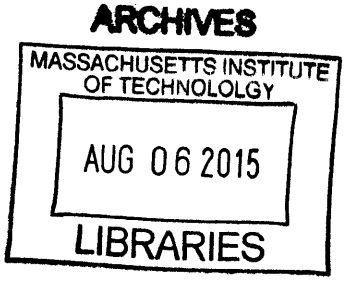
OPM Model-Based Integration of Multiple Data Repositories

Greg Wilmer

SUBMITTED TO THE SYSTEM DESIGN AND MANAGEMENT PROGRAM IN
PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

MASTER OF SCIENCE IN ENGINEERING AND MANAGEMENT
AT THE
MASSACHUTSETTS INSTITUTE OF TECHNOLOGY
JANUARY 2015 [February 2015]



Signature of Author: Signature redacted
Greg Wilmer
Systems Design and Management Program
January 2015

Certified By: Signature redacted
Dov Dori
Visiting Professor at Engineering Systems Division
Thesis Supervisor

Accepted By: Signature redacted
Patrick Hale
Executive Director
Systems Design and Management Program

This page intentionally left blank.

OPM Model-Based Integration of Multiple Data Repositories

Greg Wilmer

Submitted to the System Design and Management Program
on November 30, 2014 in Partial Fulfillment of the
Requirements for the Degree of Master of Science in
Engineering and Management

ABSTRACT

Data integration is at the heart of a significant portion of current information system implementations. As companies continue to move towards a diverse, growing set of Commercial Off the Shelf (COTS) applications to fulfill their information technology needs, the need to integrate data between them continues to increase. In addition, these diverse application portfolios are becoming more geographically dispersed as more software is provided using the Software as a Service (SaaS) model, and companies continue the pattern of moving their internal data centers to cloud-based computing.

As the growth of data integration activities continues, several prominent data integration patterns have emerged, and commercial software packages have been created that covers each of the patterns below:

1. Bulk and/or batch data extraction and delivery (ETL, ELT, etc.)
2. Messaging / Message-oriented data movement
3. Granular, low-latency data capture and propagation (data synchronization)

As the data integration landscape within an organization, and between organizations, becomes larger and more complex, opportunities exist to streamline aspects of the data integrating process not covered by current toolsets including:

1. **Extensibility by third parties**. Many COTS integration toolsets today are difficult if not impossible to extend by third parties.

2. Capabilities to handle **different types of structured data** from relational to hierarchical to graph models

3. **Enhanced modeling capabilities** through use of data visualization and modeling techniques and tools

4. Capabilities for **automated unit testing** of integrations

5. A **unified toolset that covers all three patterns**, allowing an enterprise to implement the pattern that best suites business needs for the specific scenario

6. A **Web-based toolset** that allows configuration, management and deployment via Web-based technologies allowing geographical indifference for application deployment and integration

While discussing these challenges with a large Fortune 500 client, they expressed the need for an enhanced data integration toolset that would allow them to accomplish such tasks. Given this request, the Object Process Methodology (OPM) and the Opcat toolset were used to begin design of a data integration toolset that could fulfill these needs. As part of this design process, lessons learned covering both the use of OPM in software design projects as well as enhancement requests for the Opcat toolset were documented.

Thesis Supervisor: Dov Dori
Title: Visiting Professor at Engineering Systems Division

Acknowledgements

I would like to thank Dov Dori for his guidance and support during the thesis process. His experience and insights were invaluable.

I would also like to thank Pat Hale and all of the staff at SDM for their support and guidance throughout the SDM program.

Last but definitely not least I would like to thank my wife and children for allowing me to spend countless hours behind the computer and six months in my two hundred and fifty square foot dream house on the MIT campus. I couldn't have done this without your love and support.

Table of Contents

Table of Figures

Research Objectives

The intent of this research is to utilize and assess OPM modeling and the OPCAT toolset to aid in the design of a systems data integration framework for a Fortune 500 client. Through the process we will use OPM and OPCAT to brainstorm, document and communicate design elements for the requested system with all team members. We will also document observations made while utilizing these tools to fulfill design requirements in a predominantly software environment. Lastly we will evaluate the process and propose future research topics.

Introduction

Data integration is at the heart of many current information system projects. As companies continue to move towards a diverse, growing set of Commercial Off the Shelf (COTS) applications to fulfill their information technology needs, the need to integrate data between them continues to increase. In addition, these diverse application portfolios are becoming more geographically dispersed as more software is provided using the Software as a Service (SaaS) model, and companies continue the pattern of moving their internal data centers to cloud-based computing.

As the number of applications within an organization continues to rise, so does the number of integration operations required to implement applications within the context of the organization. This results in an increased percentage of information system implementation cost being attributed to data integration activities. In addition, as different applications are implemented over time within an organization, the needs and approaches to implement those integrations inevitably varies, resulting in a broad, overly complex data integration landscape that must be managed and maintained. Bernstein and Haas (2008) (Bernstein & Hass, 2008) noted that information integration is frequently cited as the largest and most expensive challenge that information-technology shops face, and that information integration amounts to 40% of those shops' budgets. Akbay (Akbay, 2004) provided similar thoughts and statistics around expenditures. Lawson and Sharma (2014) (Lawson & Sharma, 2014) stated that "Integration is key to most of today's big tech trends, including the Internet of Things, mobile, cloud adoption and digital business... There's a lot of integration work, so enterprises will be keen to adopt agile approaches to integration".

As the growth of data integration activities continues, several prominent data integration patterns have emerged, and corresponding commercial implementation software packages have been created. Patterns have been categorized slightly differently, but most have the same gist. Bernstein and Haas (2008) delineate them as Data Warehouse Loading through the use of Extract-Transform-Load (ETL) tools, Virtual Data Integration through the use of query mediation or Enterprise Information Integration (EII) system via Web Services or other services, and Message Mapping through Message Oriented Middleware (MOM) (2013) (Moradi & Bahreininejad, 2013). Hohpe and Woolf (Hohpe & Woolf, 2004) have categorized the major patterns as File Transfer, Shared Database Remote Procedure Invocation, and Messaging (2004). We group these patterns into the following three categories:

1. Bulk or batch data extraction and delivery,

2. Messaging or message-oriented data movement, and

3. Granular, low-latency data capture and propagation

While Bulk or batch data extraction and delivery is most synonymous with Data Warehousing, this integration pattern is used in many organizations today for a multitude of different business scenarios. More commonly known as Extract, Transform, Load (ETL) or Extract, Load, Transform (ELT), this pattern consists of extracting and batching a set of data from a source system on a periodic basis, transforming it, and then loading it to a destination system. The difference between ETL and ELT is simply the system on which transformations are completed. In essence, do you extract and transform the data before being loaded into the target system, or do you extract and load to the target system, and then transform. ELT has become popular as organizations started purchasing large, powerful data warehouse hardware and found that by loading legacy data directly into the data warehouse in staging tables or data sets, they could leverage the more powerful data warehouse equipment to transform that legacy data into new formats and models. Kimball and Caserta (Kimball & Caserta, 2004) provided an excellent overview of ETL in their book "The Data Warehouse ETL Toolkit". While ETL and ELT have been around for quite a while, they still continue to play a major role in data integration today.

Message Oriented Data Movement or Message Oriented Middleware (MOM) is a pattern by which data is integrated between disparate applications by passing messages between them. Cambell, Coulson and Kounavis (Campbell, Goulson, & Kounavis, 1999) defined it as "message-passing and message-queuing middleware, in which information is passed in the form of a message from one program to one or more other programs." . Hohpe and Woolf (Hohpe & Woolf, 2004) described it as an architecture that facilitates applications publishing or sending data in the form of a message to a central message channel that can be read by one or more applications that desire that data (2004). Publish / Subscribe or Pub / Sub and Enterprise Service Bus architectures are variants of MOM. A good example of MOM is real time inventory management integration between systems. Consider, for example, a company that has a centralized inventory management system, which must be accessed and updated in real time by brick and mortar stores and a Web site. Each application at each physical location must be able to access inventory information in real time and update it as inventory is sold and returned. Batch ETL or ELT processing simply wouldn't work in this scenario. Zhai, Guo and Li provide an overview of some of the different MOM middleware (2011) (Zhai, Guo, Cui, & Li, 2011).

Data Synchronization is similar to MOM in that it typically has low latency (i.e., it is real-time or near real-time fashion), but synchronization tools have more focus on like to like schemas (minimal transformation). Examples include database primary / backup for failover, and consolidation / deconsolidation such as when a retail central office or master database must synchronize data to a replica database physically located at a store.

As the data integration landscape within an organization, and between organizations, becomes larger and more complex, opportunities exist to streamline aspects of the data integrating process not covered by current toolsets, including:

1. **Extensibility by third parties**. Many COTS integration toolsets today are difficult if not impossible to extend by third parties,

2. Capabilities to handle **different types of structured data models** from relational to hierarchical to graph models,

3. **Enhanced modeling capabilities** through use of data visualization and modeling techniques and tools,

4. Capabilities for **automated unit testing** of integrations,

5. A **unified toolset that covers all three patterns**, allowing an enterprise to implement the pattern that best suites business needs for the specific scenario, and

6. A **Web-based toolset** that allows configuration, management and deployment via Web-based technologies allowing geographical indifference for application deployment and integration

Regardless of the pattern and business scenario, all have similar traits, mainly reading, processing, and writing of data from one source to another. These similarities are described in more detail in the accompanying OPM model, which is presented in the following section.

Requirements and Opportunities for Improvement

Extensibility by Third Parties

To attain better integration toolsets, we need an approach that allows multiple entities or parties to contribute to the effort of the integration toolset. Open sourcing of the integration toolset is one way to allow multiple parties to be involved in its creation, enhancement and maintenance. However, simply providing source code isn't enough. The architecture and data model of the toolset must be made such that extensibility can be accomplished easily and with minimal risk to the existing code base.

One mechanism by which the integrating toolset can be made more extensible by third parties is to make the underlying data model for the integration toolset as generic as possible. By making the underlying data model generic in nature, code changes and additions can be made without changing the underlying data model, thus minimizing the need to refactor existing code. One pattern used consistently throughout the data integrating system model is the generic "settings" pattern. This pattern is shown in Figure 1.

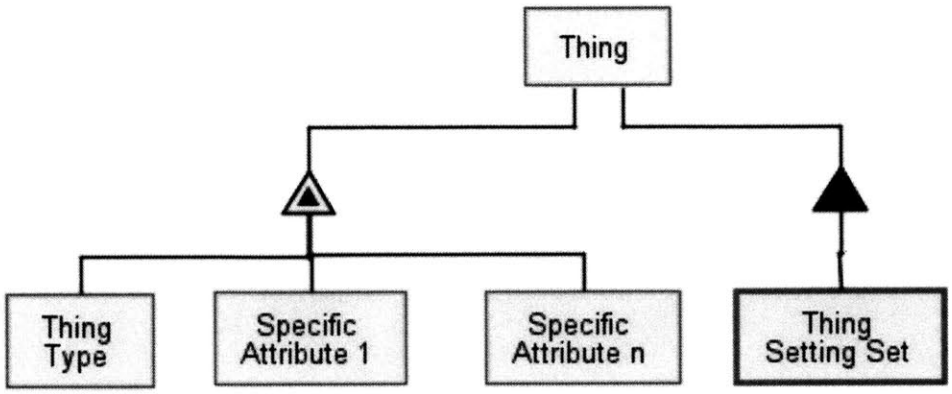


Figure 1 - Generic Settings Pattern – Thing unfolded

Thing **exhibits** Specific Attribute 1, Specific Attribute n, **and** Thing Type.
Thing **consists of** Thing Setting Set.

In the example above, a Thing is described by a type (Thing Type) and a set of specific attributes (Specific Attribute 1 to Specific Attribute n). It is also described by a generic set of attributes defined and stored by the Thing Setting Set. The Thing Setting Set is a collection of key/value pairs that can be used to generically define and collect other attributes for the Thing.

Each Thing Setting Set is defined for a specific type of Thing as shown in Figure 2.

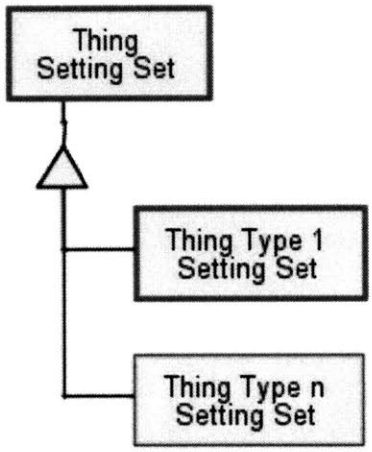


Figure 2 – Generic Settings Pattern - Thing Setting Set unfolded

Thing Type 1 Setting Set **is a** Thing Setting Set.
Thing Type n Setting Set **is a** Thing Setting Set.

Each Thing Type Setting Set can have a unique set of attributes for that specific Thing Type, as shown in Figure 3.

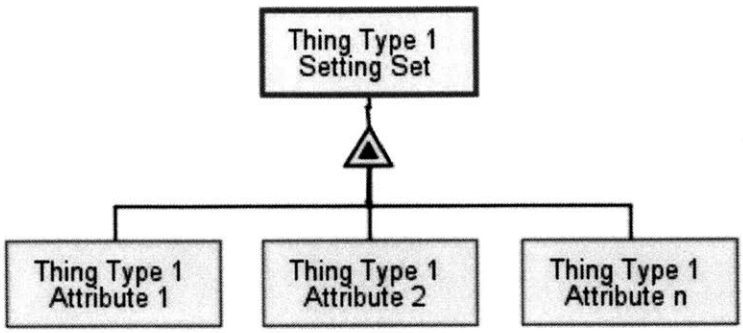


Figure 3 – Generic Settings Pattern - Thing Type 1 Setting Set unfolded

Thing Type 1 Setting Set **exhibits** Thing Type 1 Attribute 1, Thing Type 1 Attribute 2, **and** Thing Type 1 Attribute n.

Concrete examples of this can be found throughout the Data Integrating Model. One specific example is creating and storing Connection Definitions. A Connection Definition in the Data Integrating Model is used to store information about a connection that is utilized to connect to a System to read or write data. There will likely be many different Connection Types including DAS/NAS Connection Types, SSH Connection Types, SMTP Connection Types, Queue Connection Types, Database Connection Types, etc. All Connection Definitions might have common attributes such as a Connection Name, but other attributes vary depending on the Connection Type. Figure 4 shows the model for a Connection Definition.

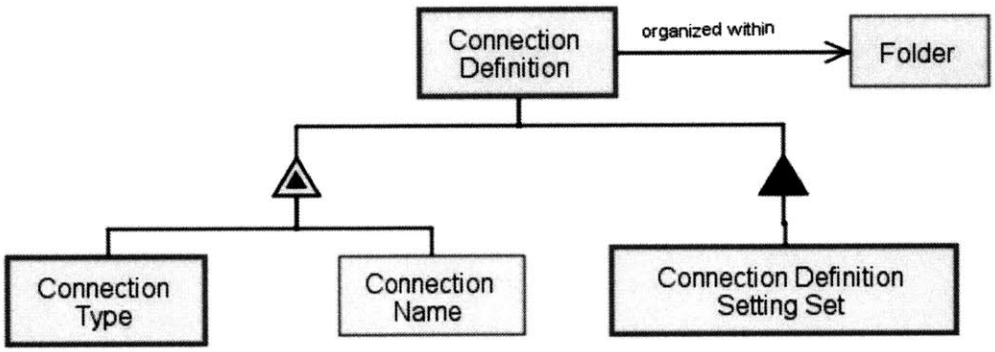


Figure 4 - Generic Settings Pattern - Connection Definition unfolded

Connection Definition **exhibits** Connection Type **and** Connection Name.
Connection Definition **consists of** Connection Definition Setting Set.
Connection Definition **organized within** Folder.

Each Connection Type has a specific Connection Definition Setting Set as shown in Figure 5.

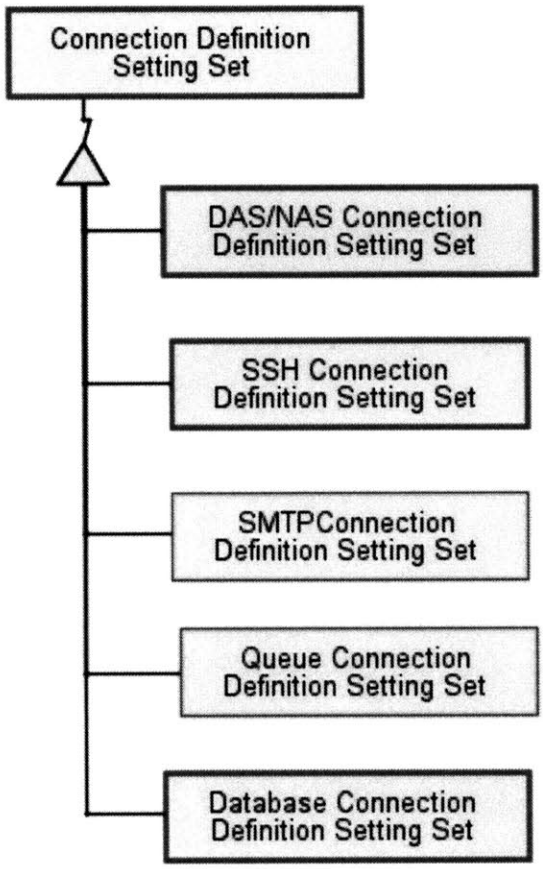


Figure 5 - Generic Settings Pattern - Connection Definition Setting Set unfolded

Database Connection Definition Setting Set **is a** Connection Definition Setting Set.
DAS/NAS Connection Definition Setting Set **is a** Connection Definition Setting Set.
Queue Connection Definition Setting Set **is a** Connection Definition Setting Set.
SMTP Connection Definition Setting Set **is a** Connection Definition Setting Set.
SSH Connection Definition Setting Set **is a** Connection Definition Setting Set.

An example Setting Set for the SSH Connection Setting Set contains the following as shown in Figure 6.

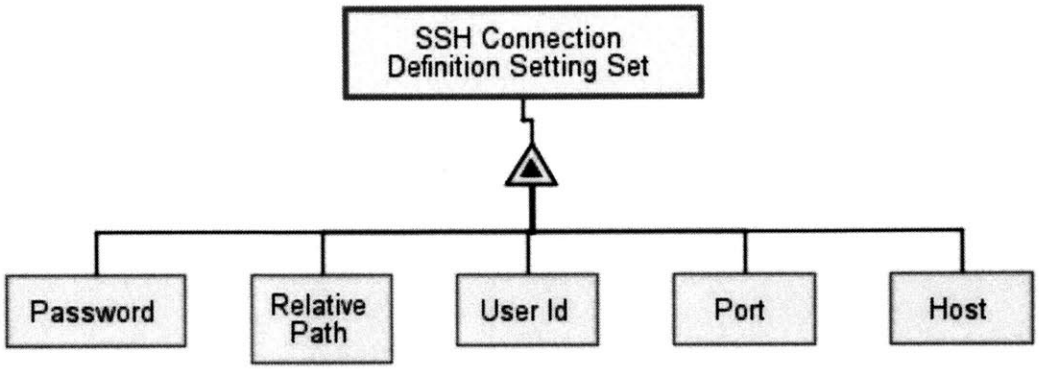


Figure 6 - Generic Settings Pattern - SSH Connection Definition Setting Set unfolded

SSH Connection Definition Setting Set **exhibits** Port, Relative Path, User Id, Password, **and** Host.

By applying the generic settings pattern to Connection Definitions, third parties can add additional Connection Types and Definition Capabilities without the need to change the underlying Data Integration System data model.

Different Types of Structured Models

The second opportunity to streamline integration activities is to allow for the integration of different types of structured data. Three common types of data models are the relational, hierarchical and graph-based models. Many integration toolsets allow two databases modeled in the same data model to be integrated (e.g., relational with relational), but additional work needs to be done for integrating data with disparate model types. Allowing data that is modeled in a graph-based format to be integrated with data that is modeled using a relational model format is increasingly important as companies desire to mix legacy relational data from COTS applications with big data or other graph-based modeled data from the World Wide Web.

In order to allow for integrating different data model types, a generic Data Model Definition will be supported. Each model type will have common attributes, such as a Model Name, Model Type and whether that Model can be shared across integrations. Initial supported Data Model Definition types will include Relational Models, such as Relational database structures, Graph Models, such as RDF structures, and Hierarchical Models, such as XML-based structures. Figure 8 shows the general Data Model Definition structure.

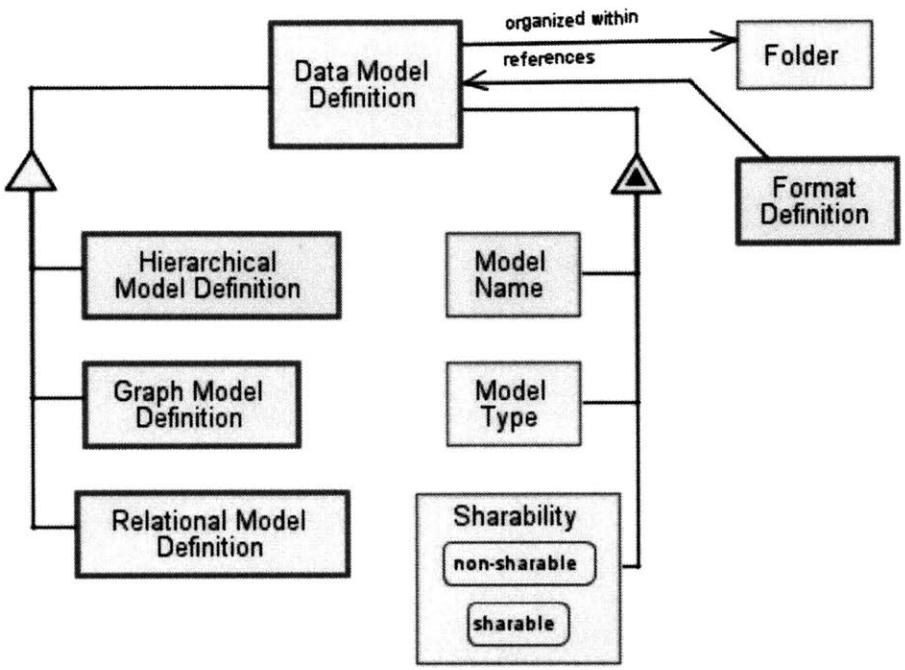


Figure 7 - Structured Models - Data Model Definition unfolded

Format Definition **references** Data Model Definition.
XML Attribute Setting Set **exhibits** XPath Expression.
Fixed Length Attribute Setting Set **exhibits** Start Position **and** End Position.
Delimited Attribute Setting Set **exhibits** Position.
Data Model Definition **exhibits** Model Name, Model Type, **and** Sharability.
Sharability **can be** non-sharable **or** sharable.
Data Model Definition **organized within** Folder.
Graph Model Definition **is a** Data Model Definition.
Relational Model Definition **is a** Data Model Definition.
Hierarchical Model Definition **is a** Data Model Definition.

The Relational Model Definition is a meta-model that supports description of a generic relational model, including its entities, attributes and the relationships among them. Figure 9 depicts the Relational Model Definition.

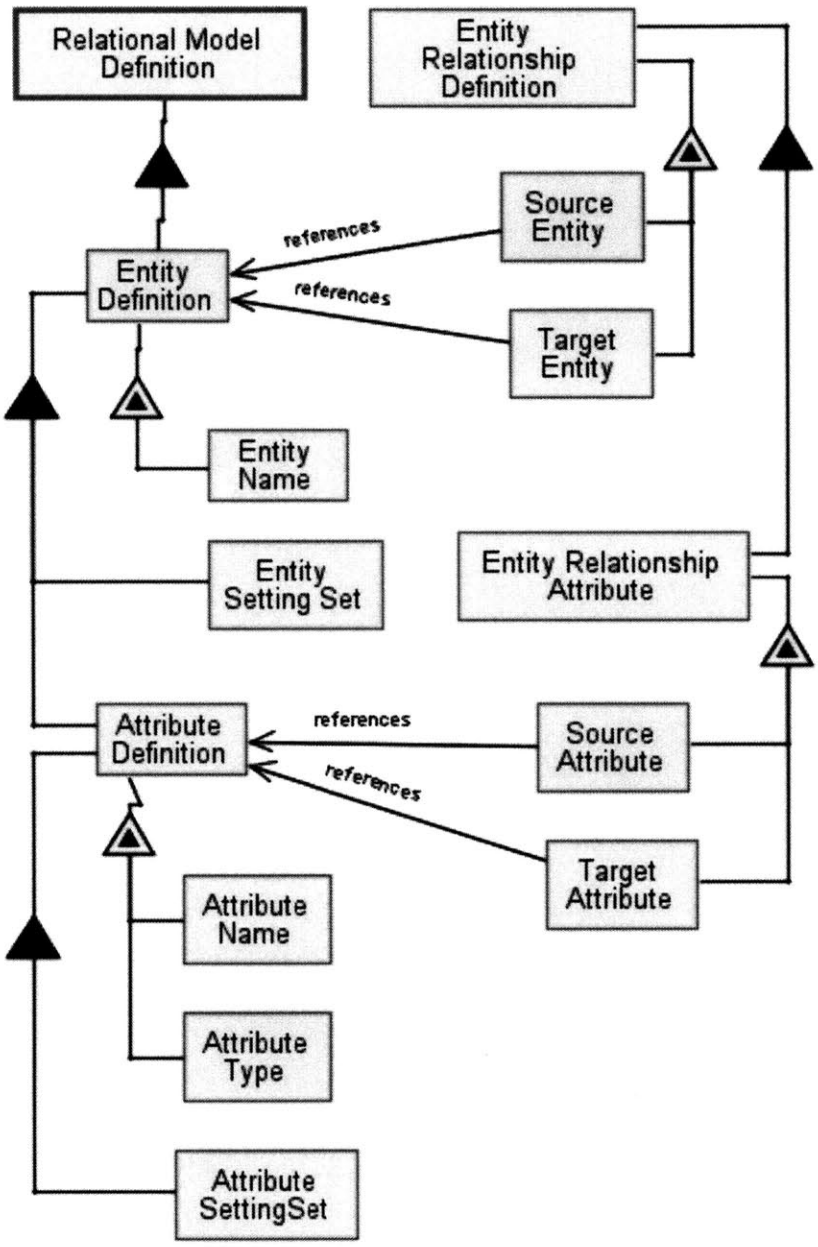


Figure 8 - Structured Model - Relational Model Definition unfolded

Entity Relationship Definition **exhibits** Source Entity **and** Target Entity.
Source Entity **references** Entity Definition.
Target Entity **references** Entity Definition.
Entity Relationship Definition **consists of** Entity Relationship Attribute.
Entity Relationship Attribute **exhibits** Source Attribute **and** Target Attribute.
Source Attribute **references** Attribute Definition.
Target Attribute **references** Attribute Definition.
Relational Model Definition **consists of** Entity Definition.

Entity Definition **exhibits** Entity Name.
Entity Definition **consists of** Attribute Definition **and** Entity Setting Set.
Attribute Definition **exhibits** Attribute Name **and** Attribute Type.
Attribute Definition **consists of** Attribute Setting Set.

Figure 10 depicts the Graph Meta Model.

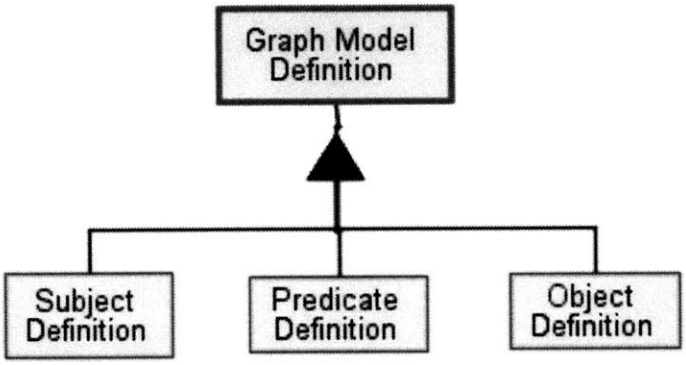


Figure 9 - Structured Model - Graph Model Definition unfolded

Graph Model Definition **consists of** Subject Definition, Predicate Definition, **and** Object Definition.

In the Data Integrating System, entities and attributes of a relational model can be mapped to and from any Object Definition with a Graph Model as well as any Entity or Attribute of the Hierarchical Model.

Enhanced Modeling Capabilities

The third opportunity to streamline integration activities is by utilization of advanced data visualization techniques to assist in the modeling process of structured data. Much research has been done on the role of ontologies (global and local) in integrating data between disparate systems (2014) (Zhang, 2014), (2013) (Varughese, 2013), (2014) (Xu, Yan, Wang, & Gong, 2014) amongst others. The most time consuming and difficult part of these approaches is the definition of the ontologies, especially when this is done in conjunction with COTS packages. Brands (Brands, 2014) discusses the role of data visualization and discovery in making business decisions, suggesting 90% of the people in a decision making survey agreed that the use of data visualization to present information significantly reduces the time to make decisions. Regardless of whether those decisions are business oriented or technically oriented, visualization clearly helps individuals more quickly understand what they are looking at. The same could be argued for visual data modeling in the context of ontology definition for integration purposes.

Automated Unit Testing

The concept of unit and system automated testing has been largely ignored in the field of data integration. While many understand the value of automated testing as it pertains to application development, its use in data integration scenarios has been lacking. Moradi and Bahreininejad provide a framework for evaluating core features of Enterprise Integration Middleware technologies. In this framework, they specify several functional criteria including Messaging, Semantic Transformation (modeling and transformation), Bulk Data Movement (ETL and ELT), etc. but the concept of a testing framework for the integrating process is missing. (Moradi & Bahreininejad, 2013).
Kabiri and Chiadmi also express the need for testing capabilities associated with ETL processes in their suggested research opportunities in the article Survey of ETL Processes. "Tests are fundamental aspects of software engineering. In spite of this importance, and regarding ETL, they are neglected. Thus, an automatic or even a semi-automatic approach for validating or getting data for tests is very hopeful" (Kabiri & Chiadmi, 2013).

As part of the Data Integrating process, a specific sub-process will be created for Test Case Defining and Executing, as shown in Figure 10. This step will be optional depending on desire from the User.

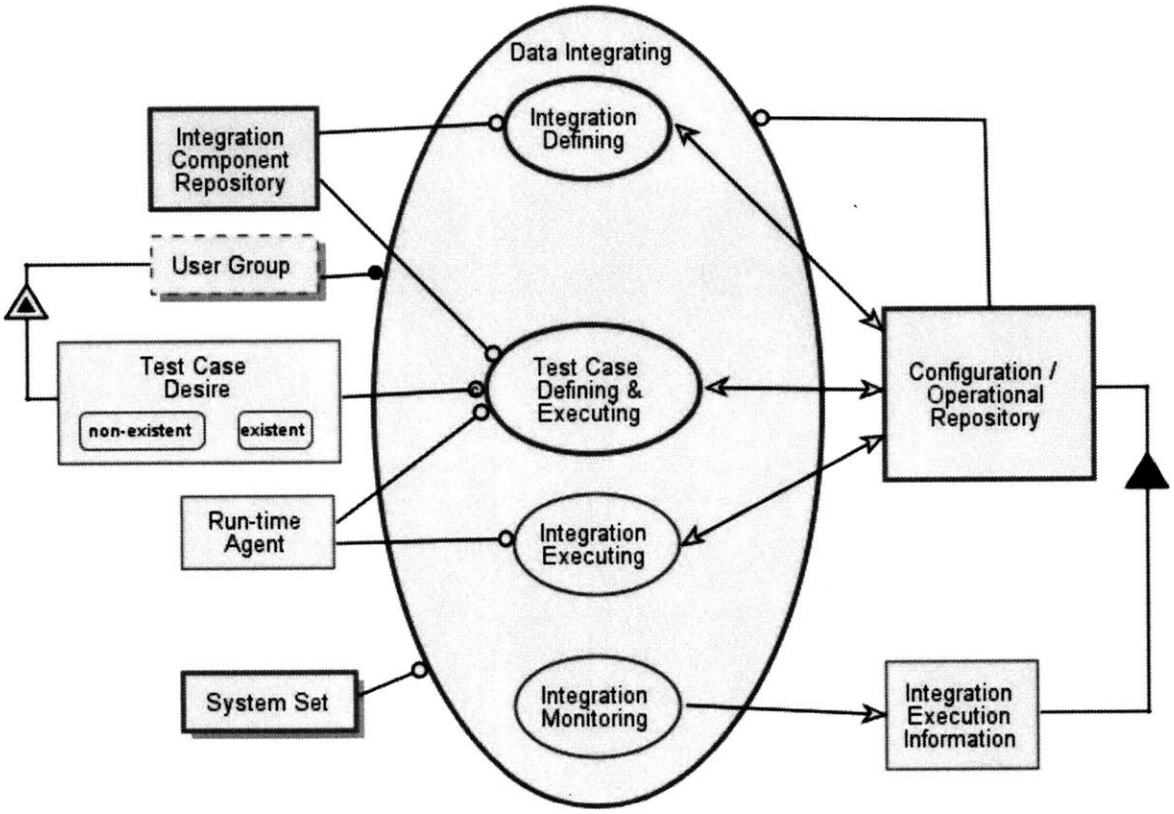


Figure 10 – Automated Unit Testing - Data Integrating in-zoomed

System Set **is physical.**
System Set **can be** not integrated **or** integrated.
    not integrated **is initial.**
    integrated **is final.**
User Group **is environmental and physical.**
User Group **can be** uninformed **or** informed.
    uninformed **is initial.**
    informed **is final.**
Data Integrating **requires** Data Integrating System.
Data Integrating **changes** User Group **from** uninformed **to** informed **and** System Set **from** not integrated **to** integrated.

The Test Case Defining and Executing process is further broken down in Figure 11 to the subsequent Test Case Defining and Test Case Executing processes.

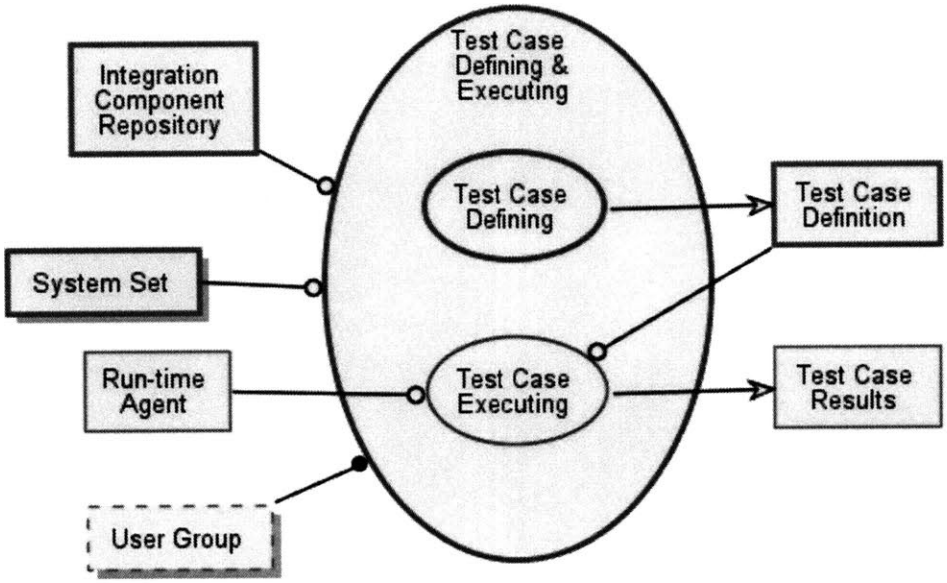


Figure 11 - Automated Unit Testing - Test Case Defining & Executing in-zoomed

System Set **is physical.**
User Group **is environmental and physical.**
User Group **handles** Test Case Defining & Executing.
Test Case Defining & Executing **consists of** Test Case Defining **and** Test Case Executing.
Test Case Defining & Executing **requires** System Set **and** Integration Component Repository.
Test Case Defining & Executing **zooms into** Test Case Defining **and** Test Case Executing.
    Test Case Defining **yields** Test Case Definition.
    Test Case Executing **requires** Test Case Definition **and** Run-time Agent.
    Test Case Executing **yields** Test Case Results.

Test Cases for an integration typically come in the form of pre-defined input data set(s) for a given test scenario. The validation for a given test is pre-defined output data set(s) that results from the processing of the input data set(s).

The Data Integrating System model is centered on an Integration Directed Acyclic Graph or DAG of components that read, process and write data, with data being passed between components (see Figure 50 – Component Definition Set Defining in-zoomed). Thus, a test case is linked to an Integration DAG Definition and can cover one or more nodes within that DAG. For each node within the DAG, input and/or output (validation) data is provided for that node. Figure 12 shows a portion of the Test Case Defining process that includes selecting the Integration DAG, as well as creating the node data sets for each node that will participate in the test.

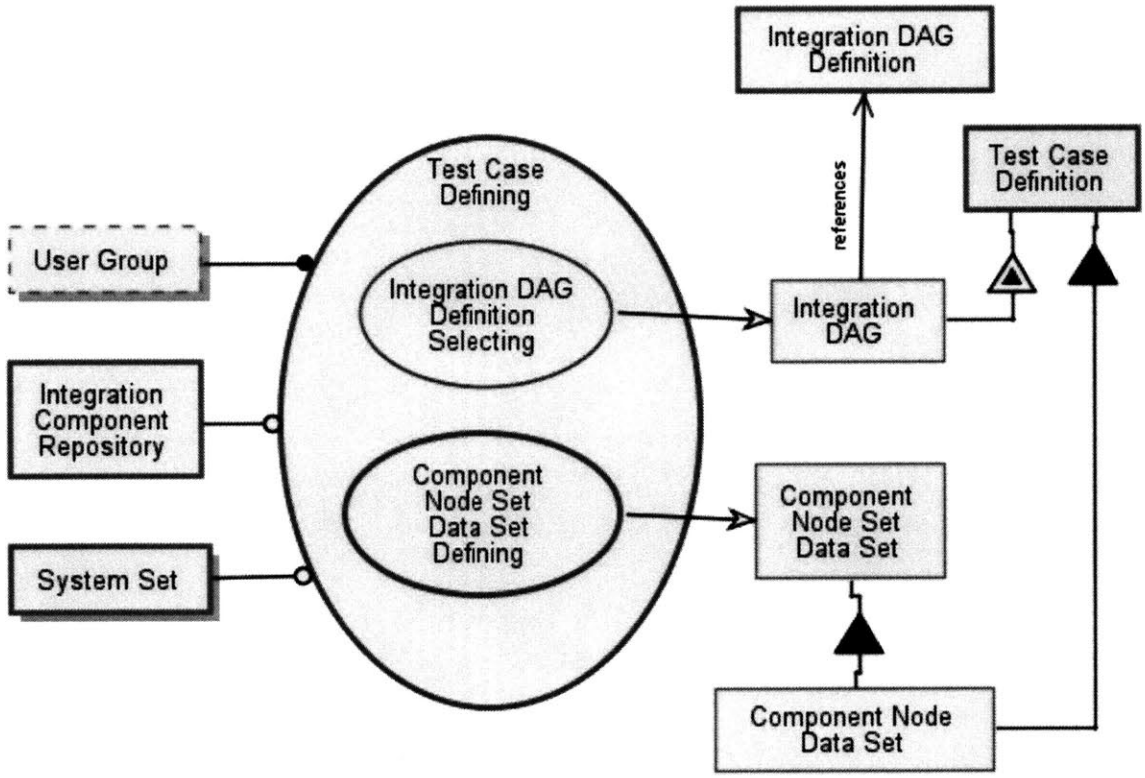


Figure 12 - Automated Unit Testing - Test Case Defining in-zoomed

System Set **is physical.**
User Group **is environmental and physical.**
User Group **handles** Test Case Defining.
Test Case Definition **exhibits** Integration DAG.
Integration DAG **references** Integration DAG Definition.
Test Case Definition **consists of many** Component Node Data Sets.
Component Node Set Data Set **consists of** Component Node Data Set.
Test Case Defining **consists of** Integration DAG Definition Selecting **and** Component Node

Set Data Set Defining.
Test Case Defining **requires** Integration Component Repository **and** System Set.
Test Case Defining **zooms into** Integration DAG Definition Selecting **and** Component Node Set Data Set Defining.
Integration DAG Definition Selecting **yields** Integration DAG.
Component Node Set Data Set Defining **yields** Component Node Set Data Set.

Component Node Set Data Set Defining is the process of creating one or more Component Node Data Sets for one or more nodes of the DAG, as expressed in the OPD in Figure 13).

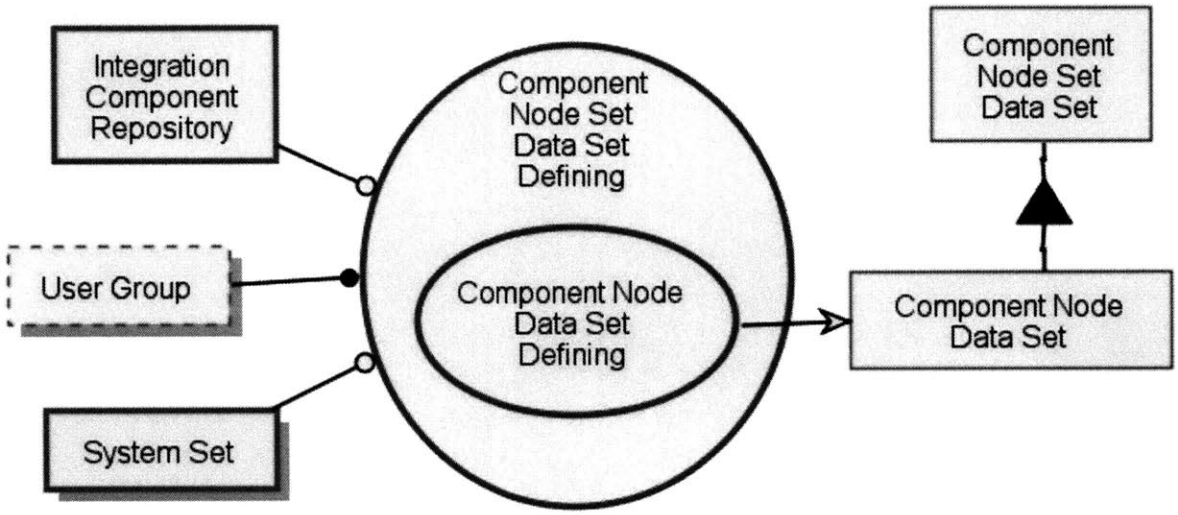


Figure 13 - Automated Unit Testing - Component Node Set Data Set Defining in-zoomed

System Set **is physical.**
User Group **is environmental and physical.**
User Group **handles** Component Node Set Data Set Defining.
Component Node Set Data Set **consists of** Component Node Data Set.
Component Node Set Data Set Defining **consists of** Component Node Data Set Defining.
Component Node Set Data Set Defining **requires** System Set **and** Integration Component Repository.
Component Node Set Data Set Defining **zooms into** Component Node Data Set Defining.
Component Node Data Set Defining **yields** Component Node Data Set.

Component Node Data Set Defining is the process of defining a test data set for a single node within the DAG. It consists of creating the input and output (validation) data sets for a single node, as shown in Figure 14.

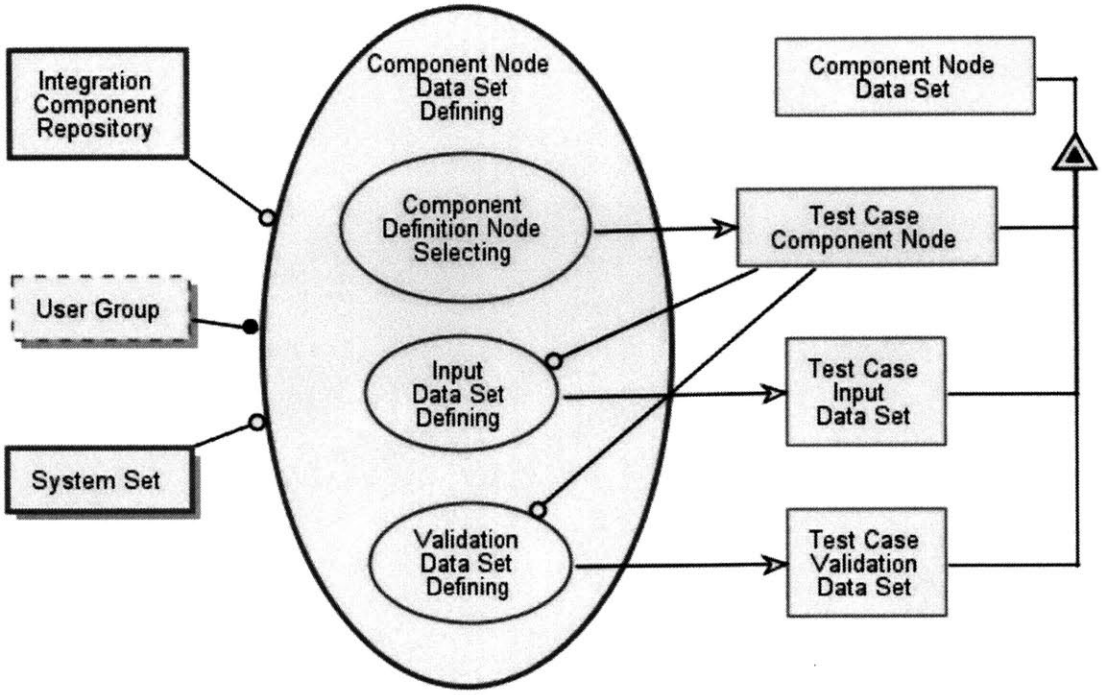


Figure 14 - Automated Unit Testing - Component Node Data Set Defining in-zoomed

System Set **is physical.**
User Group **is environmental and physical.**
User Group **handles** Component Node Data Set Defining.
Component Node Data Set **exhibits** Test Case Input Data Set, Test Case Validation Data Set, **and** Test Case Component Node.
Component Node Data Set Defining **consists of** Input Data Set Defining, Validation Data Set Defining, **and** Component Definition Node Selecting.
Component Node Data Set Defining **requires** System Set **and** Integration Component Repository.
Component Node Data Set Defining **zooms into** Component Definition Node Selecting, Input Data Set Defining, **and** Validation Data Set Defining.

Component Definition Node Selecting **yields** Test Case Component Node.
Input Data Set Defining **requires** Test Case Component Node.
Input Data Set Defining **yields** Test Case Input Data Set.
Validation Data Set Defining **requires** Test Case Component Node.
Validation Data Set Defining **yields** Test Case Validation Data Set.

The resultant Test Case Definition, defined in Figure 15, includes the following elements:

- **Integration DAG** – The Integration DAG Definition for which this test case is being created

- **Component Node Data Set** – A collection of data sets that can be used as input data to the nodes of the DAG and output data sets that can be used to validate the output of the DAG.

- **Test Case Component Node** – An individual node in the DAG that will be tested

- **Test Case Input Data Set** – The input data set for a specific Test Case Component Node. This data will be fed to the component as part of the execution of the test.

- **Test Case Output (Validation) Data Set** – The output data set for a specific Test Case Component Node that will be used to validate the output of that node. Output data from execution of the node with input data provided that equals the supplied output data set will indicate success, while a mismatch in the generated output compared to the supplied output will indicate failure of the test.

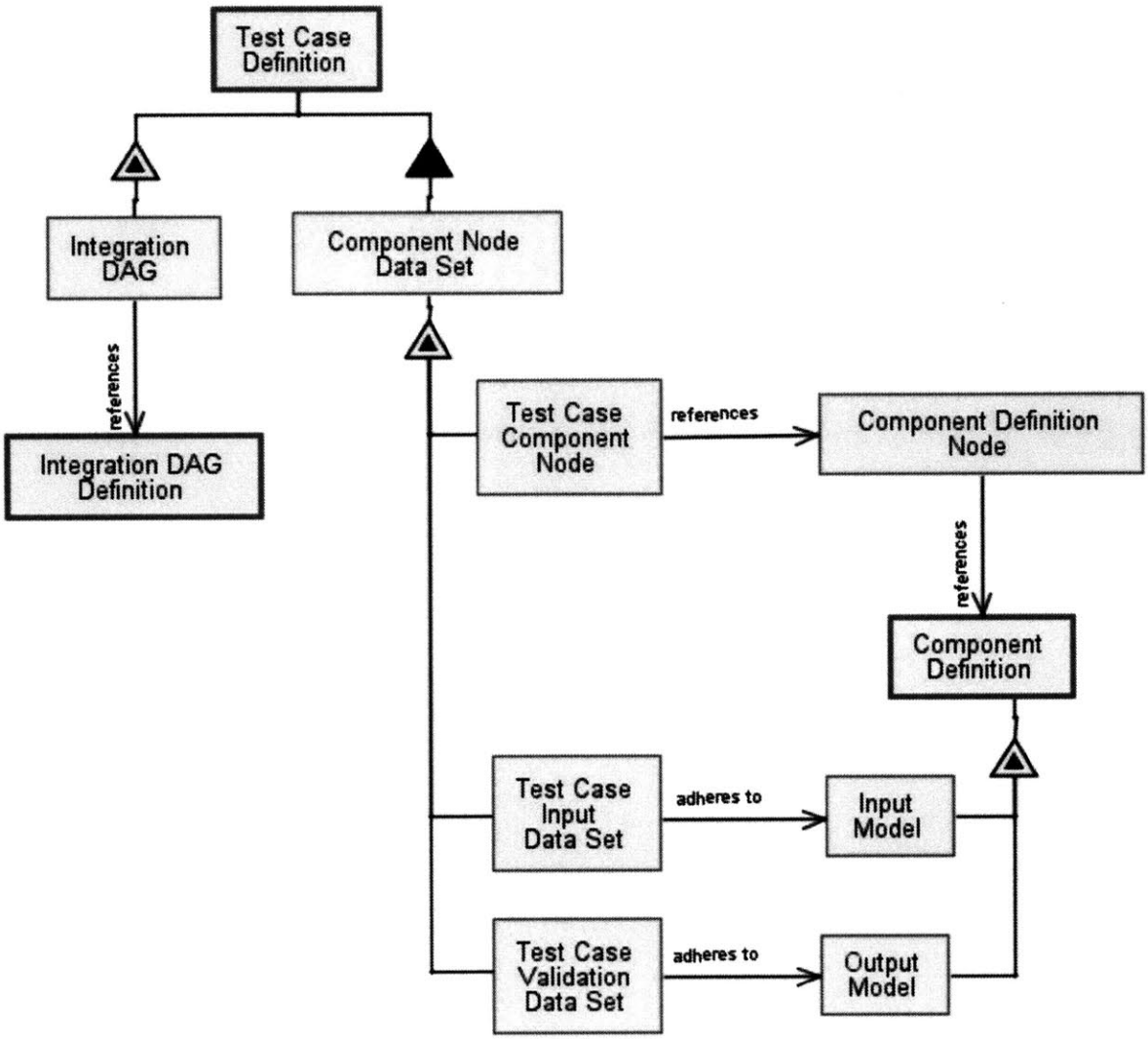


Figure 15 - Automated Unit Testing - Test Case Definition unfolded

Component Definition **exhibits** Input Model **and** Output Model.
Component Definition Node **references** Component Definition.
Test Case Definition **exhibits** Integration DAG.
Integration DAG **references** Integration DAG Definition.
Test Case Definition **consists of many** Component Node Data Sets.
Component Node Data Set **exhibits** Test Case Input Data Set, Test Case Validation Data Set, **and** Test Case Component Node.
Test Case Input Data Set **adheres to** Input Model.
Test Case Validation Data Set **adheres to** Output Model.
Test Case Component Node **references** Component Definition Node.

A unified toolset that covers all three patterns

Streamlining integration activities can be achieved by providing an integration toolset that covers the three patterns surveyed above for different business scenarios. The proliferation of toolsets was indicated by Stonebraker who recognized the need for consolidation early on in his article "Too Much Middleware" (2002) (Stonebraker, 2002). Bernstein & Hass discuss the plethora of tools provided to reduce the effort and cost of data integrating activities, but note that since data integrating tasks are complex and have many different scenarios, many of those tools are very specialized (2008) (Bernstein & Hass, 2008). Halevy, Rajaraman and Ordille state "As a community, our goal should be to create tools that facilitate data integration in a variety of scenarios" (2006) (Halevy, Rajaraman, & Ordille, 2006). A search of papers on the topics seems to validate that people are researching these topics in silos versus together as there are tens of thousands of articles on ETL and MOM individually but very few that reference them together.

The Data Integrating System is fundamentally message-based. An Integration DAG defines a set of Component Definition Nodes linked by Component Definition links to form a graph of readers, processors, and writers that form the basis of an integration. Data Sets are transferred between nodes of the DAG via messages as in any message oriented middleware. This results in the Data Integrating System being highly capable of handling message-oriented scenarios. However, an ETL process can also be designed and defined as a graph. The Integrating System Supports several types of special command messages, including Startup Messages and Shutdown Messages, as depicted in Figure 16.

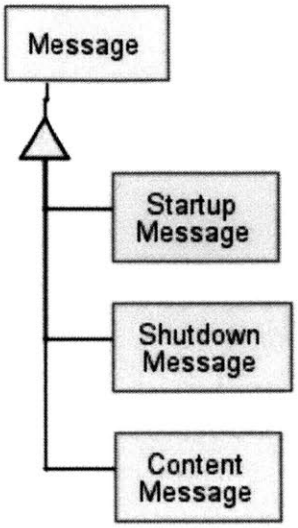


Figure 16 - Unified Toolset - Message unfolded

When an Integration DAG is executed via the Integration Executing process, the following occurs:

- Component Nodes are created for every Component Definition Node defined in the Integration DAG Definition
- Each of the Component Nodes created above is started in a separate executable thread
- A Startup Message is sent to any Component Node in the DAG that does not have an inbound Component Definition Link. This Startup Message will begin processing of the Components in the DAG as depicted in Figure 17.

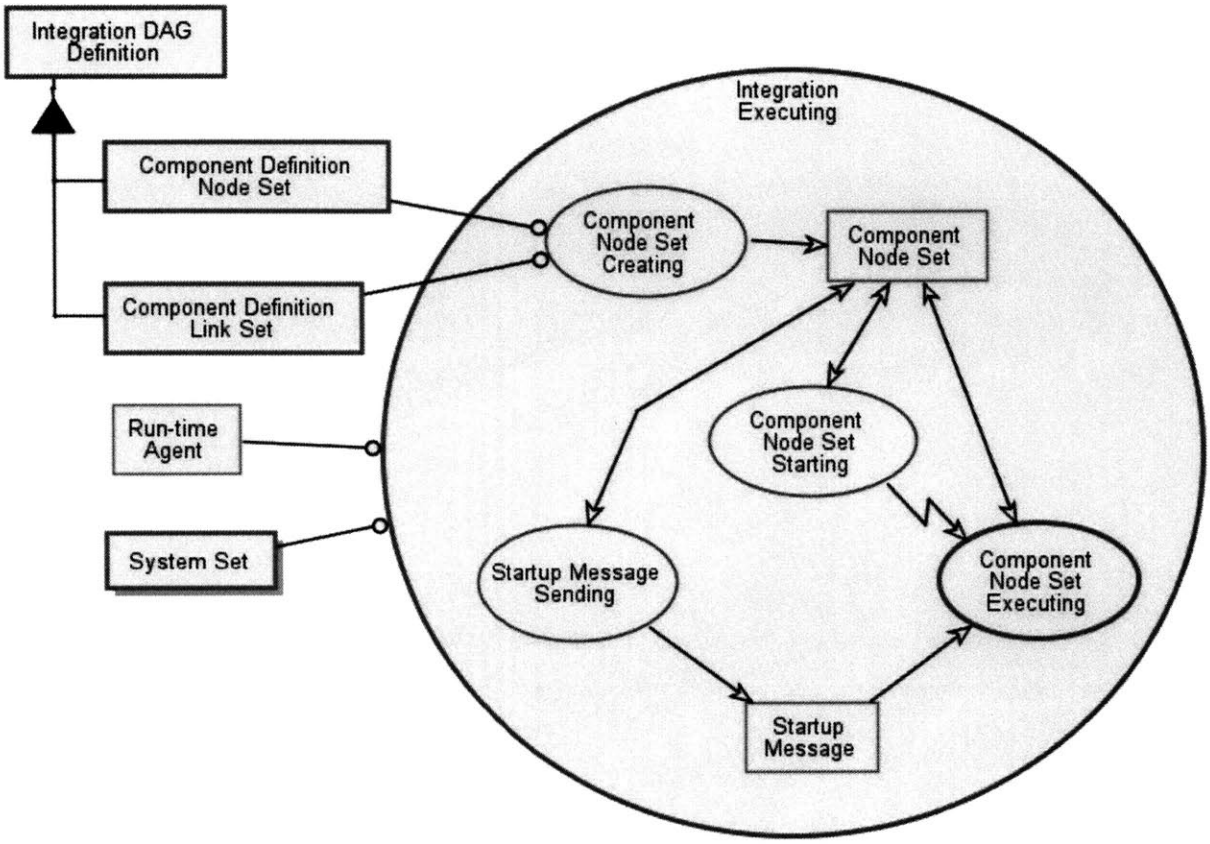


Figure 17 - Unified Toolset - Integration Executing in-zoomed

System Set **is physical.**
Integration DAG Definition **consists of** Component Definition Node Set **and** Component Definition Link Set.
Integration Executing **exhibits** Component Node Set **and** Message.
Integration Executing **consists of** Component Node Set Creating, Component Node Set Starting, Startup Message Sending, **and** Component Node Set Executing.
Integration Executing **requires** System Set **and** Run-time Agent.
Integration Executing **zooms into** Component Node Set Creating, Component Node Set Starting, Component Node Set Executing, **and** Startup Message Sending, **as well as** Component Node Set **and** Message.

Component Node Set Creating **requires** Component Definition Link Set **and** Component Definition Node Set.

Component Node Set Creating **yields** Component Node Set.
Component Node Set Starting **affects** Component Node Set.
Component Node Set Starting **invokes** Component Node Set Executing.
Component Node Set Executing **affects** Component Node Set.
Component Node Set Executing **consumes** Message.
Startup Message Sending **affects** Component Node Set.
Startup Message Sending **yields** Message.

In an ETL or ELT scenario which may consist of a Reader (extract), Transformer and Writer (load), the Reader component is kicked off by a Startup Message, reads data from a Data Set, sends that data in messages to its linked Component Nodes, then sends a Shutdown Message to its linked Component Nodes and shuts itself down. Once the Transformer processes all inbound message data and sends the transformed data to its linked Component Nodes, it receives the Shutdown message from the Reader Component and shuts itself down. The Writer Component does the same. This results in an Integration DAG that executes very similarly to an ETL or ELT job that begins and ends with a specific set of tasks being completed.

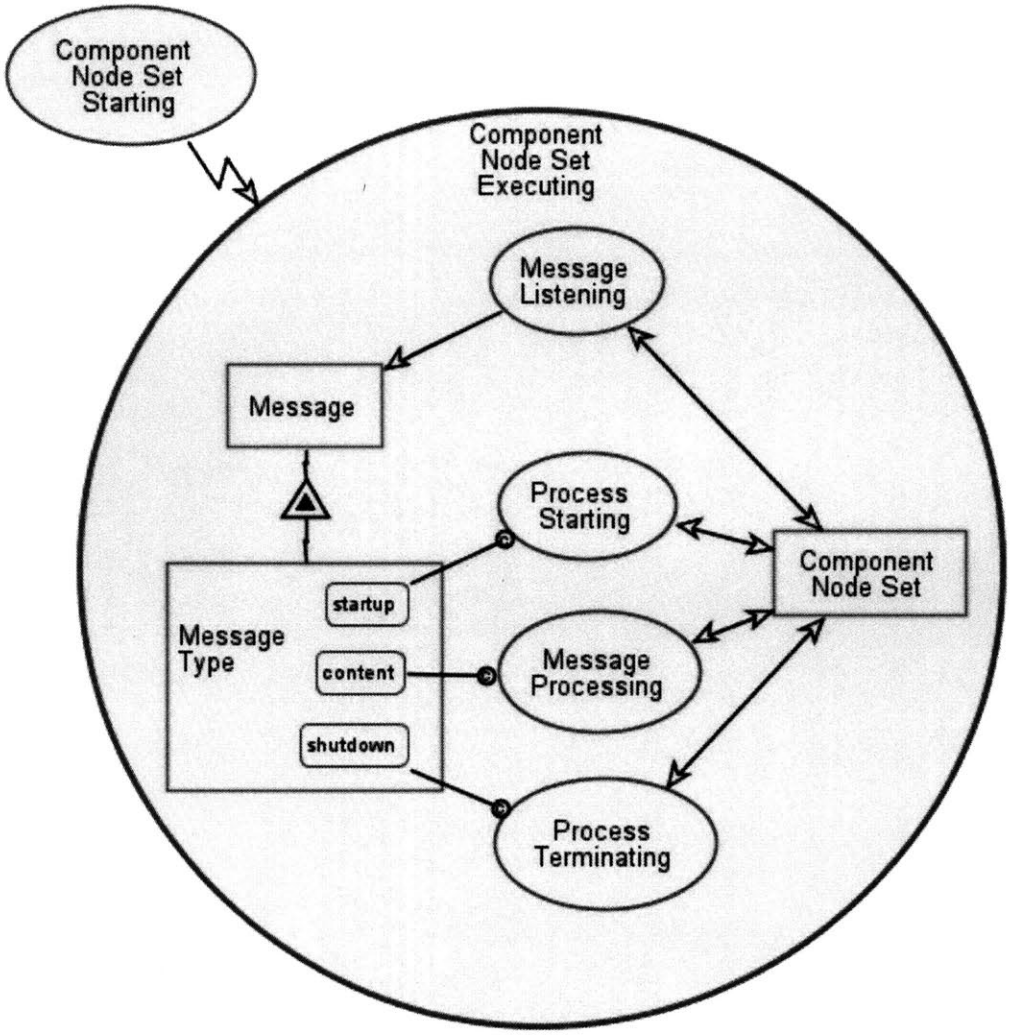


Figure 18 - Unified Toolset - Component Node Set Executing in-zoomed

Web-based Toolset

Integration can also be achieved by providing a Web-based integration platform that works over standard Internet protocols to allow configuration, management and

execution of integrations in the cloud over standard web protocols. Most legacy integration platforms continue to use thick clients for configuration and management of integration jobs, and few have seamless remote agent capabilities that allow interaction of agents over web protocols. While much research calls for using Web Services to facilitate this type of integration (Pahl & Zhu, 2012), strict service-based integration is often not feasible when COTS software is used if that package and vendor do not provide a service base as an option.

The OPM model presented next defines an integration platform that attempts to address some of these needs. The model is in its early stages and will most likely evolve over the next several years. Yet, it has been the driving force of design discussions around a better integration toolset for the future in a large scale data integration project. This project entails creation of a master data management (mdm) solution and re-architecture of all inbound and outbound integrations to and from the mdm system to all other operational systems within the enterprise. In all, hundreds of integrations across systems will be based on the integrating architecture and system defined.

Data Integrating Model

This model describes a Data Integrating System that integrates Data Sets between Systems in a System Set. As part of the Data Integrating process, a User Group is kept informed about the data that is integrated and the status of the Data Integrating process.

System Diagram

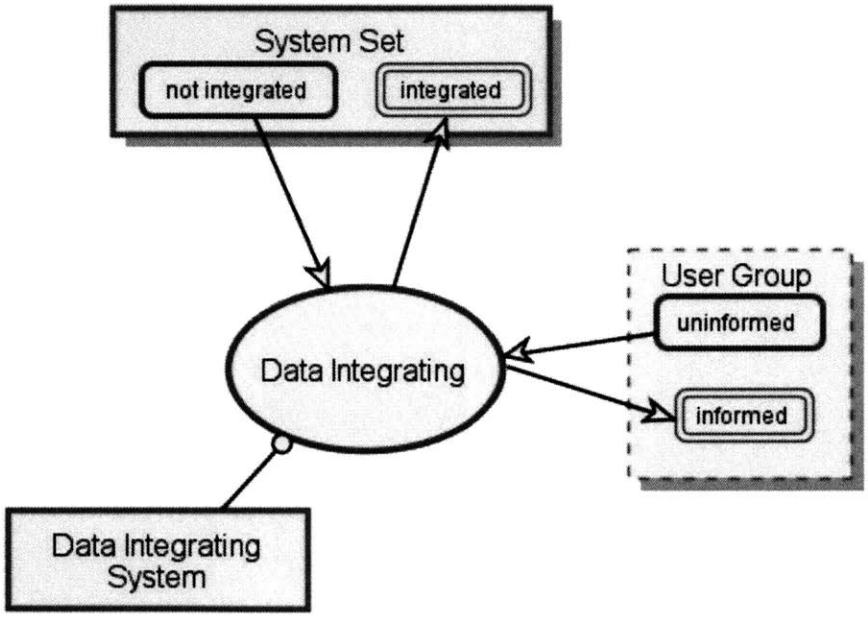


Figure 19 - Data Integrating - System Diagram

System Set **is physical.**
System Set **can be** not integrated **or** integrated.
    not integrated **is initial.**
    integrated **is final.**
User Group **is environmental and physical.**
User Group **can be** uninformed **or** informed.
    uninformed **is initial.**
    informed **is final.**
Data integrating **requires** Data Integrating System.
Data integrating **changes** User Group **from** uninformed **to** informed **and** System Set **from** not integrated **to** integrated.

The System Diagram specifies that the process of Data Integrating changes a s et of systems—the System Set—from not being integrated to being integrated. This is done by the Data Integrating System, which is the instrument for this process, and the User Group is informed about the process.

System Set unfolded

The System Set consists of one or more software Systems, each containing a Data Storage & Retrieval System that hosts a Data Set. The Data Storage and Retrieval System may be as simple as a File System which contains flat files, or may be more complex such as a Database Management System. Note that Database Management System doesn't restrict to relational database management systems, but may be a non-relational store as well such as a NoSQL database, column store, or other.

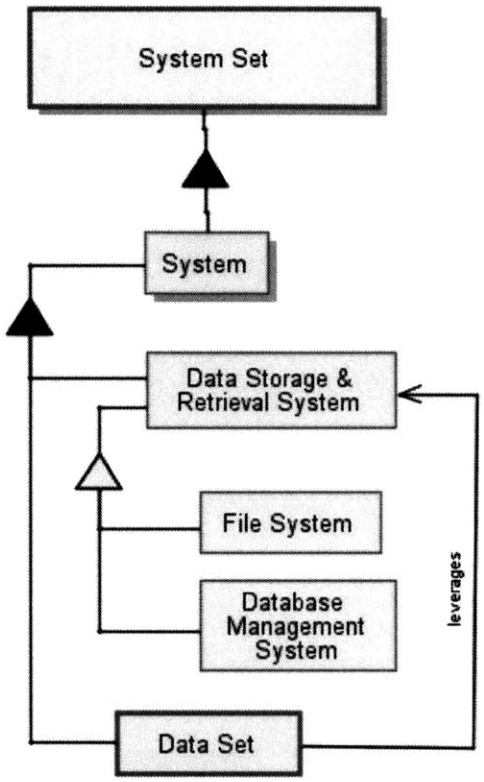


Figure 20 - System Set unfolded

System Set **is physical.**
System Set **consists of many** Systems.
System **is physical.**
System **consists of** Data Storage & Retrieval System **and** Data Set.
Data Set **leverages** Data Storage & Retrieval System.
File System **is a** Data Storage & Retrieval System.
Database Management System **is a** Data Storage & Retrieval System.

Data Set unfolded

A Data Set can be either Structured or Unstructured. Some integrations may be tasked with moving unstructured binary data from one location to another, while others may be tasked with moving all, or parts of, structured data sets. Many existing toolsets allow for only the movement of one type or the other. By consciously defining different types of Data Sets, we can ensure the Data Integrating System allows for movement of both types of data.

If the Data Set is structured, it is defined by a Data Model and Format. The Data Model describes entities an attributes of the data while the Format describes the layout of the data within the Data Set itself.

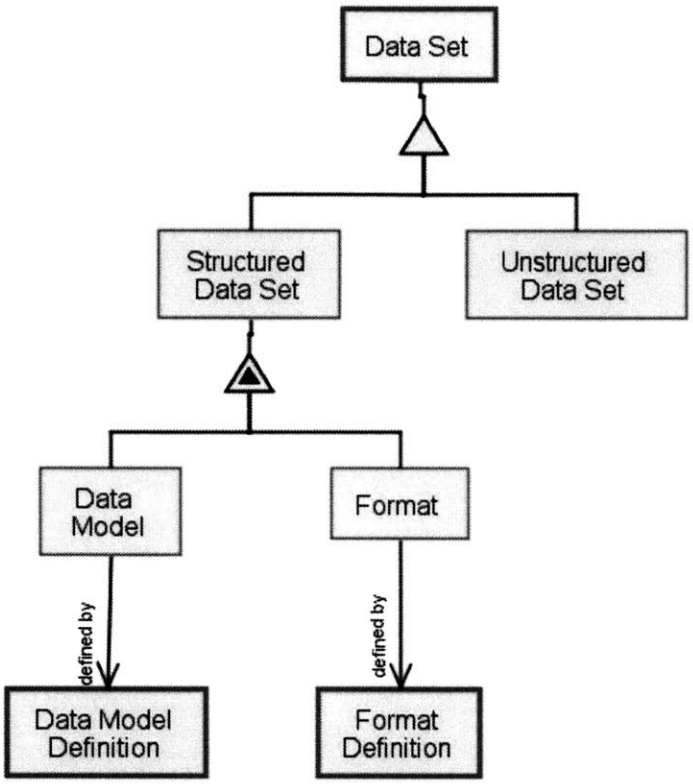


Figure 21 - Data Set unfolded

Data Integrating System unfolded

The Data Integrating System is comprised of four main components, the Integration Component Repository, the Configuration / Operational Repository, the Graphical User Interface and the Run-time Agent.

- **Integration Component Repository** is a code-based repository that contains Integration Components the User Group uses to define an integration. Sample components include things like Connection Components, Database Reader and Writer Components, File Reader and Writer Components, Translator Components, etc. See Integration Component Repository unfolded for details. For clarity on naming conventions, all things within the Integration Component Repository have names ending in "Component."

- **Configuration / Operational Repository** is a database management system that will be used to store component definitions that are defined by the User Group. Component definitions are configured instances of the components available in the Integration Component Repository. As an example, a Database Connection Component is utilized by the User Group to define a Connection Definition which is stored in the Configuration / Operational Repository. The Connection Definition contains the details about a specific connection to a System.

- **Graphical User Interface** is the interface between the User Group and the System which allows creation of Component and Integration DAG Definitions.

- **Run-time Agent** is a code-based agent that will be used to run or execute integrations that are defined by the User Group. It is utilized in the Integration Executing process.

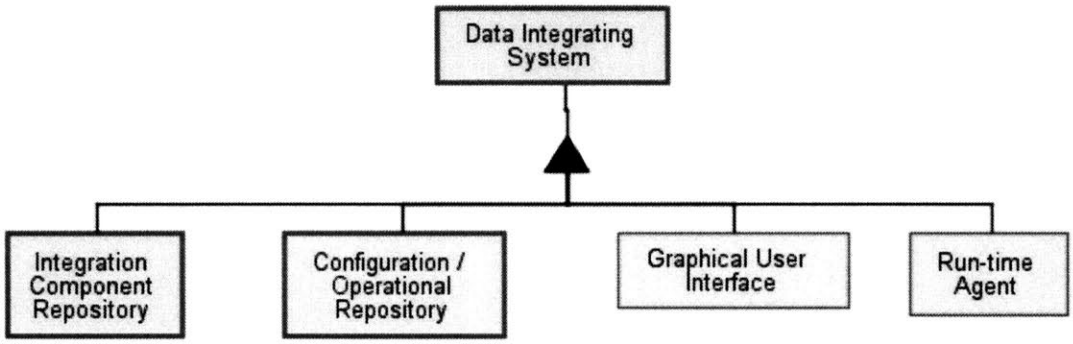


Figure 22 - Data Integrating System unfolded

Data Integrating System **consists of** Integration Component Repository, Configuration / Operational Repository, Run-time Agent, **and** Graphical User Interface.

Data Integrating in-zoomed

The data integrating process is broken down into four major sub-processes, Integration Defining, Integration Test Case Defining & Executing, Integration Executing, and Integration Monitoring. The defining process allows a User in the User Group to define the integration. The definition of the integration is stored within the Configuration / Operational Repository for use by the subsequent processes. The Integration Test Case Defining & Executing process allows the user to define a set of integration unit test scenarios and the associated data sets that go along with them. It then allows execution of the test case scenarios on a repeated basis for automation of unit testing. The Integration Executing process allows the defined integration to be run or executed, and the Integration Monitoring process allows a User in the User Group to monitor the status of an executing or executed integration.

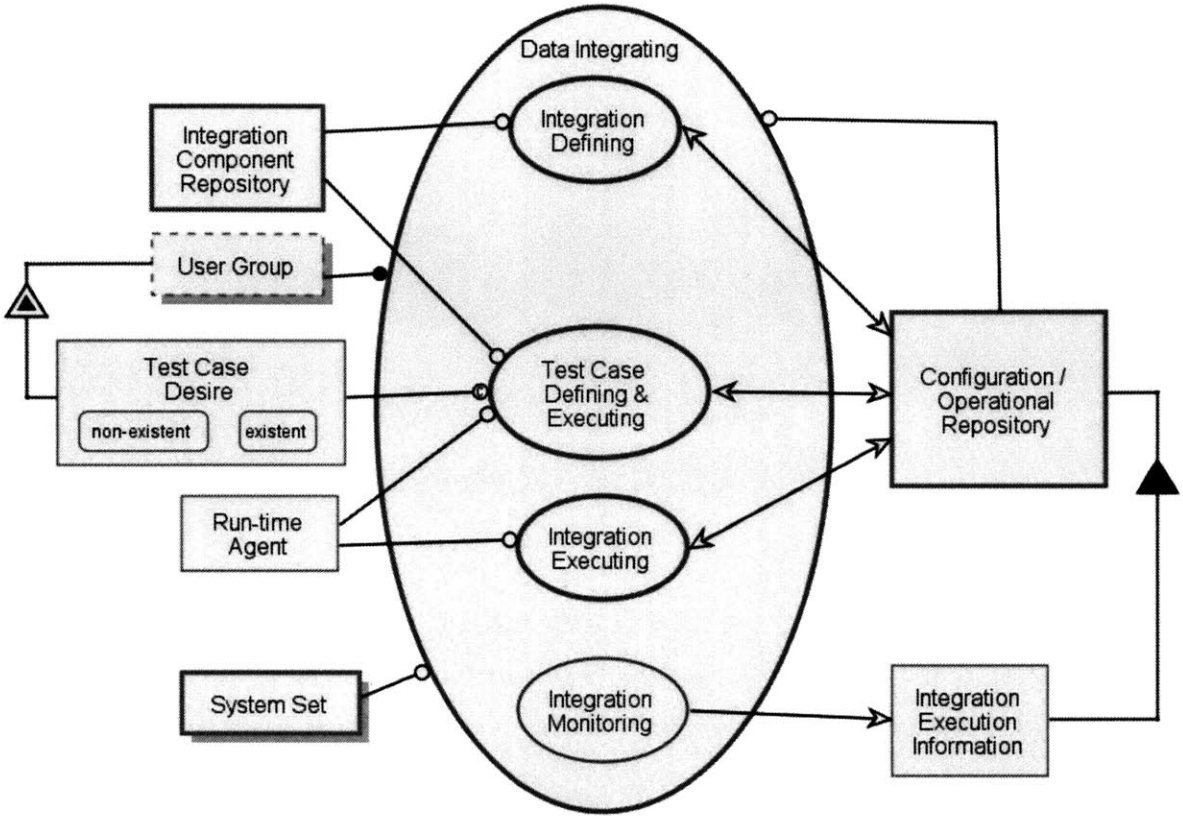


Figure 23 - Data Integrating in-zoomed

System Set **is physical.**
User Group **is environmental and physical.**
User Group **exhibits** Test Case Desire.
Test Case Desire **can be** non-existent **or** existent.
User Group **handles** Data Integrating.
Configuration / Operational Repository **consists of** Integration Execution Information.
Data Integrating **consists of** Integration Defining, Integration Executing, Integration Monitoring, **and** Test Case Defining & Executing.
Data Integrating **requires** Configuration / Operational Repository **and** System Set.
Data Integrating **zooms into** Integration Defining, Test Case Defining & Executing, Integration Executing, **and** Integration Monitoring.
Integration Defining **requires** Integration Component Repository.
Integration Defining **affects** Configuration / Operational Repository.
Test Case Defining & Executing **occurs if** Test Case Desire **is** in existent.
Test Case Defining & Executing **requires** Run-time Agent **and** Integration Component Repository.
Test Case Defining & Executing **affects** Configuration / Operational Repository.
Integration Executing **requires** Run-time Agent.
Integration Executing **affects** Configuration / Operational Repository.
Integration Monitoring **yields** Integration Execution Information.

Integration Component Repository unfolded

Components in the Integration Component Repository are grouped into three categories, Connection Component Set, Data Model Component Set and Integration DAG Component Set.

- **Connection Components** are used to define Connection Definitions to Data Sets that need to be read or written as part of the integration. Note that a Connection Definition is generic in nature. It can be used to define the connection information for a file system, a relational database, or a NoSql data store. The separate and distinction of Connection Components from other DAG components is described in more detail in the Connection Set Defining in-zoomed process.

- **Data Model Components** are used to define the structure of a structured data set.

- **Integration DAG Component Set** – When an integration is defined in the Data Integrating System, it is defined as a directed acyclic graph (DAG) of Components. As described previously, those Integration Components can be Readers connected to Translators connected to Writers, or any other combination of components assembled in a graph of processing elements. The Integration DAG Component Set consists of all Components that can be used within the Integration DAG.

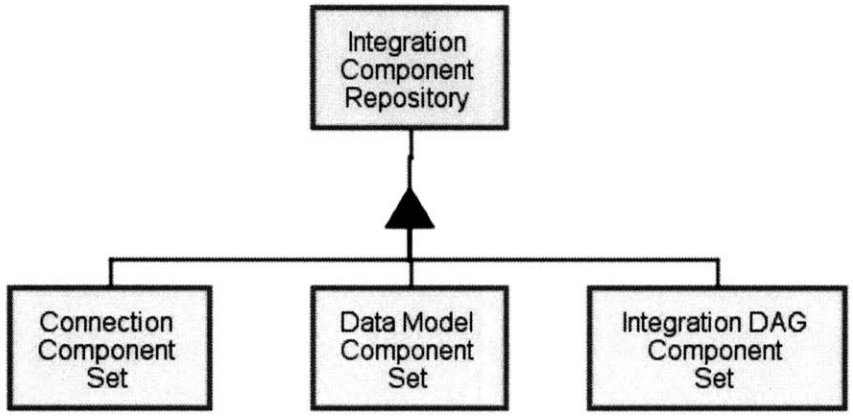


Figure 24 - Integration Component Repository unfolded

Integration Component Repository consists of Integration DAG Component Set, Connection Component Set, and Data Model Component Set.

Configuration / Operational Repository unfolded

The Configuration / Operational Repository is the data store which houses all of the Connection, Data Model, Integration DAG and Test Case Definitions created within the Integration Defining process. It also stores Integration Execution information from the Integrating System.

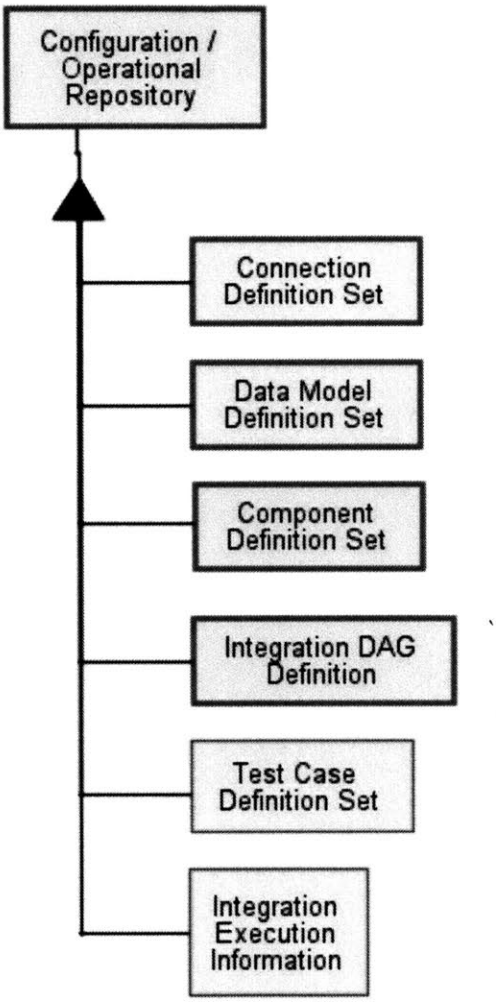


Figure 25 - Configuration / Operational Repository unfolded

Configuration / Operational Repository **consists of** Integration DAG Definition, Connection Definition Set, Data Model Definition Set, Integration Execution Information, Component Definition Set, **and** Test Case Definition Set.

Integration Defining in-zoomed

The integration defining process allows a user in the User Group to define an integration. An integration is created by defining a Connection Definition Set, an optional Data Model Definition Set and an Integration DAG Definition.

- **Connection Definition Set** – The Connection Definition Set consists of a set of Connection Definitions that will be used for the integration. These Connection Definitions are used in conjunction with other Integration DAG Definitions to provide access to a Data Set that resides on a System. An integration can have many connections associated with it.

- **Data Model Set Defining** – Data Model Set Defining is an optional step in the Integration Defining process. A Data Model Definition is created at this point if there are common Data Model Definitions that the User would like to use in this, and other integrations. A common integration approach between disparate systems is often to create a common canonical model or global ontology between the systems. This global ontology serves as a common data model definition between the systems. In a global ontology pattern, each system exchanging data maps its internal data model or local ontology to the global ontology and exchanges data with other systems through that global ontology. No system maps its local ontology directly to another system's local ontology. By taking this approach, each system only needs to map its local ontology once, to the global ontology, regardless of the number of systems to which the system is sending data (Zhang, A Query Driven Method of Mapping from Global Ontology to Local Ontology in Ontology-based Data Integration, 2014). Not every integration should be forced down a local/global ontology approach. For integrations where only a local ontology is desired, the local ontology can be created at during the Integration DAG Defining process as part of one or more Component Definitions used for reading data.

- **Integration DAG Defining** – Integration DAG Defining is the process by which the individual Integration DAG Definitions (i.e. readers, translators, writers, etc.) are defined and connected together to form an integration flow. The input of that flow is one or more Data Sets from the source systems, and the output of the flow is one or more Data Sets to the target systems resulting in a Data Set being integrated. Note the Integration DAG Defining creates the definition of the flow; it does not execute the flow.

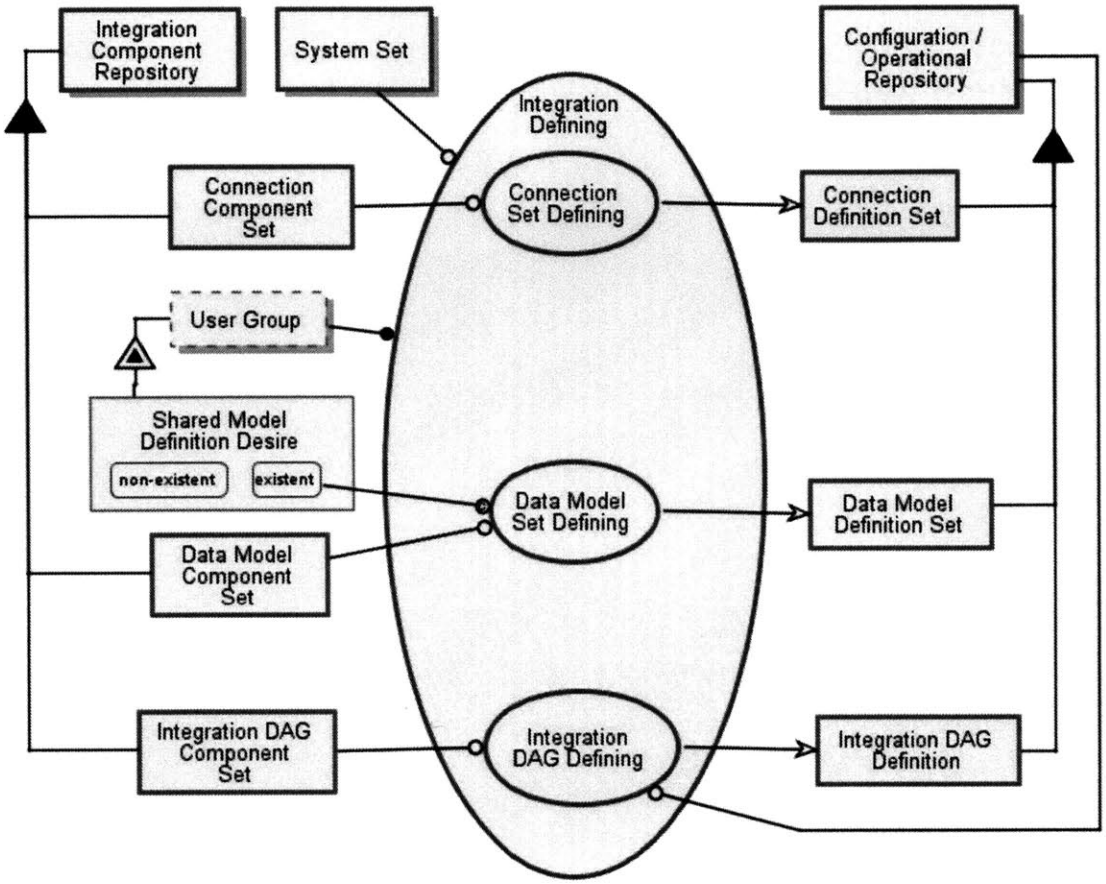


Figure 26 - Integration Defining in-zoomed

System Set **is physical.**
User Group **is environmental and physical.**
User Group **exhibits** Shared Model Definition Desire.
Shared Model Definition Desire **can be** existent **or** non-existent.
User Group **handles** Integration Defining.
Integration Component Repository **is physical.**
Integration Component Repository **consists of** Integration DAG Component Set, Connection Component Set, **and** Data Model Component Set.
Configuration / Operational Repository **is physical.**
Configuration / Operational Repository **consists of** Integration DAG Definition, Connection Definition Set, **and** Data Model Definition Set.
Integration Defining **consists of** Integration DAG Defining, Connection Set Defining, **and** Data Model Set Defining.
Integration Defining **requires** System Set.
Integration Defining **zooms into** Connection Set Defining, Data Model Set Defining, **and** Integration DAG Defining.
Connection Set Defining **requires** Connection Component Set.
Connection Set Defining **yields** Connection Definition Set.
Data Model Set Defining **occurs if** Shared Model Definition Desire **is** existent.
Data Model Set Defining **requires** Data Model Component Set.

Data Model Set Defining **yields** Data Model Definition Set.
Integration DAG Defining **requires** Configuration / Operational Repository **and** Integration DAG Component Set.
Integration DAG Defining **yields** Integration DAG Definition.

Connection Set Defining in-zoomed

A connection definition defines a connection to a Data Set on a System. A conscious decision was made to separate the definition of a Connection from the Integration Components within the Integration DAG itself. Most integration toolsets combine the definition of the connection with the component that uses that connection. For example, if you are defining a file reader or a database reader, most tools force you to define the connection information in combination with the database or individual file being accessed as part of that component. Separating the Connection Definition from the Component Definition allows Connection Definitions to be reused across a set of components. It also allows for easier maintenance of changes to the connection information

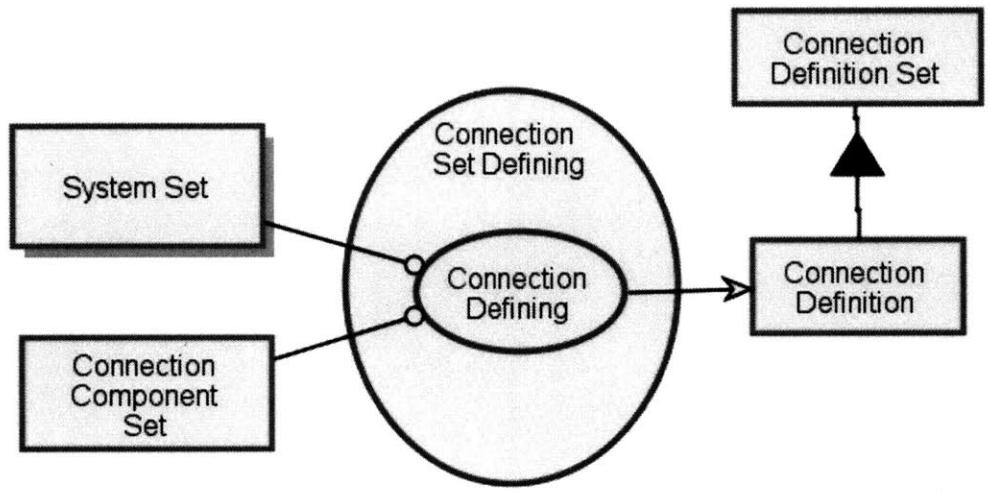


Figure 27 - Connection Set Defining in-zoomed

System Set **is physical.**
Connection Definition Set **consists of many** Connection Definitions.
Connection Set Defining **zooms into** Connection Defining.
Connection Defining **requires** Connection Component Set **and** System Set.
Connection Defining **yields** Connection Definition.

Connection Component Set unfolded

The Connection Component Set consists of a variety of types of Component Connections that can be used to create Connection Definitions. From databases to direct and network attached storage to FTP, queues and other, the Connection Component Set allows for creation of any connection used in conjunction with other Integration DAG Definitions.

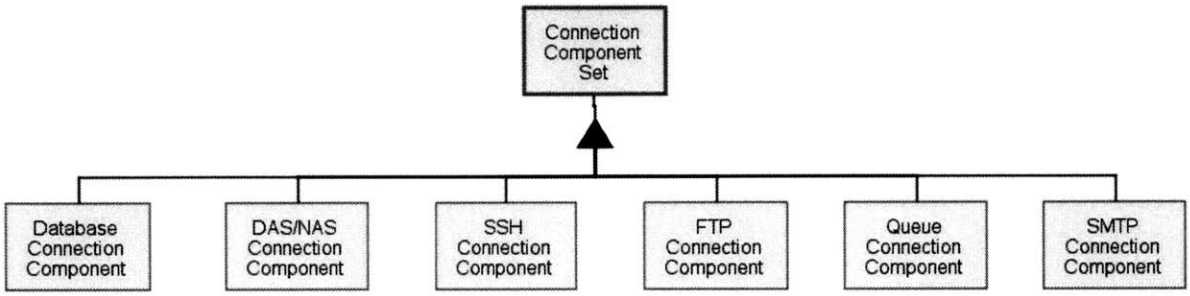


Figure 28 - Connection Component Set unfolded

Connection Component Set consists of Database Connection Component, DAS/NAS Connection Component, SSH Connection Component, Queue Connection Component, SMTP Connection Component, and FTP Connection Component.

Connection Definition unfolded

A Connection Definition has the following elements:

- **Folder** – A Connection Definition is organized within a Folder. This Folder structure is used in the user interface to allow a User in the User Group to categorize and organize Connection Definitions in any way they see fit.

- **Connection Type** – The type of connection this Connection Definition represents. See Connection Type unfolded for additional details.

- **Connection Name** – A user defined name for this connection

- **Connection Definition Setting Set** – Different attributes must be defined and stored for each Connection Type. For example, a Database Connection Definition needs to store information about the server on which the database resides, the TCP/IP port on which the database is listening, a username and password, etc. For a DAS/NAS connection, the attributes that must be stored are quite different. Instead of hardcoding the model for these different attributes, a generic Connection Definition Setting Set is used. The Setting Set is a set of developer defined key/value pairs that go along with each Connection Type. For example key = database_port, value = 3307. Utilizing a generic mechanism for unique settings for each Connection Type allows for greater flexibility in creating new

Connection Types in the future without modifying the underlying structure of how a Connection Definition is stored within the Data Integrating System. This would also allow easier extension by other third parties including other vendors or users in an open source community setting.

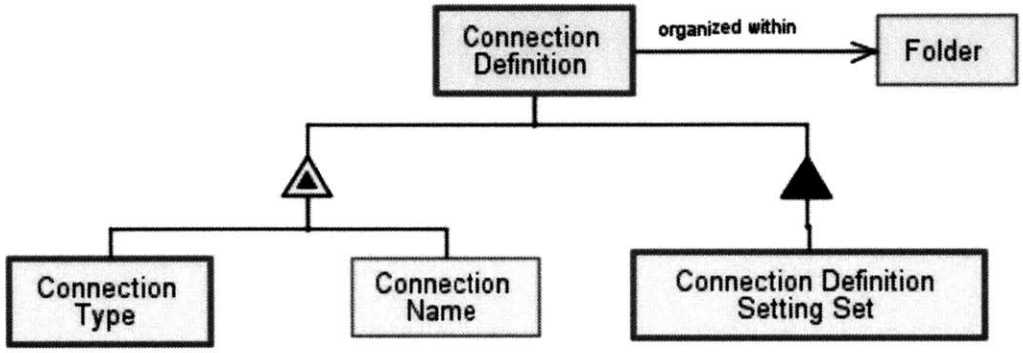


Figure 29 - Connection Definition unfolded

Connection Definition **exhibits** Connection Type **and** Connection Name.
Connection Definition **consists of** Connection Definition Setting Set.
Connection Definition **organized within** Folder.

Connection Type unfolded

There are currently five Connection Types defined. As discussed earlier, this will be extended greatly over time both by the original creators of the Data Integrating System as well as by other third parties.

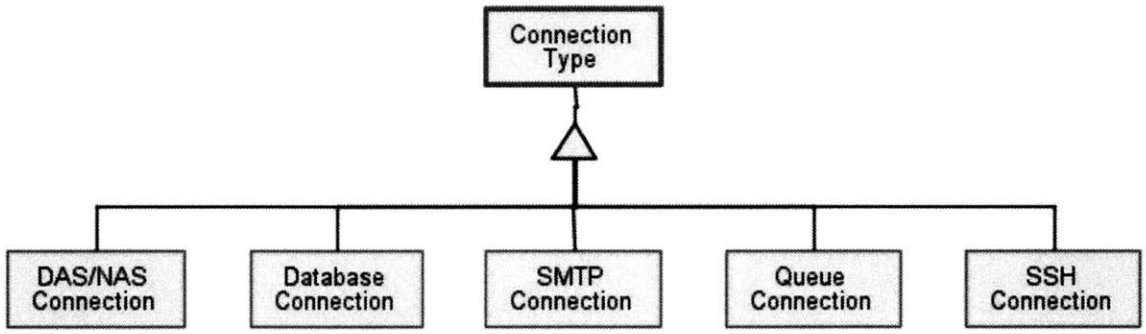


Figure 30 - Connection Type unfolded

Database Connection **is a** Connection Type.
DAS/NAS Connection **is a** Connection Type.
SSH Connection **is a** Connection Type.

Queue Connection **is a** Connection Type.
SMTP Connection **is a** Connection Type.

Connection Definition Setting Set unfolded

The Connection Definition Setting Set can currently be of one of the types described below. Each type of Connection Definition Setting Set has different key/value pairs associated with it.

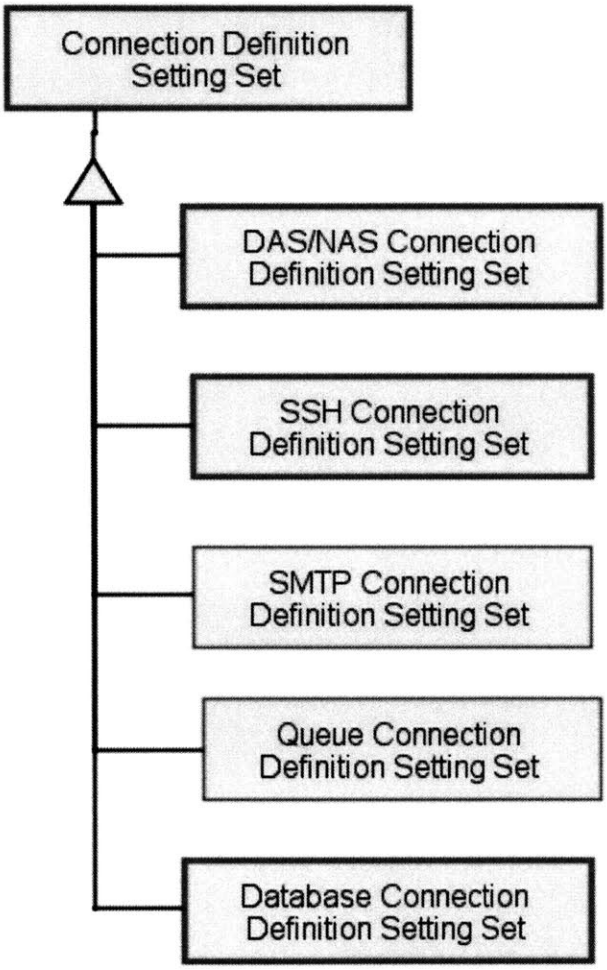


Figure 31 - Connection Definition Setting Set unfolded

Database Connection Definition Setting Set **is a** Connection Definition Setting Set.
DAS/NAS Connection Definition Setting Set **is a** Connection Definition Setting Set.
Queue Connection Definition Setting Set **is a** Connection Definition Setting Set.
SMTP Connection Definition Setting Set **is a** Connection Definition Setting Set.
SSH Connection Definition Setting Set **is a** Connection Definition Setting Set.

Connection Defining in-zoomed

The Connection Defining process allows each attribute of the Connection Definition to be set by the User.

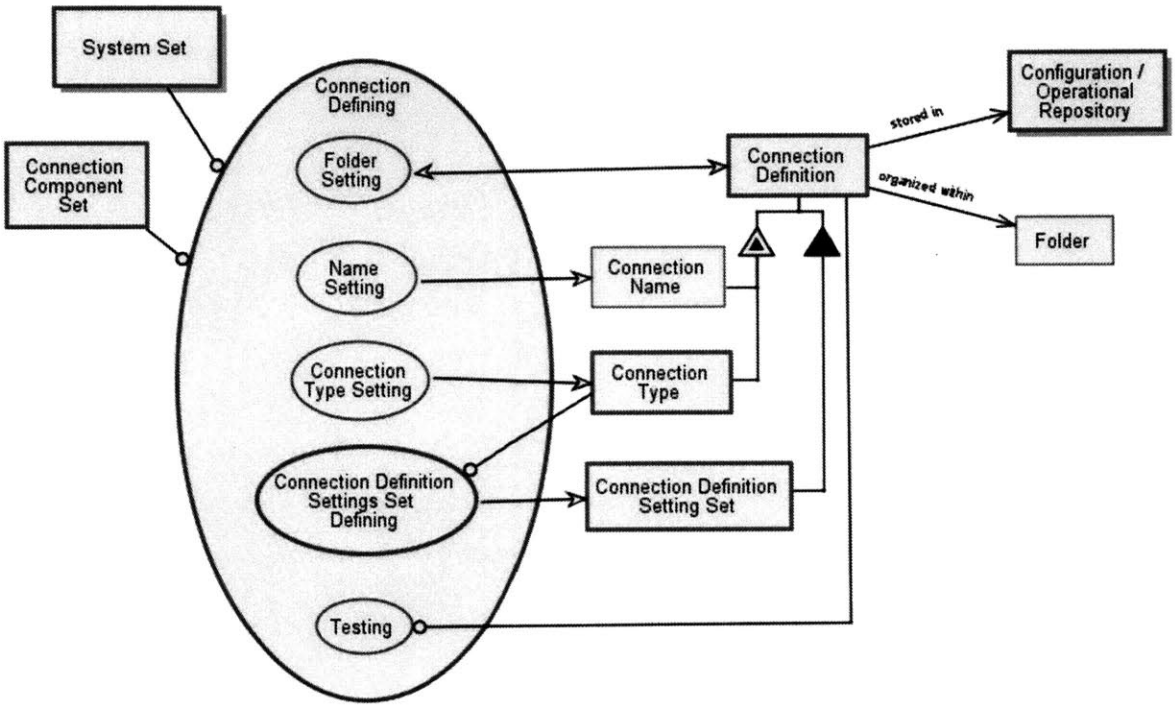


Figure 32 - Connection Defining in-zoomed

System Set **is physical.**
Configuration / Operational Repository **is physical.**
Connection Definition **exhibits** Connection Type **and** Connection Name.
Connection Definition **consists of** Connection Definition Setting Set.
Connection Definition **stored in** Configuration / Operational Repository.
Connection Definition **organized within** Folder.
Connection Defining **consists of** Connection Type Setting, Connection Definition Settings Set Defining, Testing, Name Setting, **and** Folder Setting.
Connection Defining **requires** Connection Component Set **and** System Set.
Connection Defining **zooms into** Folder Setting, Name Setting, Connection Type Setting, Connection Definition Settings Set Defining, **and** Testing.

Folder Setting **affects** Connection Definition.
Name Setting **yields** Connection Name.
Connection Type Setting **yields** Connection Type.
Connection Definition Settings Set Defining **requires** Connection Type.
Connection Definition Settings Set Defining **yields** Connection Definition Setting Set.
Testing **requires** Connection Definition.

Connection Definition Setting Set Defining in-zoomed

As described previously, defining of the Connection Definition Setting Sets depends directly on and results in a different Connection Definition Setting Set by Connection Type.

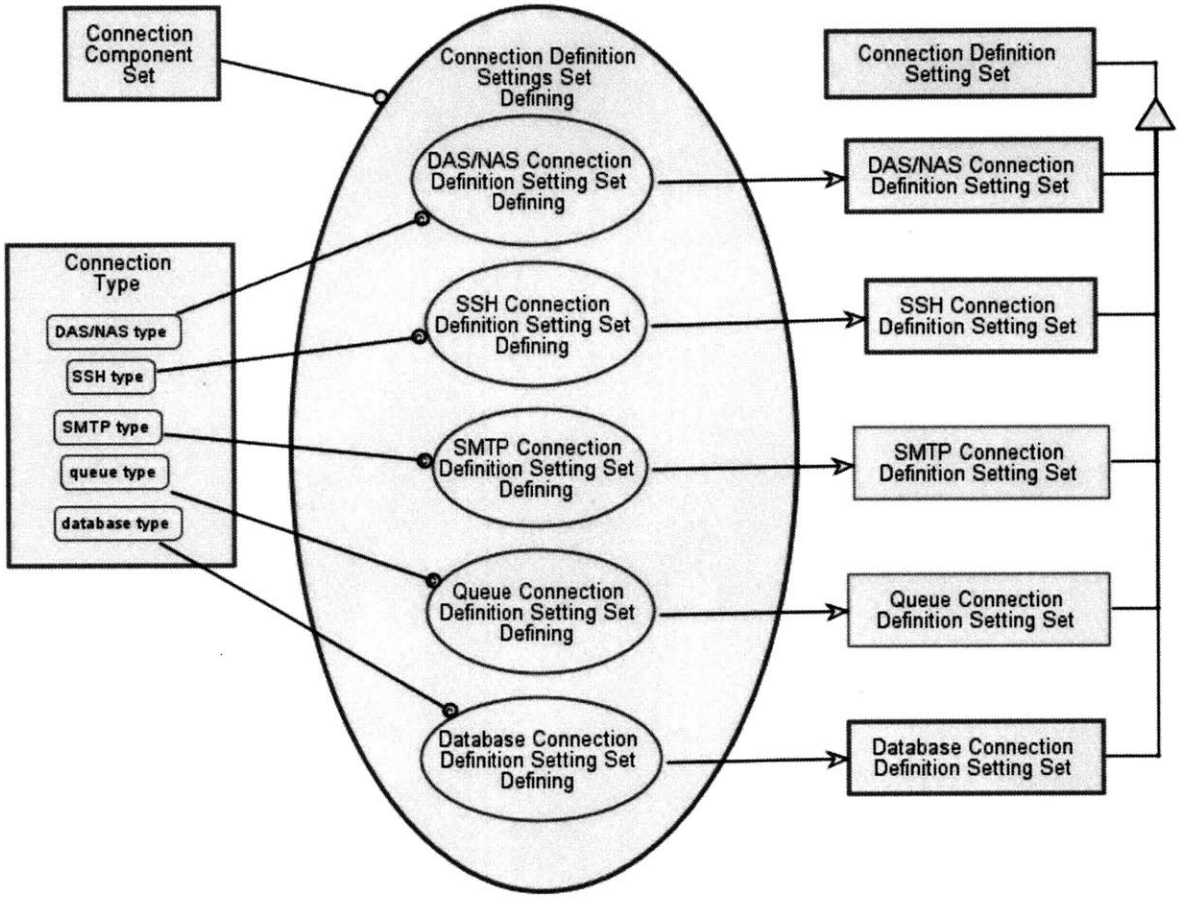


Figure 33 - Connection Definition Setting Set Defining in-zoomed

Data Integrating System **is physical.**
System Set **is physical.**
System Set **can be** not integrated **or** integrated.
    not integrated **is initial.**
    integrated **is final.**
User Group **is environmental and physical.**
User Group **can be** uninformed **or** informed.
    uninformed **is initial.**
    informed **is final.**
Data Integrating **requires** Data Integrating System.
Data Integrating **changes** User Group **from** uninformed **to** informed **and** System Set **from** not integrated **to** integrated.

DAS/NAS Connection Definition Setting Set unfolded

The Relative Path of the DAS/NAS Connection Definition Setting Set allows a root path to be set for the connection. Any Integration DAG Definition that utilizes this DAS/NAS Connection will begin looking for things starting at the Relative Path defined in this setting.

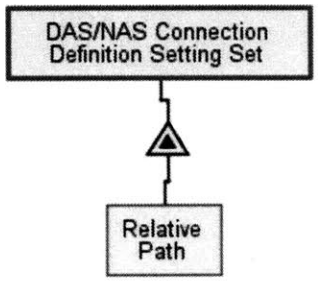


Figure 34 - DAS/NAS Connection Definition Setting Set unfolded

DAS/NAS Connection Definition Setting Set **exhibits** Relative Path.

SSH Connection Definition Setting Set unfolded

An SSH Connection Definition Setting Set is described by the following:

- **Host** – The server name or IP address on which the SSH server resides
- **Port** – The port on which the SSH server is listening for connections
- **Relative Path** – The relative path on which the connection will begin looking for things
- **User Id** – The user id with which to log into the SSH server
- **Password** – The password with which to use to log into the SSH server

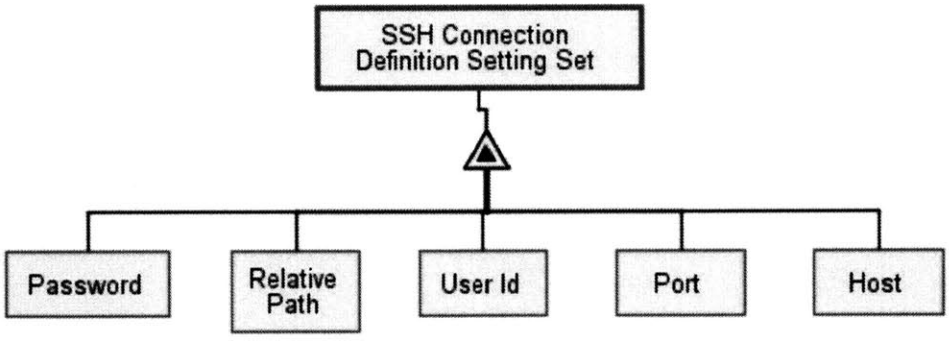


Figure 35 - SSH Connection Definition Setting Set unfolded

SSH Connection Definition Setting Set **exhibits** Port, Relative Path, User Id, Password, **and** Host.

Database Connection Definition Setting Set unfolded

A Database Connection Definition Setting Set is described by the following:

- **Host** – The server name or IP address on which the database server resides
- **Port** – The port on which the database server is listening for connections
- **Drive** – The JDBC driver used to connect to the database server
- **User Id** – The user id with which to log into the database server
- **Password** – The password with which to use to log into the database server

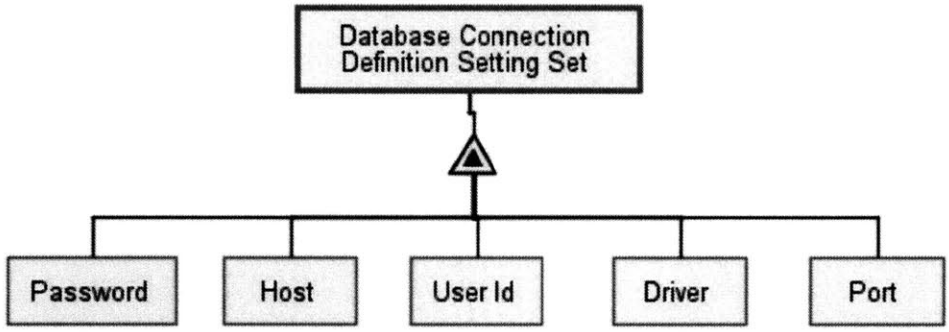


Figure 36 - Database Connection Definition Setting Set unfolded

Database Connection Definition Setting Set **exhibits** Port, Driver, User Id, Password, **and** Host.

Data Model Set Defining in-zoomed

After connections have been defined for the integration, the next step is to define optional Data Model Definitions needed for the integration. As described earlier, the Data Model Definitions can be defined globally at the integration layer or when defining an individual Integration DAG Component Definition. Regardless of definition location, the steps in defining the Data Model Definition are the same.

For each integration, a number of different Data Models can be used, thus when defining models for integration the results is a set of Data Model Definitions.

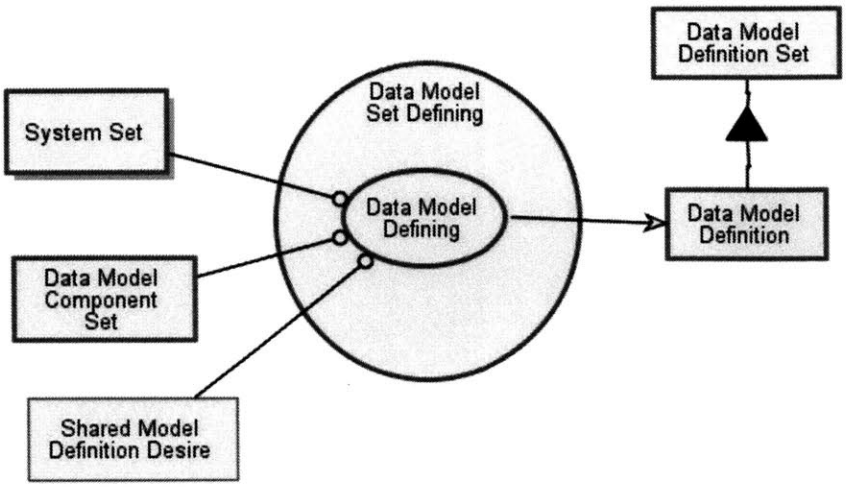


Figure 37 - Data Model Set Defining in-zoomed

System Set **is physical.**
Data Model Definition Set **consists of many** Data Model Definitions.
Data Model Set Defining **consists of** Data Model Defining.
Data Model Set Defining **zooms into** Data Model Defining.

Data Model Defining **requires** Shared Model Definition Desire, Data Model Component Set, **and** System Set.

Data Model Defining **yields** Data Model Definition.

Data Model Component Set unfolded

The Data Model Component Set consists of Components used to create Data Model Definitions. Support for three types of models are provided.

- **Relational Model** – A relational model used in a relational database
- **Hierarchical Model** – A hierarchical or tree model – commonly used in xml structures
- **Graph Model** – A graph model consisting of subject, object, and predicate, commonly used in the semantic web (2014) (Khamis, Zhong, & Gon, 2014).

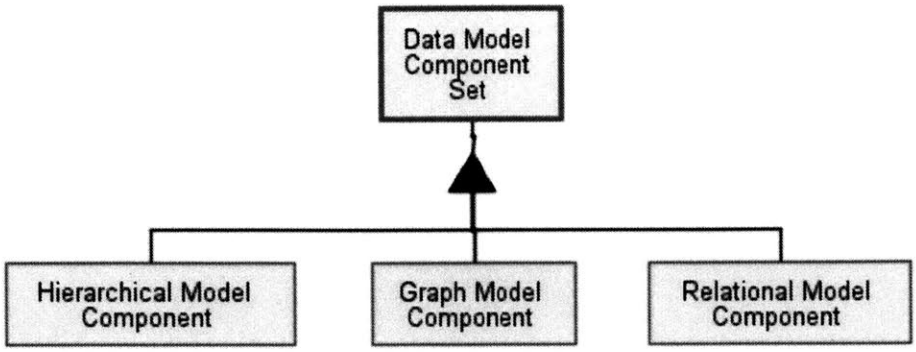


Figure 38 - Data Model Component Set unfolded

Data Model Component Set **consists of** Hierarchical Model Component, Graph Model Component, **and** Relational Model Component.

Data Model Definition unfolded

A Data Model Definition can be one of three types, a Hierarchical Model Definition, a Graph Model Definition or a Relational Model Definition. Regardless of the type, all Data Model Definitions have the following attributes:

- **Folder** – A Data Model Definition is organized within a Folder. This Folder structure is used in the user interface to allow a User in the User Group to categorize and organize Data Model Definitions in any way they see fit
- **Model Name** – A user defined name for this Data Model Definition
- **Model Type** – The type of Data Model Definition
- **Sharability** – Whether this Data Model Definition can be shared amongst other Integration DAG Definitions (sharable) or not (unsharable)
- **Format Definition** – The Data Model Definition may also reference a Format definition. See Figure 57 - Format Definition unfolded

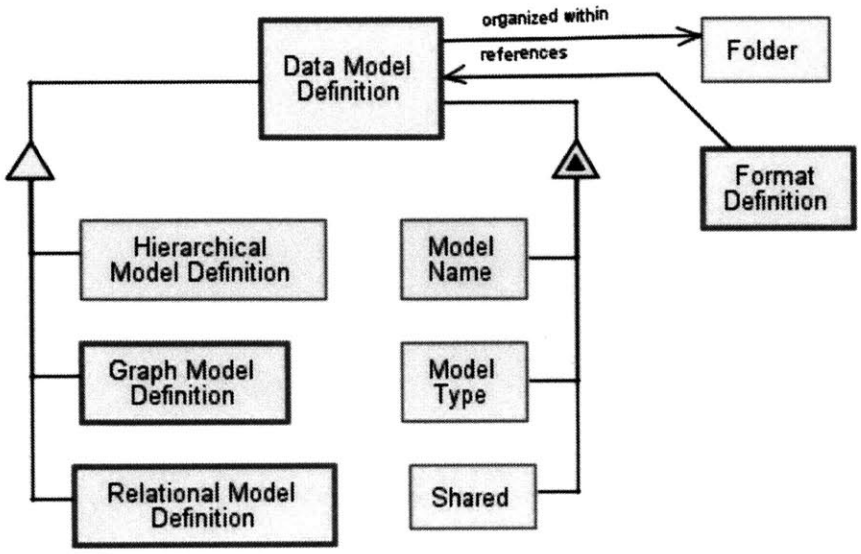


Figure 39 - Data Model Definition unfolded

Format Definition **references** Data Model Definition.
XML Attribute Setting Set **exhibits** XPath Expression.
Fixed Length Attribute Setting Set **exhibits** Start Position **and** End Position.
Delimited Attribute Setting Set **exhibits** Position.
Data Model Definition **exhibits** Model Name, Model Type, **and** Shared.
Data Model Definition **organized within** Folder.
Graph Model Definition **is a** Data Model Definition.
Relational Model Definition **is a** Data Model Definition.
Hierarchical Model Definition **is a** Data Model Definition.

Data Model Defining in-zoomed

The Data Model Defining process describes the steps in creating a Data Model Definition. This process is utilized from within the Integration Defining process as well as other processes such as Reader Defining and Writer Defining.

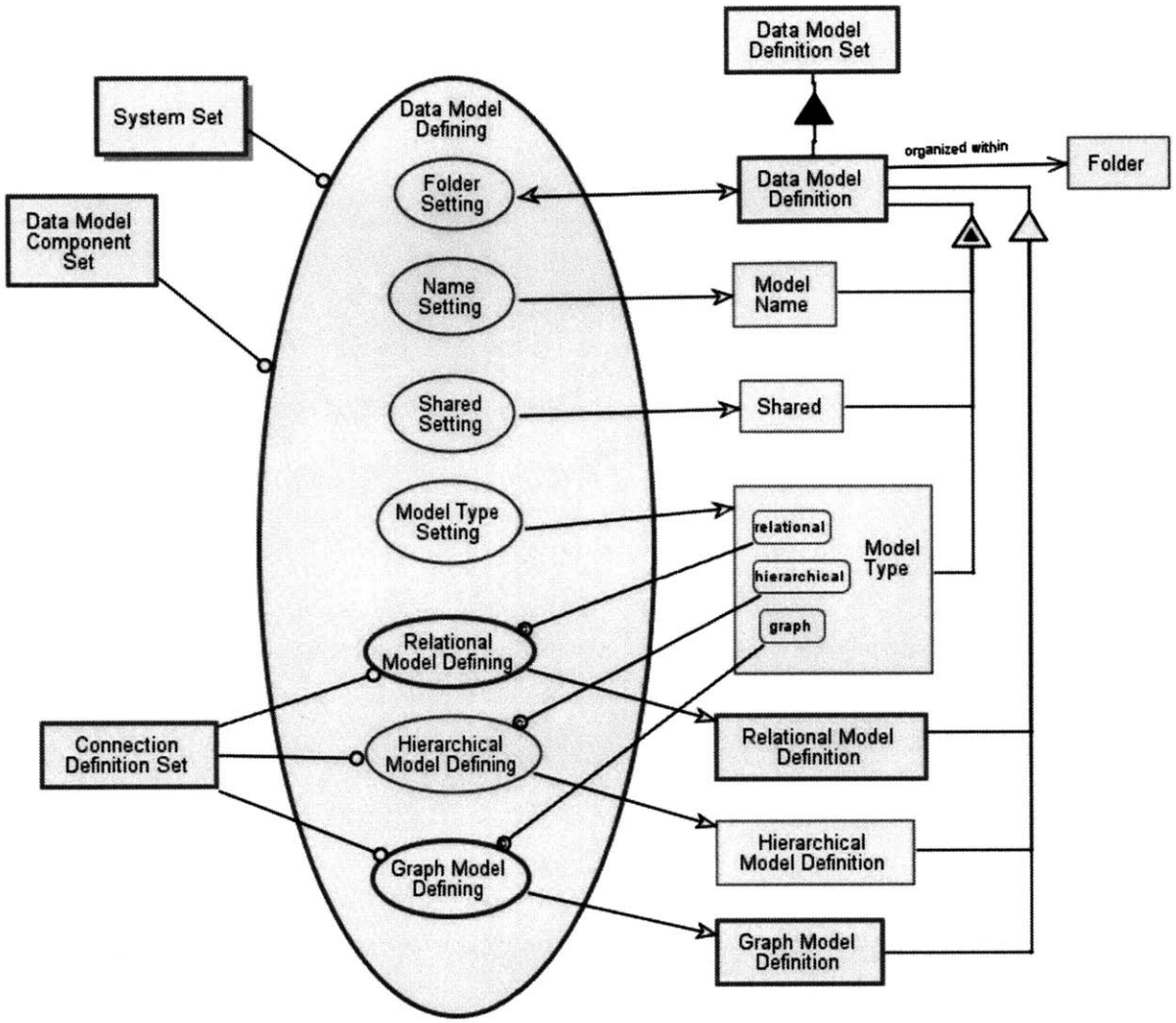


Figure 40 - Data Model Defining in-zoomed

System Set **is physical.**
Data Model Definition Set **consists of many** Data Model Definitions.
Data Model Definition **exhibits** Model Name, Model Type, **and** Shared.
Model Type **can be** relational, hierarchical, **or** graph.
Data Model Definition **organized within** Folder.
Graph Model Definition **is a** Data Model Definition.
Relational Model Definition **is a** Data Model Definition.
Hierarchical Model Definition **is a** Data Model Definition.
Data Model Defining **consists of** Folder Setting, Name Setting, Model Type Setting, Relational Model Defining, Hierarchical Model Defining, Graph Model Defining, **and** Shared Setting.
Data Model Defining **requires** not integrated System Set **and** Data Model Component Set.
Data Model Defining **zooms into** Folder Setting, Name Setting, Shared Setting, Model Type Setting, Relational Model Defining, Hierarchical Model Defining, **and** Graph Model Defining.

Folder Setting **affects** Data Model Definition.
Name Setting **yields** Model Name.
Shared Setting **yields** Shared.
Model Type Setting **yields** Model Type.
Relational Model Defining **occurs if** Model Type **is** relational.
Relational Model Defining **requires** Connection Definition Set.
Relational Model Defining **yields** Relational Model Definition.
Hierarchical Model Defining **occurs if** Model Type **is** hierarchical.
Hierarchical Model Defining **requires** Connection Definition Set.
Hierarchical Model Defining **yields** Hierarchical Model Definition.
Graph Model Defining **occurs if** Model Type **is** graph.
Graph Model Defining **requires** Connection Definition Set.
Graph Model Defining **yields** Graph Model Definition.

Relational Model Definition unfolded

The Relational Model Definition is used to store the structure of a relational model and is one of the outcomes of the Data Model Defining process. This metamodel allows a User in the User Group to create Entity Definitions (tables), Attribute Definitions (columns), and Entity Relationship Definitions (relationships / constraints / foreign keys). From this metamodel, data can be read from and written to relational data sources.

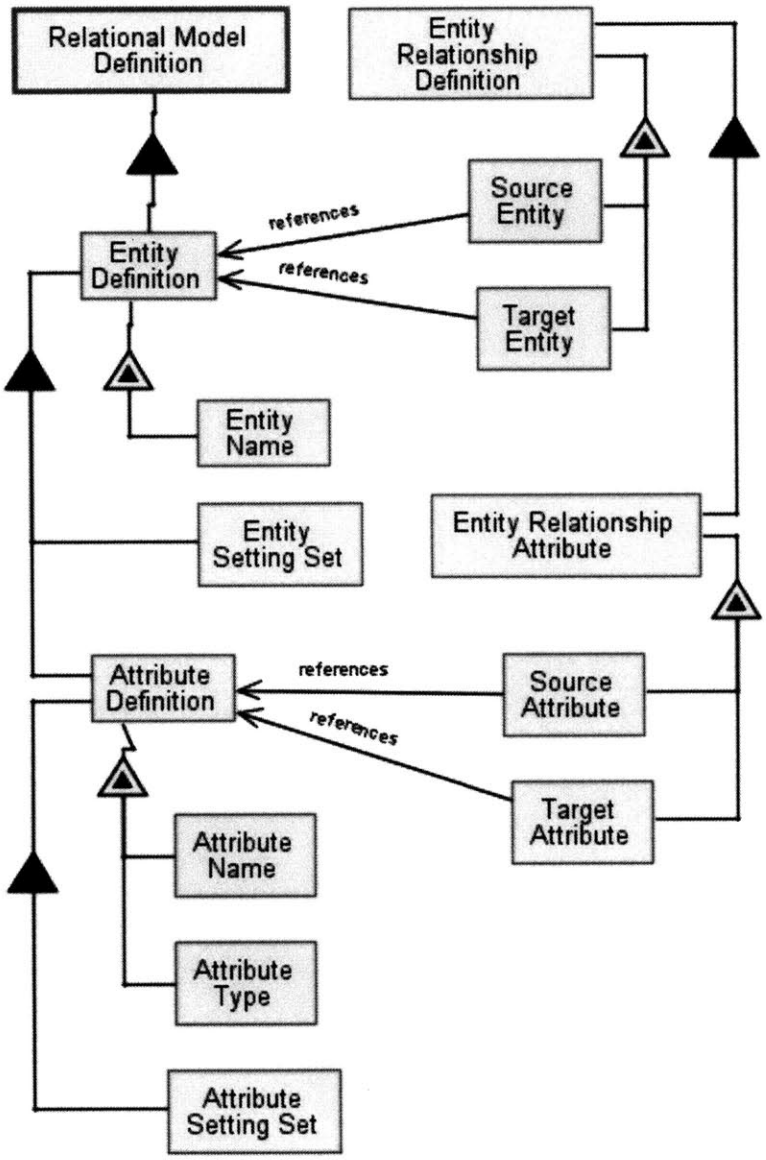


Figure 41 - Relational Model Definition unfolded

Entity Relationship Definition **exhibits** Source Entity **and** Target Entity.
Source Entity **references** Entity Definition.
Target Entity **references** Entity Definition.
Entity Relationship Definition **consists of** Entity Relationship Attribute.
Entity Relationship Attribute **exhibits** Source Attribute **and** Target Attribute.
Source Attribute **references** Attribute Definition.
Target Attribute **references** Attribute Definition.
Relational Model Definition **consists of** Entity Definition.
Entity Definition **exhibits** Entity Name.
Entity Definition **consists of** Attribute Definition **and** Entity Setting Set.
Attribute Definition **exhibits** Attribute Name **and** Attribute Type.
Attribute Definition **consists of** Attribute Setting Set.

Graph Model Definition unfolded

Graph Model Definitions can be defined and are represented as follows.

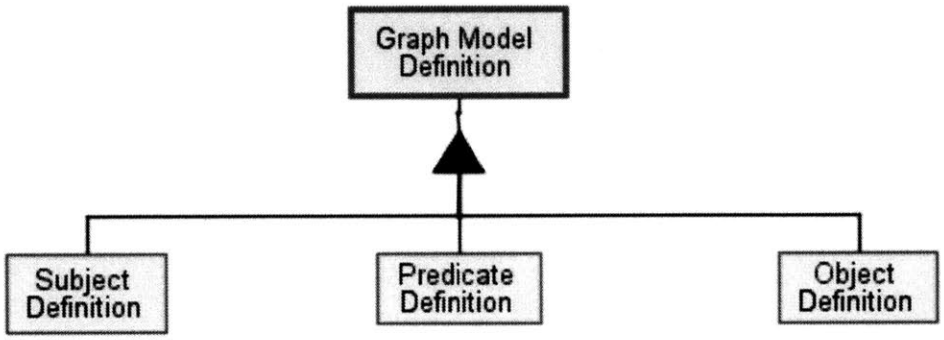


Figure 42 - Graph Model Definition unfolded

Graph Model Definition **consists of** Subject Definition, Predicate Definition, **and** Object Definition.

Relational Model Defining in-zoomed

For each of the Data Model Definition types described above, the subsequent processes are used to define them.

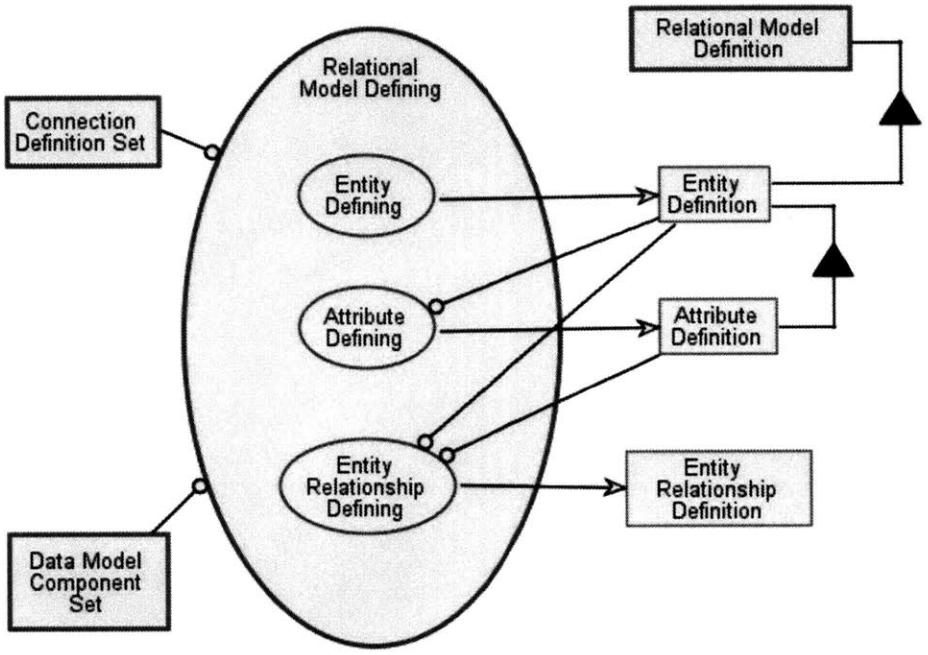


Figure 43 - Relational Model Defining in-zoomed

Relational Model Definition **consists of** Entity Definition.
Entity Definition **consists of** Attribute Definition.
Relational Model Defining **consists of** Entity Defining, Attribute Defining, **and** Entity Relationship Defining.
Relational Model Defining **requires** Data Model Component Set **and** Connection Definition Set.
Relational Model Defining **zooms into** Entity Defining, Attribute Defining, **and** Entity Relationship Defining.
Entity Defining **yields** Entity Definition.
Attribute Defining **requires** Entity Definition.
Attribute Defining **yields** Attribute Definition.
Entity Relationship Defining **requires** Attribute Definition **and** Entity Definition.
Entity Relationship Defining **yields** Entity Relationship Definition.

Graph Model Defining in-zoomed

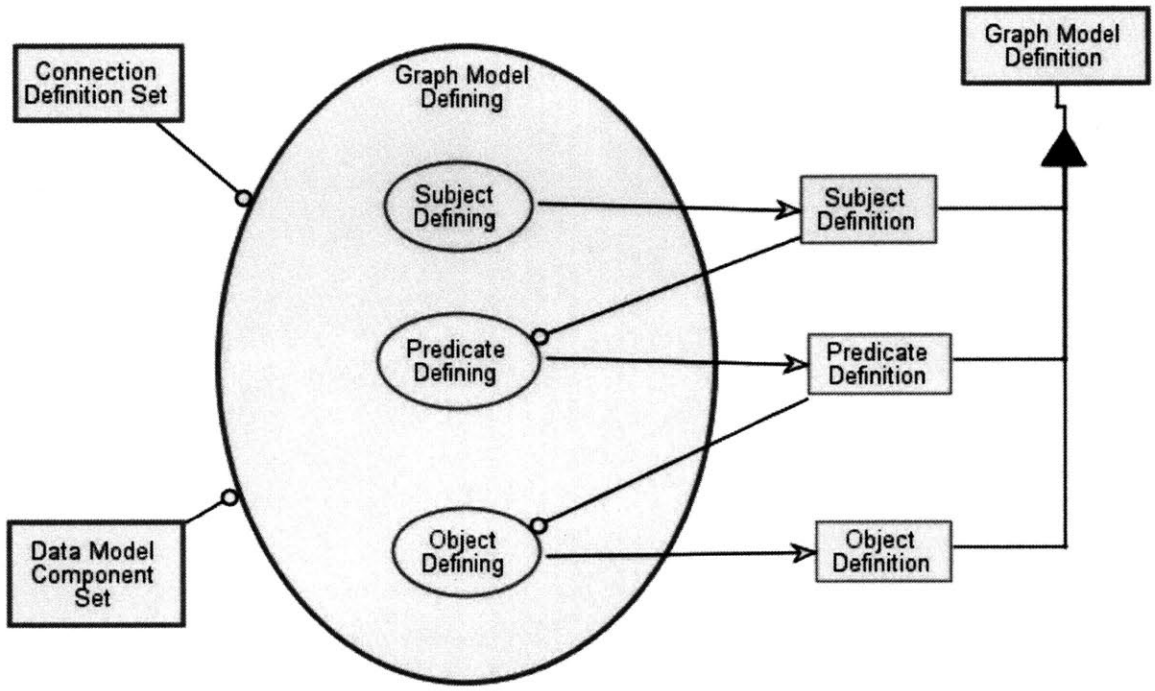


Figure 44 - Graph Model Defining in-zoomed

Graph Model Definition **consists of** Subject Definition, Predicate Definition, **and** Object Definition.
Graph Model Defining **consists of** Subject Defining, Predicate Defining, **and** Object Defining.
Graph Model Defining **requires** Data Model Component Set **and** Connection Definition Set.
Graph Model Defining **zooms into** Subject Defining, Predicate Defining, **and** Object Defining.

Subject Defining **yields** Subject Definition.
Predicate Defining **requires** Subject Definition.
Predicate Defining **yields** Predicate Definition.
Object Defining **requires** Predicate Definition.
Object Defining **yields** Object Definition.

Integration DAG Component Set unfolded

Once Connection Definitions and Data Model Definitions are constructed, the next step in the process is to define the Integration DAG Definition. This DAG represents the steps needed for integration processing.

The Integration DAG Component Set is a set of components used in defining the DAG. It consists of:

- **Reader Component Set** – Set of components used to define readers which read data from a Data Set on a System

- **Processor Component Set** - A set of components used to define processing steps such as translating, aggregating, routing, etc.

- **Writer Component Set** – A set of components used to define writers which write data to a Data Set on a System

- **Format Component Set** – A set of components that allows definition of a Format Definition (how data is laid out) for a given Data Set

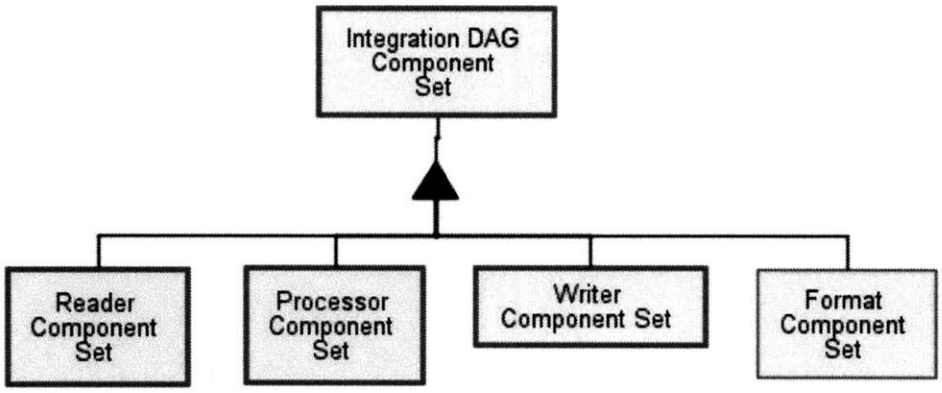


Figure 45 - Integration DAG Component Set unfolded

Integration DAG Component Set **consists of** Reader Component Set, Processor Component Set, Writer Component Set, **and** Format Component Set.

Integration DAG Definition unfolded

The Integration DAG Definition represents an integration defined by a User in the User Group. It contains the following elements:

- **Folder** – An Integration DAG Definition is organized within a Folder. This Folder structure is used in the user interface to allow a User in the User Group to categorize and organize Integration DAG Definitions in any way they see fit

- **Name** – A user defined name for this Integration DAG Definition

- **Component Definition Node Set** – A set of Component Definitions (nodes) belonging to that Integration DAG.

- **Component Definition Link Set** - A set of Component Definition Links that tie the Component Definitions (nodes) together in the Integration DAG.

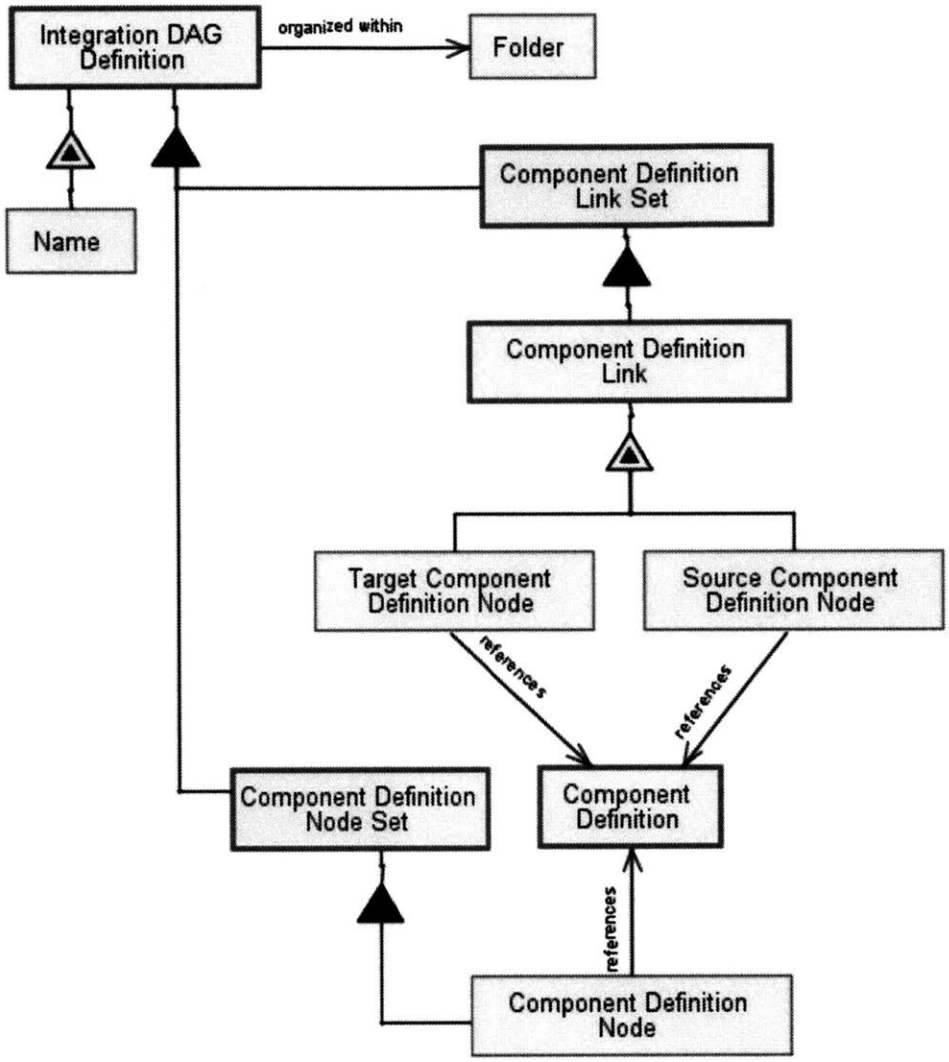


Figure 46 - Integration DAG Definition unfolded

Component Definition Link **exhibits** Source Component Definition Node **and** Target Component Definition Node.

Source Component Definition Node **references** Component Definition Node Set.

Target Component Definition Node **references** Component Definition Node Set.

Integration DAG Definition **exhibits** Name.

Integration DAG Definition **consists of many** Component Definition Node Sets.

Component Definition Node Set **references** Component Definition.

Integration DAG Definition **organized within** Folder.

Component Definition Node Set unfolded

Each Integration DAG Definition (integration) has a Component Definition Node Set. A Component Definition Node Set is a collection of Component Definition Nodes. Each Component Definition Node represents a Component Definition in context to the Integration DAG Definition that it resides within. As an example, a Component Definition may represent a database reader that reads data from a relational database. The Component Definition is referenced by a Component Definition Node when that Component Definition is placed on or used within an Integration DAG. The context of the Component Definition within the integration as a whole is represented by the Component Definition Node.

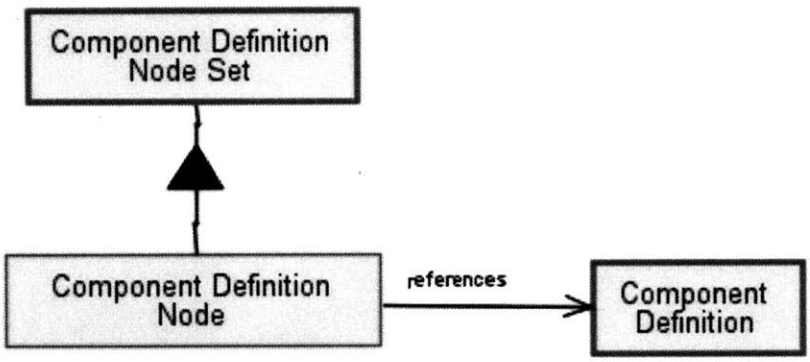


Figure 47 - Component Definition Node Set unfolded

Component Definition Node Set **consists of** Component Definition Node.
Component Definition Node **references** Component Definition.

Integration DAG Defining in-zoomed

In defining the Integration DAG and producing the Integration DAG Definition, the User completes the following activities:

- Set the **Folder** – An Integration DAG Definition is organized within a Folder. This Folder structure is used in the user interface to allow a User in the User Group to categorize and organize Integration DAG Definitions in any way they see fit
- Set the **Name** – A user defined name for this Integration DAG Definition
- Create a **Component Definition Node Set** – A set of Component Definitions (nodes) belonging to that Integration DAG. Each action the Users wants completed with the Integration DAG will be defined as a Node within the graph

- **Component Definition Link Set** - A set of Component Definition Links that tie the Component Definitions (nodes) together in the Integration DAG.

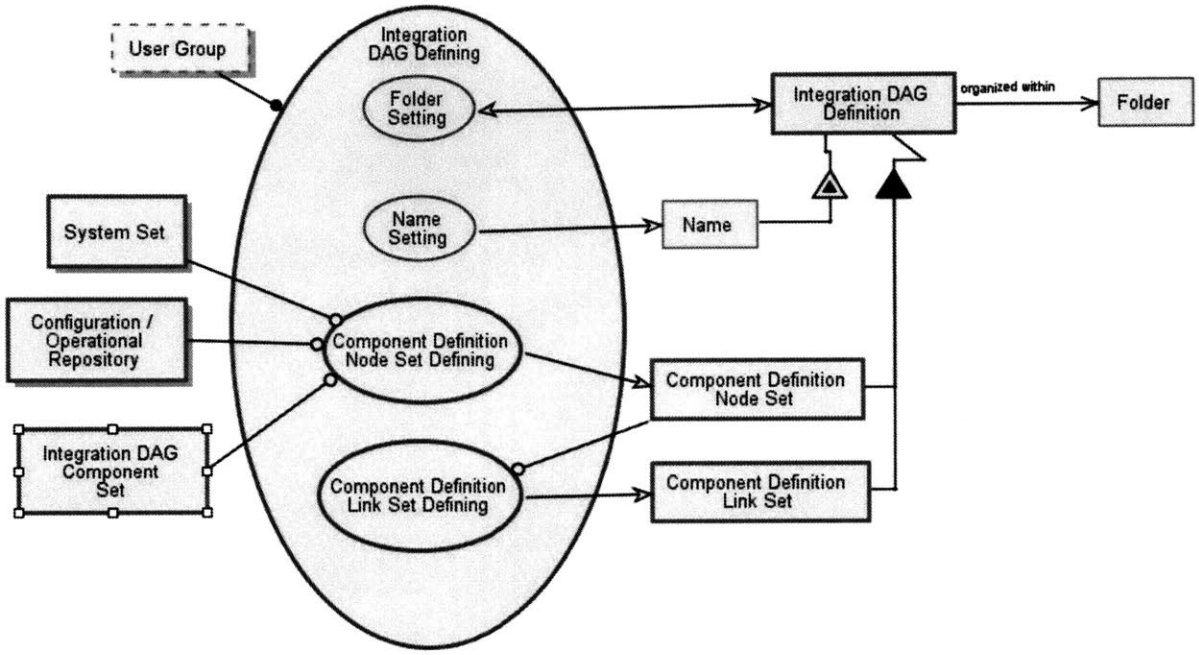


Figure 48 - Integration DAG Defining in-zoomed

System Set **is physical.**
User Group **is environmental and physical.**
User Group **handles** Integration DAG Defining.
Configuration / Operational Repository **is physical.**
Integration DAG Definition **exhibits** Name.
Integration DAG Definition **consists of many** Component Definition Node Sets **and** Component Definition Link Set.
Integration DAG Definition **organized within** Folder.
Integration DAG Defining **consists of** Component Definition Node Set Defining, Component Definition Link Set Defining, Folder Setting, **and** Name Setting.
Integration DAG Defining **zooms into** Folder Setting, Name Setting, Component Definition Node Set Defining, **and** Component Definition Link Set Defining.

Folder Setting **affects** Integration DAG Definition.

Name Setting **yields** Name.

Component Definition Node Set Defining **requires** System Set, Configuration / Operational Repository, **and** Integration DAG Component Set.

Component Definition Node Set Defining **yields** Component Definition Node Set.

Component Definition Link Set Defining **requires** Component Definition Node Set.

Component Definition Link Set Defining **yields** Component Definition Link Set.

Component Definition Node Set Defining in-zoomed

Defining a set of nodes for the Integration DAG can be done by creating the Component Definitions from scratch, by selecting previously created Component Definitions, or both.

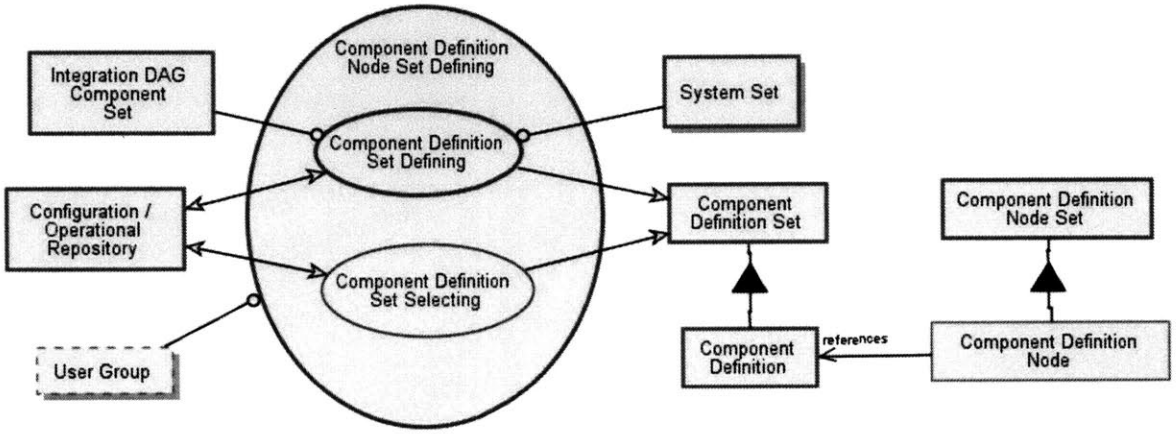


Figure 49 - Component Definition Node Set Defining in-zoomed

System Set **is physical.**
User Group **is environmental and physical.**
Configuration / Operational Repository **is physical.**
Component Definition Node Set **consists of** Component Definition Node.
Component Definition Node **references** Component Definition.
Component Definition Set **consists of** Component Definition.
Component Definition Node Set Defining **consists of** Component Definition Set Defining **and** Component Definition Set Selecting.
Component Definition Node Set Defining **requires** User Group.
Component Definition Node Set Defining **zooms into** Component Definition Set Defining **and** Component Definition Set Selecting.
Component Definition Set Defining **requires** System Set **and** Integration DAG Component Set.
Component Definition Set Defining **affects** Configuration / Operational Repository.
Component Definition Set Defining **yields** Component Definition Set.
Component Definition Set Selecting **affects** Configuration / Operational Repository.
Component Definition Set Selecting **yields** Component Definition Set.

Component Definition Set Defining in-zoomed

When creating Component Definitions for a DAG, a User in a User Group must create or select at least one Reader Definition and at least one Writer Definition. An integration must read from at least one source and write to another. Processing steps such as translating data between models are likely but not necessarily required. The output of

the Component Definition Set Defining is a set of Component Definitions that will be referenced by Component Definition Nodes in the Integration DAG Definition.

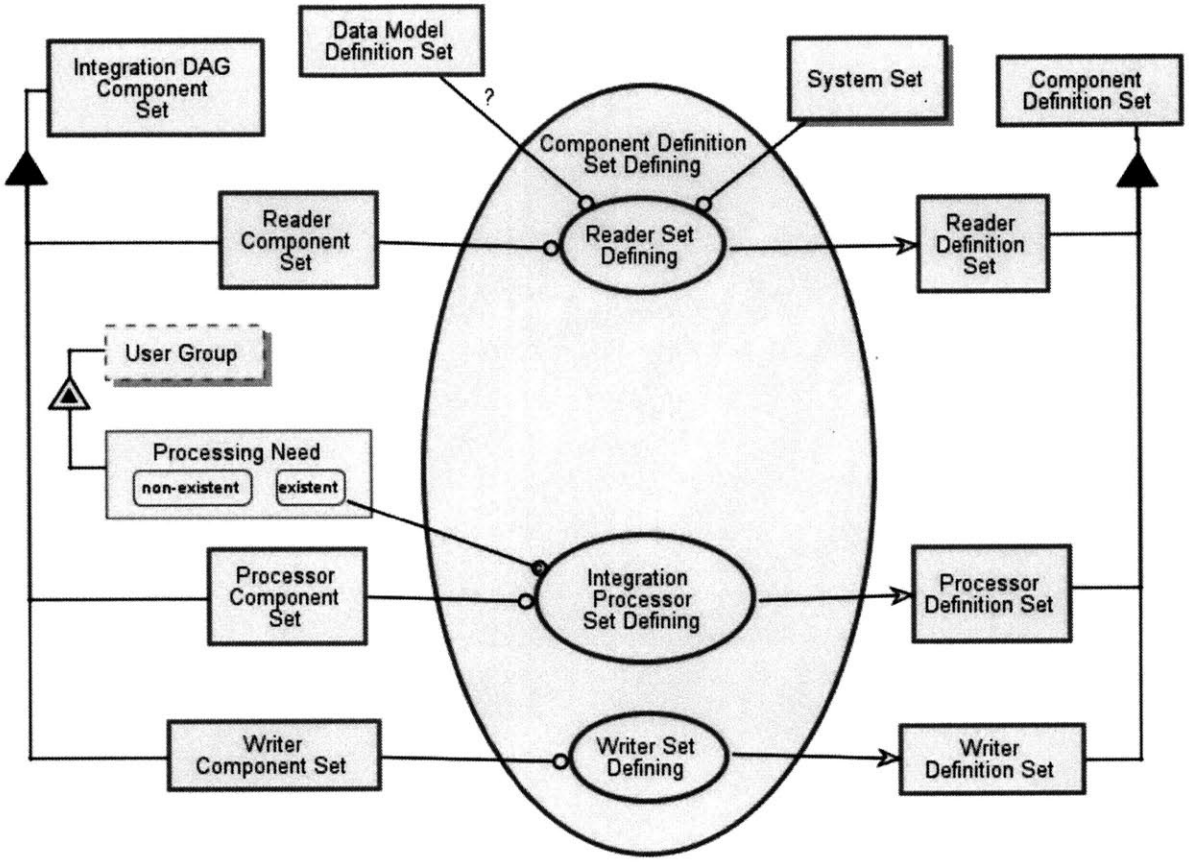


Figure 50 - Component Definition Set Defining in-zoomed

System Set **is physical.**
User Group **is environmental and physical.**
User Group **exhibits** Processing Need.

Processing Need **can be** existent **or** non-existent.

Integration DAG Component Set **consists of** Reader Component Set, Processor Component Set, **and** Writer Component Set.
Component Definition Set **consists of** Processor Definition Set, Reader Definition Set, **and** Writer Definition Set.
Component Definition Set Defining **consists of** Integration Processor Set Defining, Reader Set Defining, **and** Writer Set Defining.
Component Definition Set Defining **zooms into** Reader Set Defining, Integration Processor Set Defining, **and** Writer Set Defining.

Reader Set Defining **requires** System Set, Data Model Definition Set, **and** Reader Component Set.

Reader Set Defining **yields** Reader Definition Set.

Integration Processor Set Defining **occurs if** Processing Need **is** existent.

Integration Processor Set Defining **requires** Processor Component Set.

Integration Processor Set Defining **yields** Processor Definition Set.

Writer Set Defining **requires** Writer Component Set.
Writer Set Defining **yields** Writer Definition Set.

Reader Component Set unfolded

The Reader Component Set, used in the Reader Defining process allows a User to define a Reader Definition that allows reading data from a relational or file data source. Two initially supported readers will be Database and File readers. Other readers will be added over time including queue readers, etc.

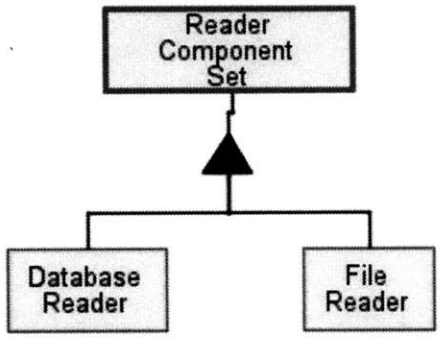


Figure 51 - Reader Component Set unfolded

Reader Component Set **consists of** Database Reader **and** File Reader.

Reader Definition Set unfolded

A Reader Definition Set is simply a collection of Reader Definitions used within an Integration DAG Definition.

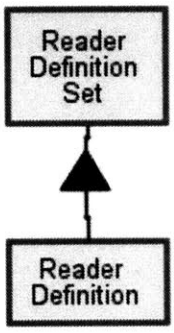


Figure 52 - Reader Definition Set unfolded

Reader Definition Set **consists of** Reader Definition.

Reader Set Defining in-zoomed

One or more Reader Definitions may be created when creating an Integration DAG Definition.

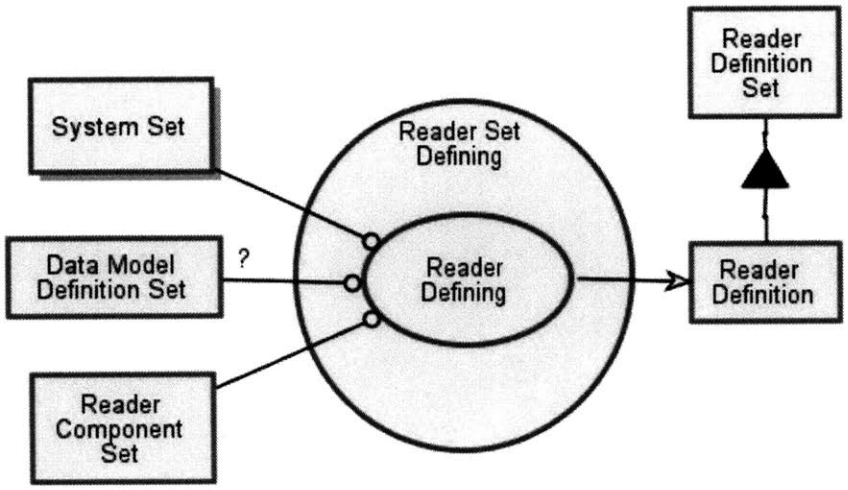


Figure 53 - Reader Set Defining in-zoomed

System Set **is physical.**
Reader Definition Set **consists of** Reader Definition.
Reader Set Defining **consists of** Reader Defining.
Reader Set Defining **zooms into** Reader Defining.

Reader Defining **requires** System Set, Data Model Definition Set, **and** Reader Component Set.

Reader Defining **yields** Reader Definition.

Reader Definition unfolded

A Reader Definition is comprised of the following attributes:

- **Folder** – A Reader Definition is organized within a Folder. This Folder structure is used in the user interface to allow a User in the User Group to categorize and organize Reader Definitions in any way they see fit
- **Reader Component Type** – The type of reader (currently database of file)
- **Shared** – Whether this Reader Definition can be shared amongst other Integration DAG Definitions
- **Output Model** - The Output Model references a Data Model Definition that describes the output model of the data provided by the reader
- **Output Format** - The Output Format references a Format Definition that describes the output format of the data provided by the reader
- **Reader Definition Setting Set** – Similar to a Connection Definition Setting Set, the Reader Definition Setting Set allows key/value pairs of reader specific

attributes to be saved without hardcoding a set of definition tables for each reader. This will allow extensibility readers without changing the underlying Data Integrating System model.

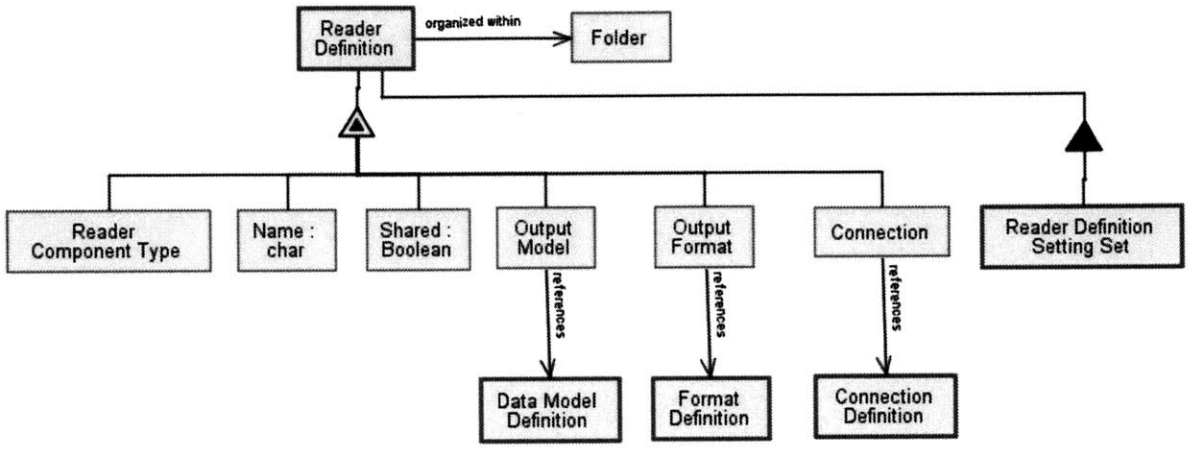


Figure 54 - Reader Definition unfolded

Reader Definition **exhibits** Name, Shared, Output Model, Output Format, Connection, **and** Reader Component Type.

Name **is of type char.**

Shared **is of type Boolean.**

Output Model **references** Data Model Definition Set.

Output Format **references** Format Definition.

Connection **references** Connection Definition.

Reader Definition **consists of** Reader Definition Setting Set.

Reader Definition **organized within** Folder.

Reader Defining in-zoomed

The Reader Defining process allows each attribute of the Reader Definition to be set by the User.

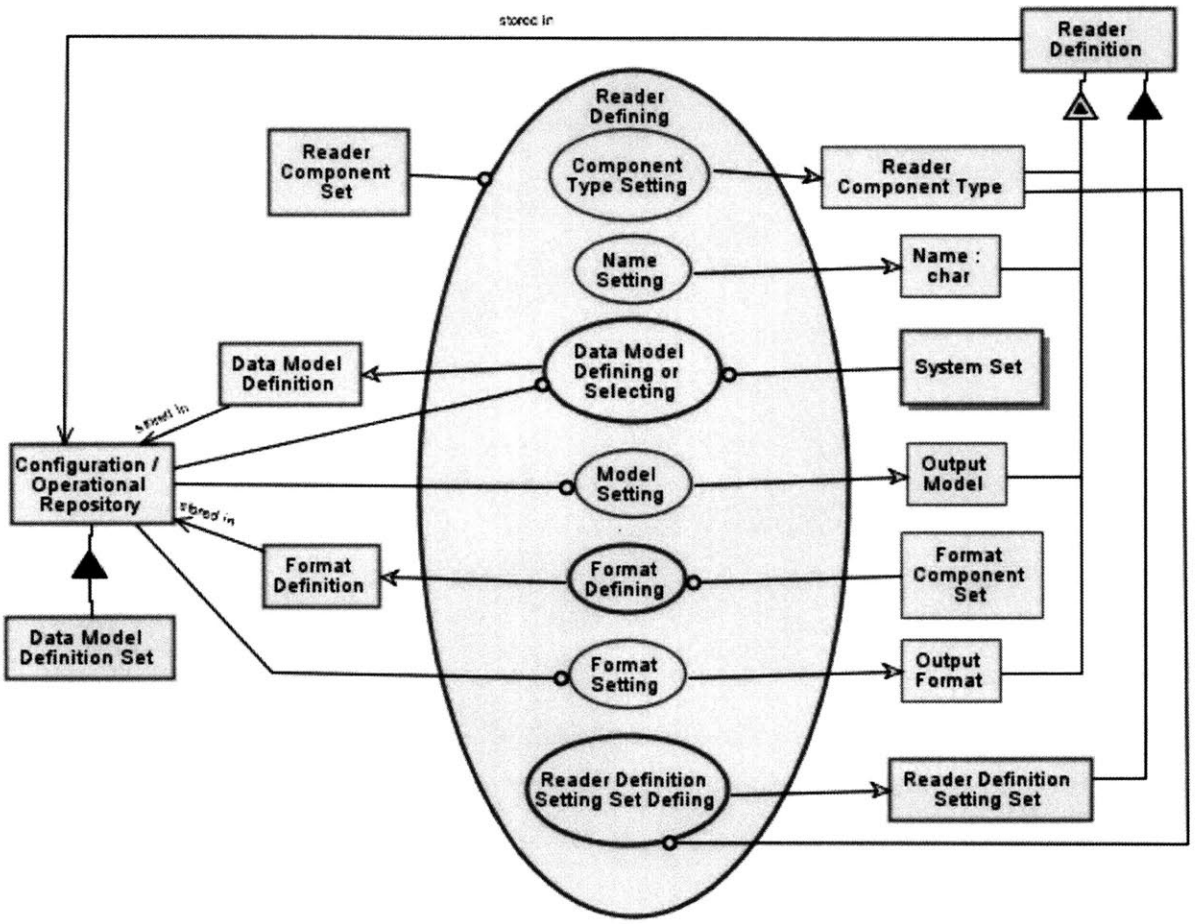


Figure 55 - Reader Defining in-zoomed

Data Integrating System **is physical.**
System Set **is physical.**
System Set **can be** not integrated **or** integrated.
    not integrated **is initial.**
    integrated **is final.**
User Group **is environmental and physical.**
User Group **can be** uninformed **or** informed.
    uninformed **is initial.**
    informed **is final.**
Data Integrating **requires** Data Integrating System.
Data Integrating **changes** User Group **from** uninformed **to** informed **and** System Set **from** not integrated **to** integrated.

Data Model Defining or Selecting in-zoomed

As part of the Reader Defining process, an Output Model must be specified. This Output Model can be created when defining the Reader Definition, or earlier in the Integration Defining process (as a global ontology) as described above. The Data Model Defining or Data Model Selecting process simply allows the user to select an existing Data Model Definition or create a new one.

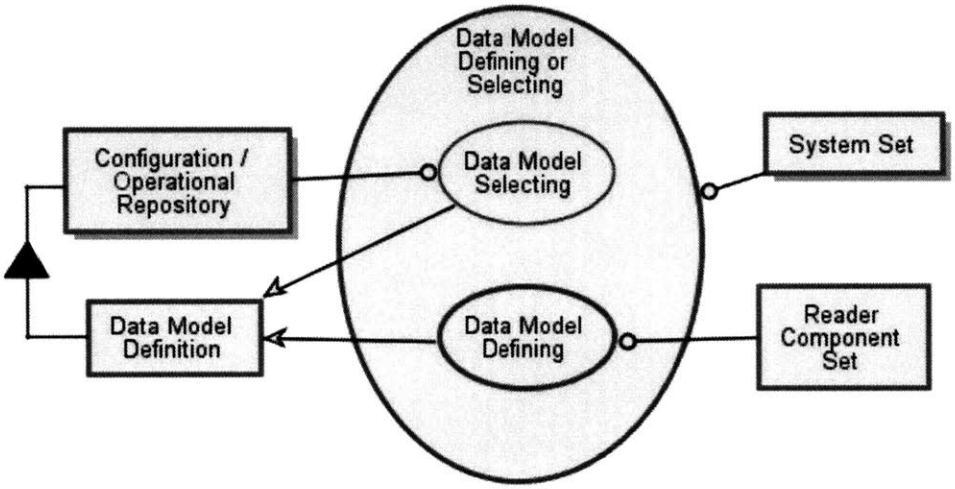


Figure 56 - Data Model Defining or Selecting in-zoomed

System Set **is physical.**
Configuration / Operational Repository **is physical.**
Configuration / Operational Repository **consists of** Data Model Definition.
Data Model Defining or Selecting **consists of** Data Model Defining **and** Data Model Selecting.
Data Model Defining or Selecting **requires** System Set.
Data Model Defining or Selecting **zooms into** Data Model Selecting **and** Data Model Defining.

Data Model Selecting **requires** Configuration / Operational Repository.
Data Model Selecting **yields** Data Model Definition.
Data Model Defining **requires** Reader Component Set.
Data Model Defining **yields** Data Model Definition.

Format Definition unfolded

A Data Set on a System may be laid out in a given format. As an example, a flat file may adhere to a Data Model Definition and be laid out in a given format (possibly fixed length or delimited). When defining a reader a Format Definition is defined. A Format Definition consists of:

- **Format Type** – The type of format. Initially supported will be XML, Delimited and Fixed Length Formats. Others may be added in the future.

- **Format Setting Set** – Key / Value pairs of settings that apply to the entire Format Definition. Examples include the delimiter in a Delimited Format definition. I.E. a '|' or ',' or tab.

- **Entity Format Setting Set** – Key / Value pairs of settings that apply to entities in the Format Definition. As an example, in a Fixed Length or Delimited Format, the file might have multiple record types (entities) that are designated by different record type identifiers in the file.

- **Attribute Format Setting Set** - Key / Value pars of settings that apply to attributes in the Format Definition. As an example, in a Fixed Length Format, the attribute will have a start and end character position with the line. In a Delimited Format, the attribute will have a position number within the line, etc.

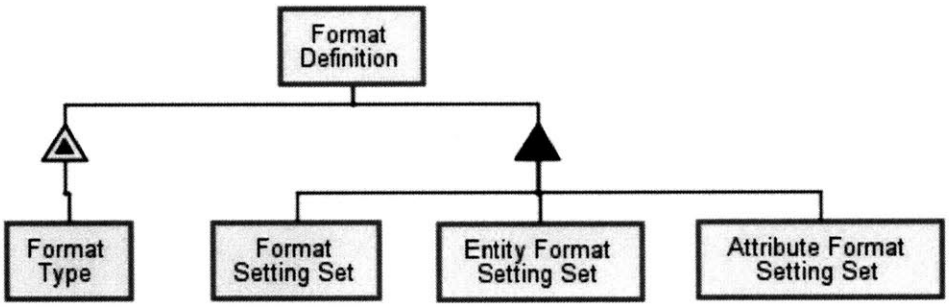


Figure 57 - Format Definition unfolded

Format Definition **exhibits** Format Type.
Format Definition **consists of** Format Setting Set, Entity Format Setting Set, **and** Attribute Format Setting Set.

Format Type unfolded

Format Type can be one of XML Format, Delimited Format or Fixed Length Format.

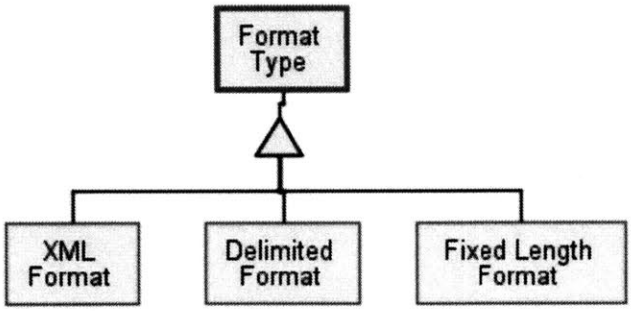


Figure 58 - Format Type unfolded

XML Format **is a** Format Type.
Fixed Length Format **is a** Format Type.
Delimited Format **is a** Format Type.

Format Setting Set unfolded

The Format Setting Set is a generic way to store information about different format types without hard coding the Data Integrating System database design, allowing for additional Format Types in the future. The XML Format Setting Set is described by an XML template that describes the desired XML format. The Delimited Format Setting Set is described by a Delimiter (“,” or “|” or tab, etc.) that defines the delimiting character that delineates the fields.

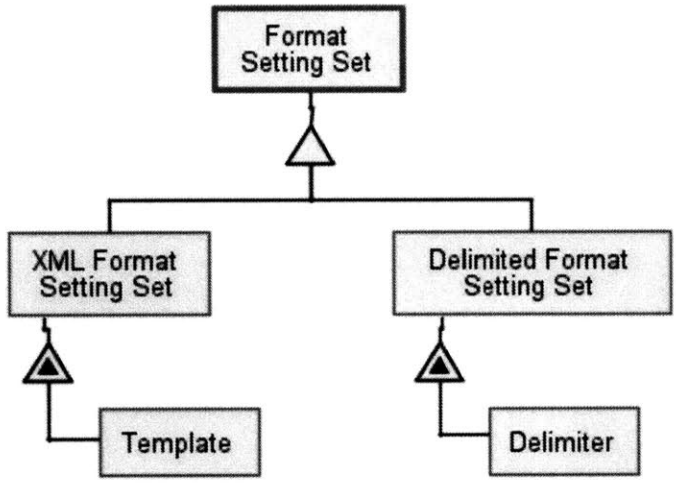


Figure 59 - Format Setting Set unfolded

XML Format Setting Set **is a** Format Setting Set.
XML Format Setting Set **exhibits** Template.
Delimited Format Setting Set **is a** Format Setting Set.
Delimited Format Setting Set **exhibits** Delimiter.

Entity Format Setting Set unfolded

The Entity Format Setting Set is used to describe Entity formats and is used in all three Format Types. The Delimited and Fixed Length Entity Setting Sets have a Record Type Pattern that describes the pattern for different record types within a delimited file (if the file contains more than one record type). The XML Entity Setting Set has an Xpath expression that defines how that instance of an entity can be paired with the previously defined XML Format Setting Set Template.

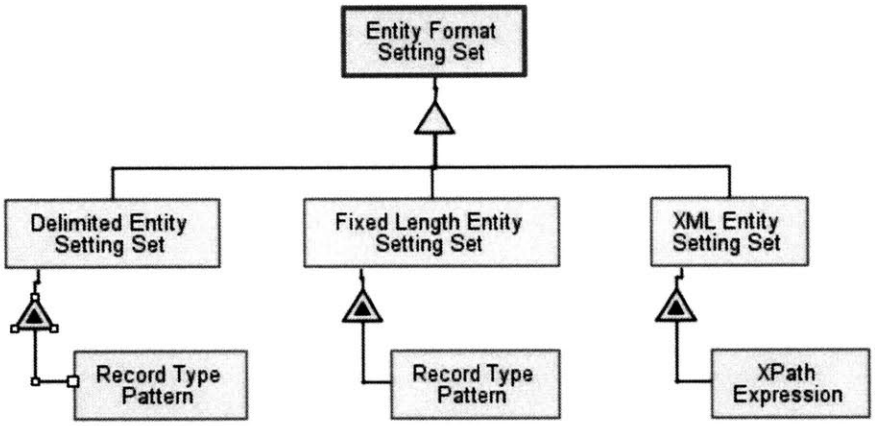


Figure 60 - Entity Format Setting Set unfolded

XML Entity Setting Set **is an** Entity Format Setting Set.
XML Entity Setting Set **exhibits** XPath Expression.
Fixed Length Entity Setting Set **is an** Entity Format Setting Set.
Fixed Length Entity Setting Set **exhibits** Record Type Pattern.
Delimited Entity Setting Set **is an** Entity Format Setting Set.
Delimited Entity Setting Set **exhibits** Record Type Pattern.

Attribute Format Setting Set unfolded

The Attribute Format Setting Set is used to describe Attribute formats and is used in all three Format Types. The Delimited Attribute Setting Set contains a Position (i.e. the ordinal position of the field within that delimited record). The Fixed Length Attribute Setting Set contains a Start and End Position that describes the start and end character position for the field within that fixed length record, and the XML Attribute Setting Set contains an XPath Expression that defines how that instance of a column can be paired with the previously defined XML Format Setting Set Template.

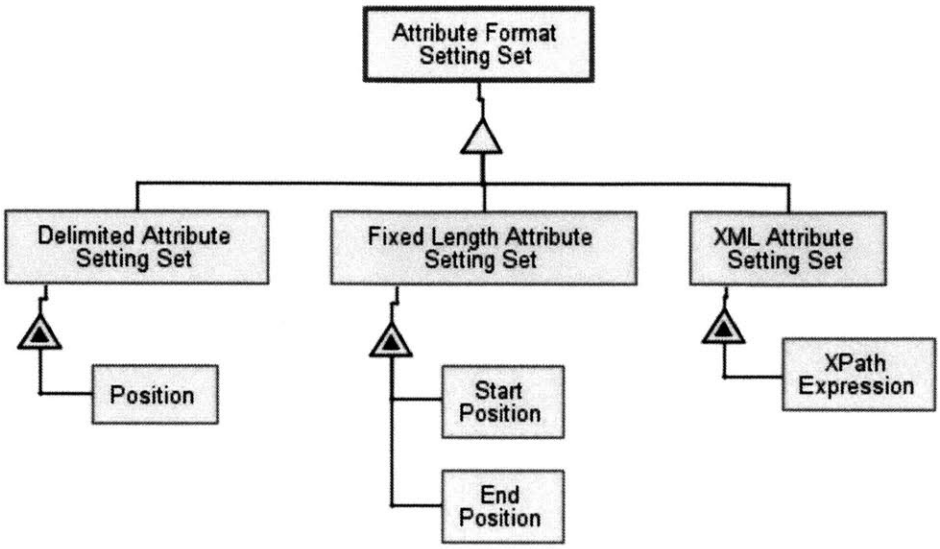


Figure 61 - Attribute Format Setting Set unfolded

XML Attribute Setting Set **is an** Attribute Format Setting Set.
XML Attribute Setting Set **exhibits** XPath Expression.
Fixed Length Attribute Setting Set **is an** Attribute Format Setting Set.
Fixed Length Attribute Setting Set **exhibits** Start Position **and** End Position.
Delimited Attribute Setting Set **is an** Attribute Format Setting Set.
Delimited Attribute Setting Set **exhibits** Position.

Format Defining in-zoomed

The Format Defining process allows each attribute of the Format Definition to be set by the User.

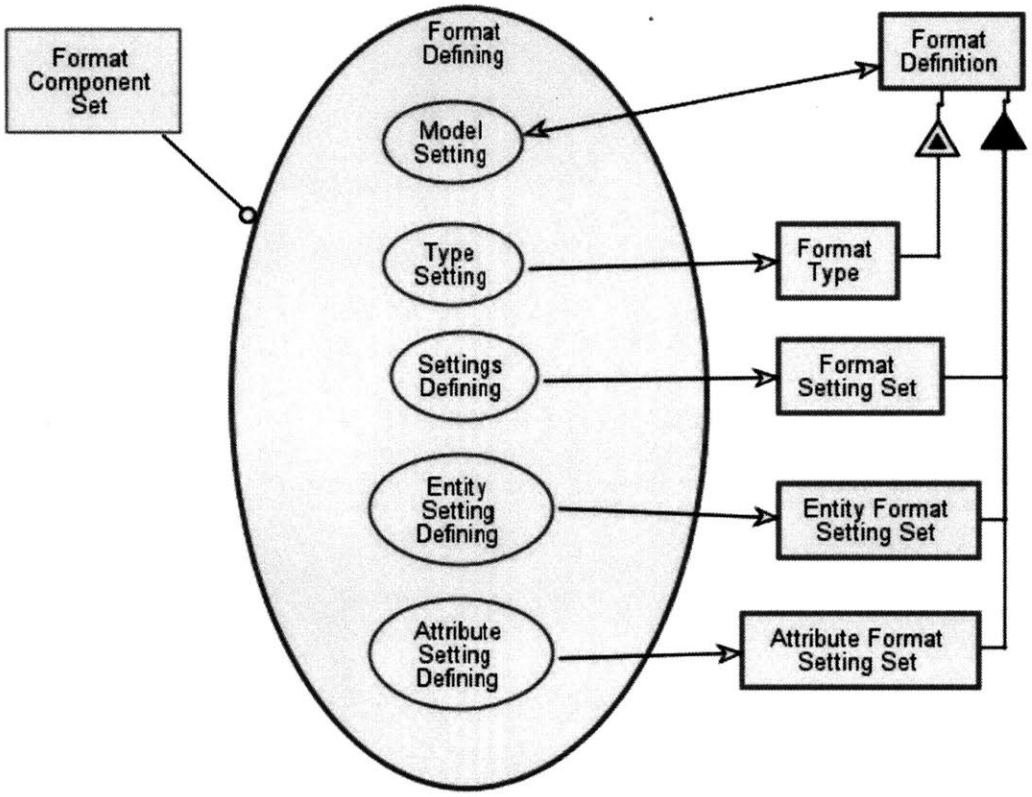


Figure 62 - Format Defining in-zoomed

Format Definition **exhibits** Format Type.
Format Definition **consists of** Format Setting Set, Entity Format Setting Set, **and** Attribute Format Setting Set.
Format Defining **consists of** Model Setting, Type Setting, Settings Defining, Entity Setting Defining, **and** Attribute Setting Defining.
Format Defining **requires** Format Component Set.
Format Defining **zooms into** Model Setting, Type Setting, Settings Defining, Entity Setting Defining, **and** Attribute Setting Defining.

Model Setting **affects** Format Definition.
Type Setting **yields** Format Type.
Settings Defining **yields** Format Setting Set.
Entity Setting Defining **yields** Entity Format Setting Set.
Attribute Setting Defining **yields** Attribute Format Setting Set.

Reader Definition Setting Set unfolded

Similar to the other Setting Set patterns, the Reader Definition Setting Set is used to store configuration information about a generic Reader. Each Reader Type can define different settings that must be configured for the type of reader. Currently the Reader Definition Setting set can be one of Database Reader Definition Setting Set or File Reader Definition Setting Set.

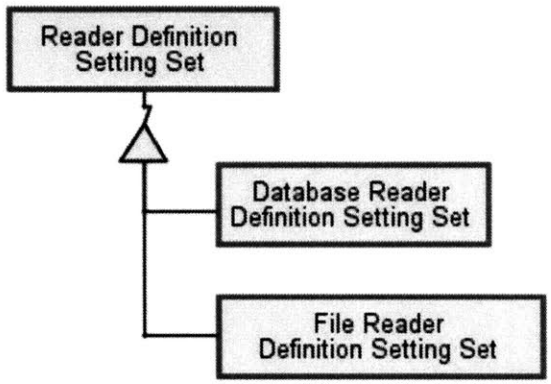


Figure 63 - Reader Definition Setting Set unfolded

Database Reader Definition Setting Set **is a** Reader Definition Setting Set.
File Reader Definition Setting Set **is a** Reader Definition Setting Set.

Database Reader Definition Setting Set unfolded

The Database Reader Definition Setting Set contains the following:

- **Query** – The query to be run to select data for the database reader
- **Trim Columns on Reader** – Whether columns read from the database should be trimmed of blank spaces at the end of the column
- **Rows Per Message** – The number of rows that should be packaged into a single message before that message is sent to the next node in the Integration DAG.

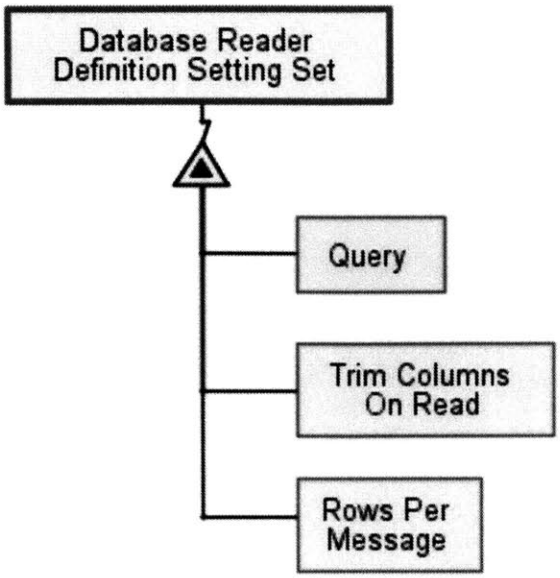


Figure 64 - Database Reader Definition Setting Set unfolded

Database Reader Definition Setting Set **exhibits** Trim Columns On Read, Query, **and** Rows Per Message.

File Reader Definition Setting Set unfolded

The File Reader Definition Setting Set contains the following:

- **Line Terminator** – The character of set of characters that denotes the end of a line in the file
- **Number of Header Lines To Skip** – The number of lines to skip at the beginning of the file specifying these lines are not data but header lines
- **Trim Columns on Reader** – Whether columns read from the file should be trimmed of blank spaces at the end of the column
- **Quote Character** – The character that should be used to quote a string within the file (important for delimited files in cases where a string contains the delimiter character)
- **Relative Path** – The relative path and file where the file resides in context of the absolute path provided by the Connection

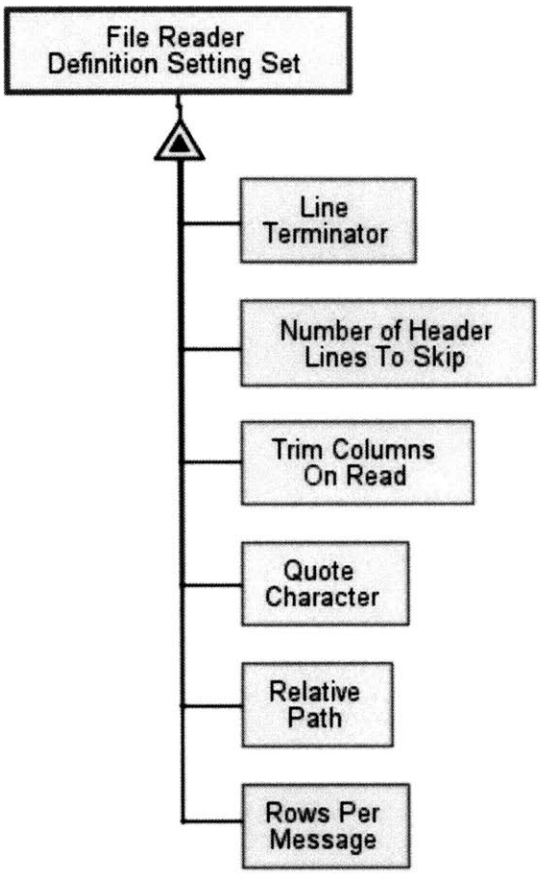


Figure 65 - File Reader Definition Setting Set unfolded

File Reader Definition Setting Set **exhibits** Quote Character, Line Terminator, Number of Header Lines To Skip, Absolute Path, Trim Columns On Read, **and** Rows Per Message.

Reader Definition Setting Set Defining in-zoomed

The Reader Definition Setting Set Defining process allows each attribute of the Reader Definition Setting Set to be set by the User.

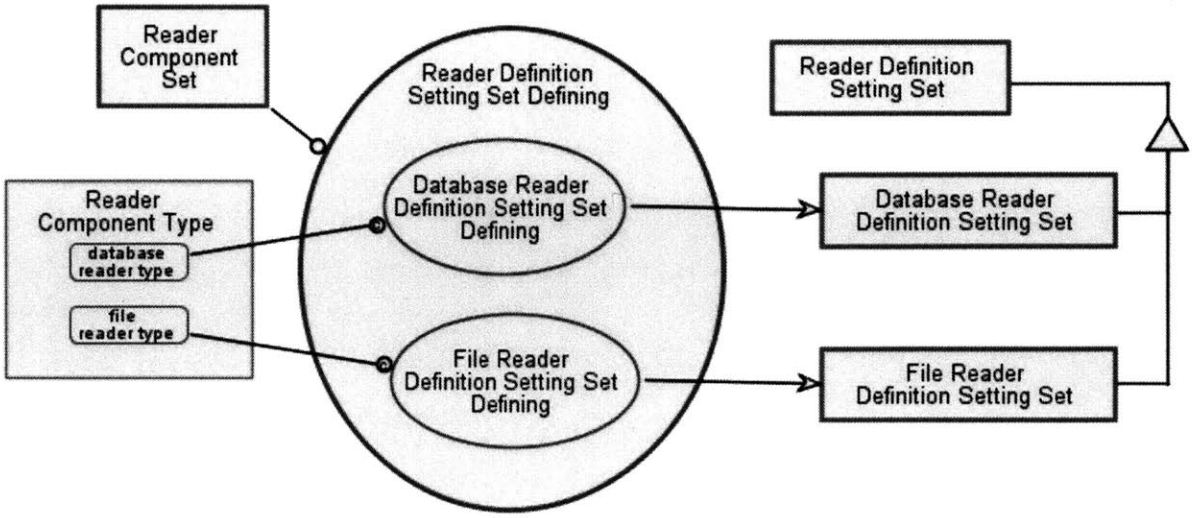


Figure 66 - Reader Definition Setting Set Defining in-zoomed

Reader Component Type **can be** database reader type **or** file reader type.
Database Reader Definition Setting Set **is a** Reader Definition Setting Set.
File Reader Definition Setting Set **is a** Reader Definition Setting Set.
Reader Definition Setting Set Defining **consists of** Database Reader Definition Setting Set Defining **and** File Reader Definition Setting Set Defining.
Reader Definition Setting Set Defining **requires** Reader Component Set.
Reader Definition Setting Set Defining **zooms into** Database Reader Definition Setting Set Defining **and** File Reader Definition Setting Set Defining.

Database Reader Definition Setting Set Defining **occurs if** Reader Component Type **is** database reader type.

Database Reader Definition Setting Set Defining **yields** Database Reader Definition Setting Set.

File Reader Definition Setting Set Defining **occurs if** Reader Component Type **is** file reader type.

File Reader Definition Setting Set Defining **yields** File Reader Definition Setting Set.

Processor Component Set unfolded

There are many types of processors being considered for inclusion into the Integration System Product. They include:

- **Record Translator** – Translates or maps an Entity and all of its Attributes from one Data Model to another.

- **Message Router** – While the Component Definition Link Set of an Integration DAG inherently defines the flow of a message through DAG, there may be times where dynamic routing based on data is desired. The Message Router is a

mechanism to facilitate this functionality. It dynamically sends a message from one Component Definition Node to one or more other Component Definition Nodes based on a routing rule and the message data itself.

- **Message Aggregator** – The Message Aggregator allows multiple messages to be aggregated into a single message.

- **Message Splitter** – The Message Splitter allows a single message to be split into multiple messages

- **Record Enricher** – The Record Enricher allows a record or Entity within a message to be enriched with additional data from other sources

- **Message Sequencer** – The Message Sequencer allows a set of messages to be sequenced into a different order

- **Transformer** – A Transformer allows transformation functions to be applied to Attributes of an Entity within a message

- **Joiner** – A Joiner allows Entities from disparate messages to be joined together into a single message

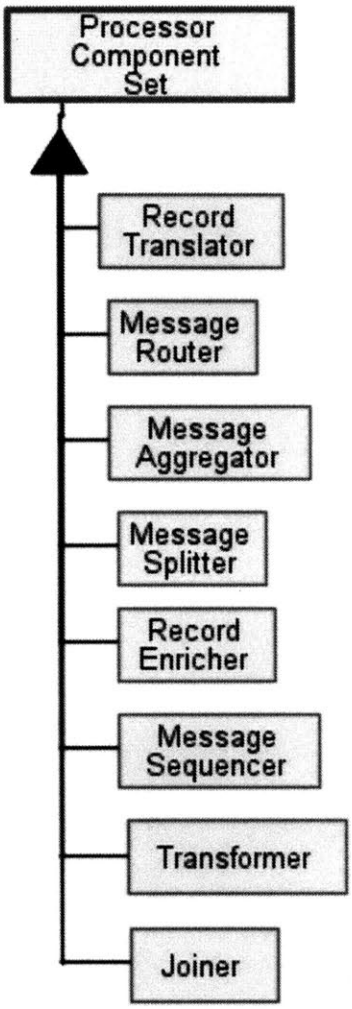


Figure 67 - Processor Component Set unfolded

Processor Component Set **consists of** Record Translator, Message Router, Message Aggregator, Message Splitter, Record Enricher, Message Sequencer, Auditor, Transformer, Record Appender, **and** Joiner.

Processor Definition Set unfolded

A Processor Definition Set contains one or more Processor Definitions

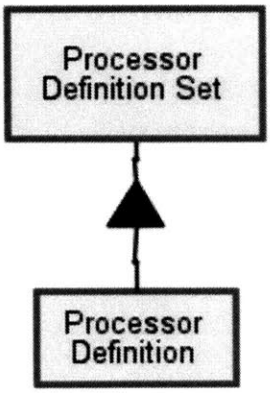


Figure 68 - Processor Definition Set - unfolded

Processor Definition Set consists of Processor Definition.

Processor Definition unfolded

Processor Definition will most likely be broken down into two different types, Message Processor Definitions which will modify a set of Messages, and Record Processor Definitions will affect Entities and Attributes within a Message.

The Processor Definition represents a processing component defined by a User in the User Group. It contains the following elements:

- **Folder** – A Processor Definition is organized within a Folder. This Folder structure is used in the user interface to allow a User in the User Group to categorize and organize Processor Definitions in any way they see fit
- **Processor Component Type** – The type of processor
- **Shared** – Whether this Processor Definition can be shared amongst other Integration DAG Definitions
- **Input Model** - The Input Model references a Data Model Definition and describes the input model of the data provided to the processor
- **Output Model** - The Output Model references a Data Model Definition and describes the output model of the data provided by the processor
- **Processor Definition Setting Set** - The Processor Definition Setting Set is a generic way to store information about different processor types without hard

coding the Data Integrating System database design, allowing for additional Processing Types in the future

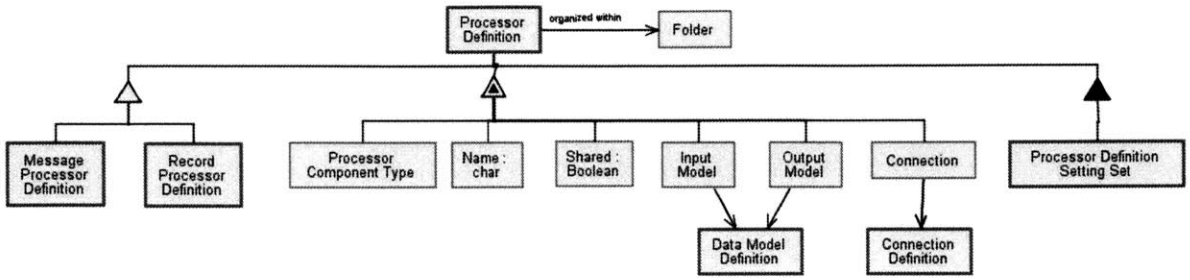


Figure 69 - Processor Definition unfolded

Processor Definition **exhibits** Name, Shared, Input Model, Output Model, Connection, **and** Processor Component Type.

Name **is of type char.**

Shared **is of type Boolean.**

Input Model **relates to** Data Model Definition.

Output Model **relates to** Data Model Definition.

Connection **relates to** Connection Definition.

Connection **relates to** Connection Definition.

Processor Definition **consists of** Processor Definition Setting Set.

Processor Definition **organized within** Folder.

Message Processor Definition **is a** Processor Definition.

Record Processor Definition **is a** Processor Definition.

Processor Definition Setting Set unfolded

Each of the Processor Types will have its own set of settings that can be set via a processor specific setting set.

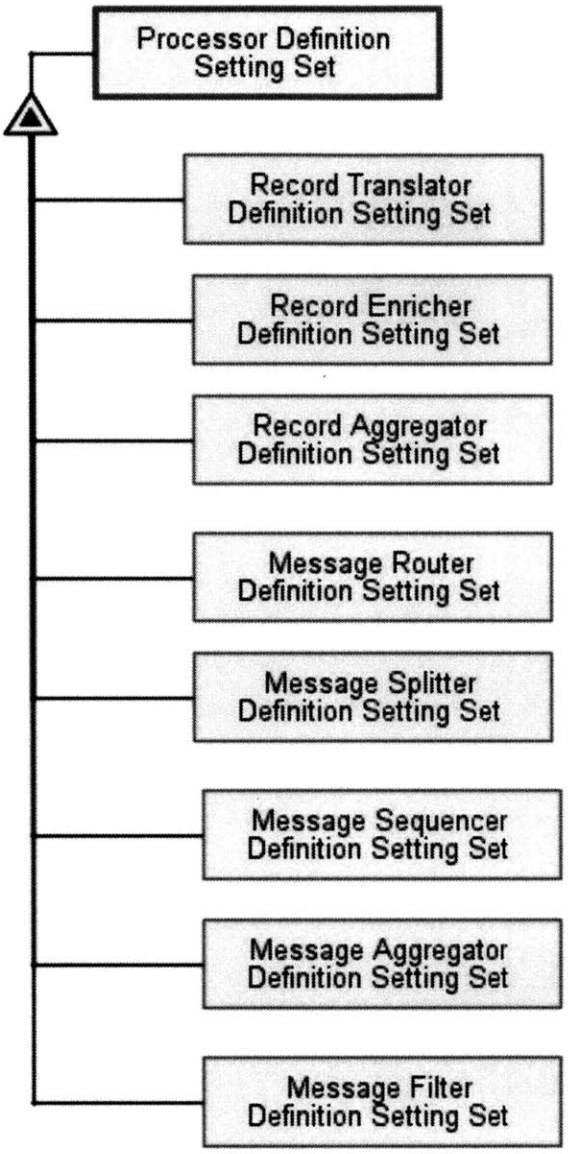


Figure 70 - Processor Definition Setting Set unfolded

Processor Definition Setting Set **exhibits** Message Router Definition Setting Set, Message Splitter Definition Setting Set, Message Sequencer Definition Setting Set, Message Aggregator Definition Setting Set, Message Filter Definition Setting Set, Record Translator Definition Setting Set, Record Enricher Definition Setting Set, **and** Record Aggregator Definition Setting Set.

Integration Processor Set Defining in-zoomed

The Integration Processor Set Defining allows all Integration Processing components to be defined for the Integration DAG. The output is a Processor Definition that will be used with the DAG.

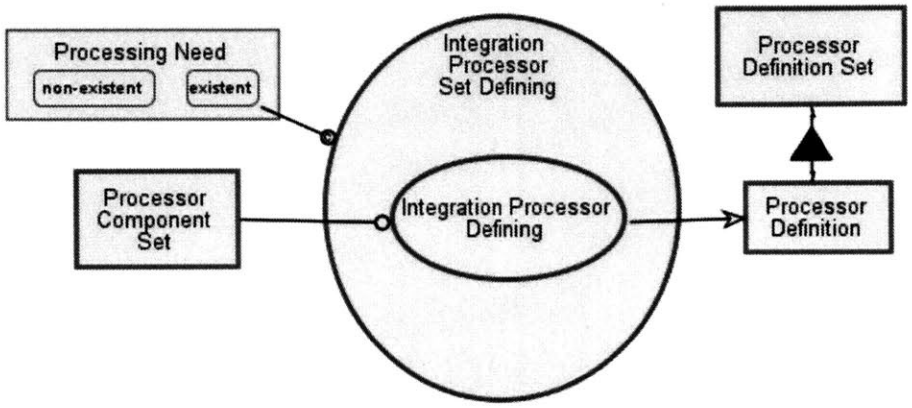


Figure 71 - Integration Processor Set Defining in-zoomed

Processor Definition Set **consists of** Processor Definition.
Processing Need **can be** existent **or** non-existent.
Integration Processor Set Defining **consists of** Integration Processor Defining.
Integration Processor Set Defining **occurs if** Processing Need **is** existent.
Integration Processor Set Defining **zooms into** Integration Processor Defining.
Integration Processor Defining **requires** Processor Component Set.
Integration Processor Defining **yields** Processor Definition.

Integration Processor Defining in-zoomed

Integration Processing Defining will be specific to the processor being defined.

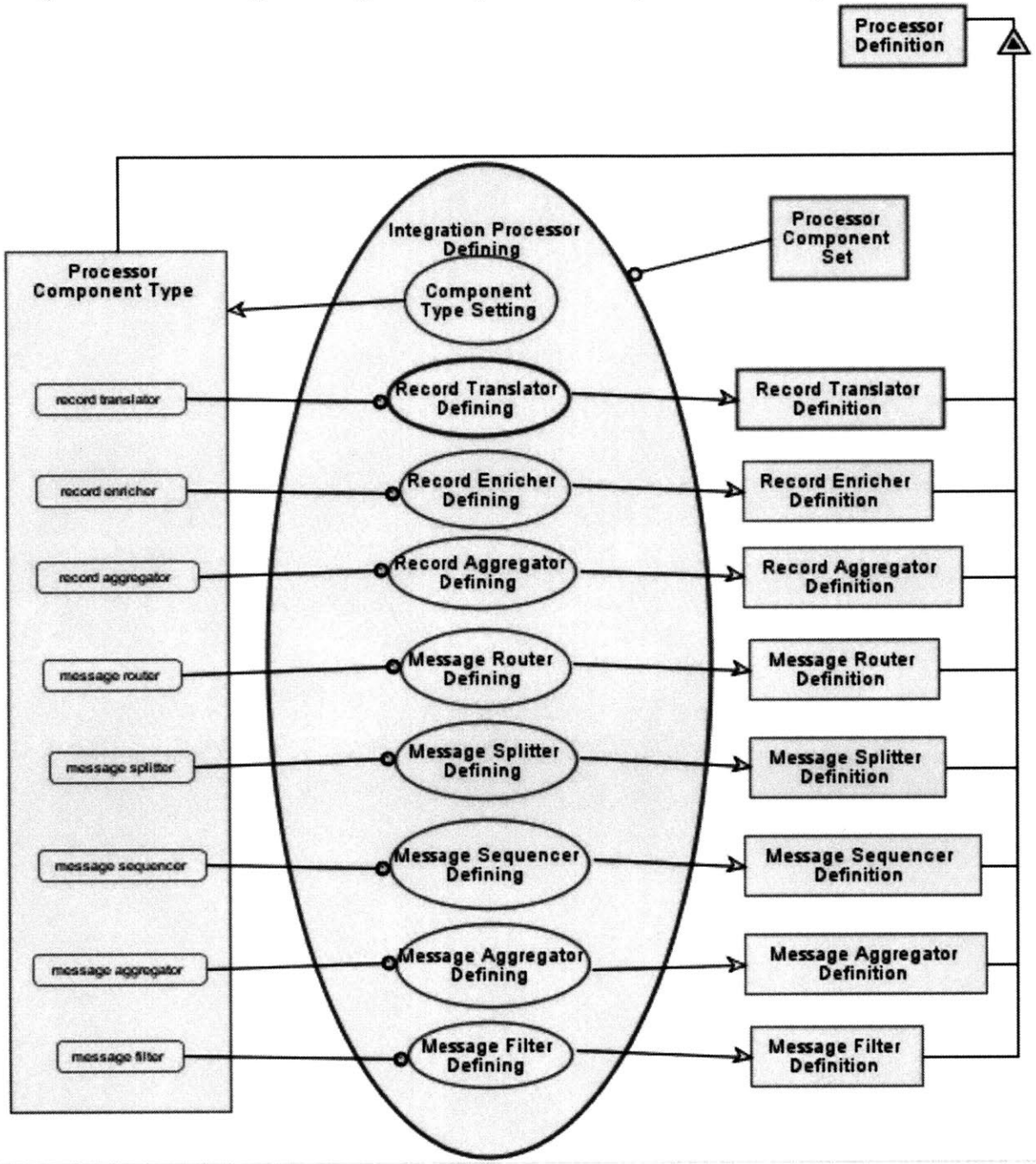


Figure 72 - Integration Processor Defining in-zoomed

Processor Definition **exhibits** Processor Component Type, Record Translator Definition, Record Enricher Definition, Record Aggregator Definition, Message Router Definition, Message Splitter Definition, Message Sequencer Definition, Message Aggregator Definition, **and** Message Filter Definition.

Processor Component Type **can be** message router, message splitter, message

sequencer, message aggregator, message filter, record translator, record enricher, or record aggregator.
Integration Processor Defining consists of Component Type Setting, Message Router Defining, Message Splitter Defining, Message Sequencer Defining, Message Aggregator Defining, Message Filter Defining, Record Translator Defining, Record Enricher Defining, and Record Aggregator Defining.
Integration Processor Defining requires Processor Component Set.
Integration Processor Defining zooms into Component Type Setting, Record Translator Defining, Record Enricher Defining, Record Aggregator Defining, Message Router Defining, Message Splitter Defining, Message Sequencer Defining, Message Aggregator Defining, and Message Filter Defining.

Component Type Setting yields Processor Component Type.

Record Translator Defining occurs if Processor Component Type is record translator.

Record Translator Defining yields Record Translator Definition.

Record Enricher Defining occurs if Processor Component Type is record enricher.

Record Enricher Defining yields Record Enricher Definition.

Record Aggregator Defining occurs if Processor Component Type is record aggregator.

Record Aggregator Defining yields Record Aggregator Definition.

Message Router Defining occurs if Processor Component Type is message router.

Message Router Defining yields Message Router Definition.

Message Splitter Defining occurs if Processor Component Type is message splitter.

Message Splitter Defining yields Message Splitter Definition.

Message Sequencer Defining occurs if Processor Component Type is message sequencer.

Message Sequencer Defining yields Message Sequencer Definition.

Message Aggregator Defining occurs if Processor Component Type is message aggregator.

Message Aggregator Defining yields Message Aggregator Definition.

Message Filter Defining occurs if Processor Component Type is message filter.

Message Filter Defining yields Message Filter Definition.

Summary

As part of this thesis, systems thinking, the Object-Process Methodology and OPCAT were used to analyze, define and document opportunities for improvement for data integrating software including:

1. Extensibility by third parties
2. Different types of structured data models
3. Enhanced modeling capabilities
4. Automated unit testing
5. Unified toolset that covers multiple integration patterns
6. Web-based toolset

In addition an OPM model was created and described, which is being used to facilitate design and implementation of a data integrating system for a large client that will use the system to integrate data between their central master data management solution and all other operational systems within their business.

Lessons Learned

In addition to the lessons learned from above surrounding opportunities for improvement in data integrating systems, there were also lessons learned centered around the use of modeling, OPM and OPCAT in a custom software building project.

- **Modeling was extremely valuable** in facilitating design discussions throughout the project, especially as the software was incrementally designed over time. With agile methodologies focuses on design and construction in a rapid and iterative fashion, design decisions can be lost and rehashed over the period of evolution of the project. Having the model to constantly refer back to was extremely helpful

- With OPM's one consistent syntax and diagram type, **it was quick and easy to get a team of people up to speed on the modeling syntax**. None of the other team members had been exposed to OPM, and within a very short timeframe the team was able to understand and read the models.

- OPM and OPCAT's **single integrated model** helped facilitate efficient design discussions. Being able to drill up and down throughout the entire model for viewing purposes without flipping back and forth between diagrams and diagram types made it easier to keep train of thought on the problem domain versus navigability between diagrams and diagram types.

- **Model starting point, from top or bottom** – As the Data Integrating System model was being defined, there were a couple of times I wanted to model a component or sub-component from the bottom up, and I struggled as OPCAT

seems to guide you down a path and more easily navigate from the top down. I believe both approaches are appropriate depending on the situation. There are times I wanted to build a bottom up model and then tie it into existing top down model, and it wasn't easy to do so.

- **Model boundaries** - It would be helpful for me to define a list of criteria for what to model and what not to. I think it would be theoretically possible to model every aspect of a software system down to the nitty-gritty detail; however, this probably isn't an efficient use of time, unless code base is truly generated from the model itself.

- **Ease of refactoring** –There were a couple of times as I was using OPM to think through the problem domain, that I reversed directions and needed to do some fairly extensive refactoring of the model. This was more difficult than anticipated as each of the model diagrams is coupled to the diagrams above and below it (i.e. in-zoomed or unfolded). The coupling between the diagrams is excellent for consistency and discipline, but provides additional challenges when refactoring.

- **Use of OPM to document an existing system** – There are many times I have been thrown into a situation where the need existed to understand a system that someone else put together. In many cases there is no documentation, and the initial designer or developer are no longer available. Having the ability to reverse engineer existing code into an OPM model would be an extremely valuable tool in understanding existing systems.

- **Data Modeling - complexities in the level of abstraction** – When defining a relational model with OPM and OPCAT, I found myself struggling with the level of abstraction at times. When defining a relational model in an Entity-Relationship diagram or within a database system itself, I know exactly what the database structure will look like. When defining that same relational model in OPCAT, I found myself struggling to determine how OPCAT would translate the model into a relational structure. The level of indirection / abstraction added complexity to the task.

- **Data modeling – required vs. optional** – When creating a data model in OPM, how do you define optional or required, field lengths, etc. These can be done in OPL but not necessarily generate the correct SQL. Same thing with indexes, constraints, etc. A complete meta model for data modeling would be most beneficial.

- **Visualization from a granularity perspective** – There is an inherent contradiction between simplicity and being able to see the big picture of a system. When creating several of my model diagrams, I struggled with whether to include multiple levels of aggregation – participation in a single diagram or whether to in-zoom or unfold each level. I tended to stick with seven to ten

things on a diagram for simplicity sake, but found this does prohibit the ability to get a better big picture look at a topic or set of items.

- **Nomenclature for things with the same name but under different things or contexts** – What should be done about nomenclature for items that are named the same, but have different things or contexts? In a programming language this could be resolved by providing package names for the things that would fully qualify the thing and its attribute.

Future Work and Research Opportunities

Future work can be grouped into three categories. The first is to continue working on the existing integrating system project. The model for the integrating system will continue to be defined and refined in the context of the project with product development happening concurrently. The Integration Executing and Integration Monitoring processes will be flashed out in detail, and items will continue to be added to the Integration Defining process, including additional reader, writer and processing component types.

In addition to the integrating project activity, there is a second body of work centered around enhancements to OPCAT that would be interesting and beneficial, including:

- **Web-based version of OPCAT** – With the advancements in web technology, it would be possible to create a web based version of OPCAT that could be utilized over the internet and with a web browser. This would provide opportunity for enhanced group based work and review of a model over a geographically dispersed audience.

- **Additional refactoring capabilities** – As discussed in the lessons learned, refactoring is possible with OPCAT, but there are opportunities to enhance the refactoring process to make it more efficient.

- **Ability to create an OPM model from code and vice versa** – The ability to reverse engineer a model from a code base would be extremely valuable. While this is a complex task, it would be a good future research opportunity as well as body of work to enhance the OPCAT toolset.

- **Ability to create an OPM model from an SQL database and vice versa** – Currently OPCAT has rudimentary capabilities to turn a model into SQL, but no capabilities to reverse engineer a model from a database. This would be another opportunity for improvement.

- **Enhancements to printing and exporting of an existing model (to pdf, etc.)** – Ability to provide a printable version of a model or sections of a model would also be extremely helpful in facilitating design sessions.

Future research in many cases could fall in line with the future work listed above. Of particular interest is the ability to reverse engineer a model from an existing code and database structure. In addition, the definition of metamodels for relational databases, graph databases and other could lay down a foundation for common OPM models that could be used across projects and models.

Bibliography

Akbay, S. (2004, June). Model-Driven Data Integration. *DM Review* , 65-66.

Bernstein, P., & Hass, L. (2008). Information Integration in the Enterprise. *Communications of the ACM , 51* (9), 72-79.

Brands, K. M. (2014, December). TECH Practices - Data Visualization and Discovery. *Strategic Finance* , 56-57.

Campbell, A., Goulson, G., & Kounavis, M. (1999, October). Managing Complexity: Middleware Explained. *IT Pro* , 22-28.

Halevy, A., Rajaraman, A., & Ordille, J. (2006, September). Data Integration: The Teenage Years. *Proceedings of VLDB* , 9-16.

Hohpe, G., & Woolf, B. (2004). *Enterprise Integration Patterns.* Boston, San Francisco, New York: Addison-Wesley.

Kabiri, A., & Chiadmi, D. (2013, August). Survey on ETL Processes. *Journal of Theoretical and Applied Information Technology* , 219-229.

Khamis, A.-l., Zhong, L., & Gon, H. (2014). Study on Digital Content Representation from Direct Label Grph to RDF/OWL Langue into Semantic Web. *Applied Mechanics and Materials* , 3304-3309.

Kimball, R., & Caserta, J. (2004). *The Data Warehouse ETL Tookit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data.* Indianapolis, IN: Wiley Publishing, Inc.

Lawson, L., & Sharma, S. (2014, 12 12). *Round-up: 2015's Top Integration Trend, In One Word*. Retrieved 12 13, 2014, from IT Business Edge: http://www.itbusinessedge.com/blogs/integration/round-up-2015s-top-integration-trend-in-one-word.html

Moradi, H., & Bahreininejad, A. (2013, March 1). Toward a Comprehensive Framework for Evaluating the Core Integration Features of Enterprise Integratin Middleware Technologies. *Journal of Systems Integration* , 13-29.

Pahl, C., & Zhu, Y. (2012). Data Integration in Mediated Service Compositions. *Computing and Informatics , 31*, 1129-1149.

Stonebraker. (2002). Too Much Middleware. *ACM SIGMOD Record* , 97-106.

Varughese, D. D. (2013). Optimising Ontology Integration Through Intermediate Ontologies. *Journal of Theoretical and Applied Information Technology , 57* (3), 441-451.

Xu, Z.-y., Yan, Y., Wang, G.-x., & Gong, L. (2014). Research on the Integration Method for Heterogeneous Database Based on Ontology. *Advanced Materials Research , 933*, 675-681.

Zhai, L., Guo, L., Cui, X., & Li, C. (2011). Research on Real-time Publish/Subscribe System supported by Data-Integration. *Journal of Software , 6* (6), 1133-1139.

Zhang, H. (2014). A query Driven Method of Mapping from Global Ontology to Local Ontology in Ontology-based Data Integration. *Journal of Software , 9* (3).

Zhang, H. (2014, March). A Query Driven Method of Mapping from Global Ontology to Local Ontology in Ontology-based Data Integration. *Journal of Software* , 738-742.