            "from" : "STATIC",
            "src" : "USD"
        },
            "VALUE" : {
                "type" : "MEASURE",
                "from" : "SOURCE",
                "src" : "[1]"
            }
        },
        "WLD_SOYBEAN_OIL" : {
            "DATE" : {
                "type" : "DATE",
                "from" : "SOURCE",
                "src" : "[0]"
            },
            "GEO" : {
                "type" : "STRING",
                "from" : "STATIC",
                "src" : "WORLD"
            },
            "ITEM" : {
                "type" : "STRING",
                "from" : "STATIC",
                "src" : "SOYBEAN OIL"
            },
            "METRIC" : {
                "type" : "STRING",
                "from" : "STATIC",
                "src" : "PRICE"
            },
            "UNITS" : {
                "type" : "STRING",
                "from" : "STATIC",
                "src" : "USD"
            },
            "VALUE" : {
                "type" : "MEASURE",
                "from" : "SOURCE",
                "src" : "[1]"
            }
        }
    },
    "facts" : [
        {
            "sources" : [
                "PPOIL_USD",
                "PROIL_USD",
                "PSOIL_USD",
                "PSUNO_USD",
                "REN_SU"
            ],
            "join" : [
                {
                    "attr" : "DATE",
                    "type" : "EQUALITY"
                },
                {
                    "attr" : "GEO",
                    "type" : "EQUALITY"
                },
                {
                    "attr" : "METRIC",
                    "type" : "EQUALITY"
                }
            ],
            "int-approach" : "COLUMN_APPEND"
        },
        {
            "sources" : [
                "WLD_SOYBEAN_OIL",
                "WLD_PALM_OIL",
                "WLD_COCONUT_OILs"
            ],
            "join" : [
                {
                    "attr" : "DATE",
                    "type" : "EQUALITY"
                },
                {
                    "attr" : "GEO",
                    "type" : "EQUALITY"
                },
                {
                    "attr" : "METRIC",
                    "type" : "EQUALITY"
                }
            ],
            "int-approach" : "COLUMN_APPEND"
        }
    ]
}
```

Appendix G

Schema and mappings for Case Study 3

```
{
"_id" : ObjectId("59e9aea39ba46d546f783a36"),
"name" : "dairy_data_mart",
"source_meta" : {
    "apro_mk_colm_1_Data" : {
        "DATE" : {
            "type" : "DATE",
            "from" : "SOURCE",
            "src" : "[0]"
        },
        "GEO" : {
            "type" : "STRING",
            "from" : "SOURCE",
            "src" : "[1]"
        },
        "ITEM" : {
            "type" : "STRING",
            "from" : "SOURCE",
            "src" : "[2]"
        },
        "METRIC" : {
            "type" : "STRING",
            "from" : "SOURCE",
            "src" : "[3]"
        },
        "UNITS" : {
            "type" : "STRING",
            "from" : "SOURCE",
            "src" : "[3]"
        },
        "VALUE" : {
            "type" : "MEASURE",
            "from" : "SOURCE",
            "src" : "[4]"
        }
    },
    "download(1).csv" : {
        "DATE" : {
                "type" : "DATE",
                "from" : "SOURCE",
                "src" : "[34]"
            },
            "GEO" : {
                "type" : "STRING",
                "from" : "SOURCE",
                "src" : "[28]"
            },
            "ITEM" : {
                "type" : "STRING",
                "from" : "SOURCE",
                "src" : "[3]"
            },
            "METRIC" : {
                "type" : "STRING",
                "from" : "SOURCE",
                "src" : "[9]"
            },
            "UNITS" : {
                "type" : "STRING",
                "from" : "SOURCE",
                "src" : "[10]"
            },
            "VALUE" : {
                "type" : "MEASURE",
                "from" : "SOURCE",
                "src" : "[37]"
            }
        },
        "milk.de > Home - Dairy World Information from the ZMB - 2" : {
            "DATE" : {
                "type" : "DATE",
                "from" : "STATIC",
                "src" : "function(){return new Date().toString();}"
            },
            "GEO" : {
                "type" : "STRING",
                "from" : "STATIC",
                "src" : "GERMANY"
            },
            "ITEM" : {
                "type" : "STRING",
                "from" : "SOURCE",
                "src" : "//table[2]/tbody/tr/td[1]"
            },
            "METRIC" : {
                "type" : "STRING",
                "from" : "STATIC",
                "src" : "PRODUCTION"
            },
            "UNITS" : {
                "type" : "STRING",
                "from" : "STATIC",
                "src" : "METRIC TONNES"
            },
            "VALUE" : {
                "type" : "MEASURE",
                "from" : "SOURCE",
                "src" : "//table[2]/tbody/tr/td[2]"
            }
        }
    },
    "facts" : [
        {
            "sources" : [
                "apro_mk_colm_1_Data",
                "download(1).csv",
                "milk.de > Home - Dairy World Information from the ZMB - 2"
            ],
            "join" : [
                {
                    "attr" : "DATE",
                    "type" : "EQUALITY"
                }
            ],
            "int-approach" : "COLUMN_APPEND"
        }
    ]
}
```