University of Arkansas, Fayetteville
ScholarWorks@UARK

Theses and Dissertations

8-2012

A Domain Specific Model for Generating ETL Workflows from Business Intents

Wesley Deneke
University of Arkansas, Fayetteville

Follow this and additional works at: http://scholarworks.uark.edu/etd

 Part of the Numerical Analysis and Scientific Computing Commons

Recommended Citation
Deneke, Wesley, "A Domain Specific Model for Generating ETL Workflows from Business Intents" (2012). *Theses and Dissertations*. 547.
http://scholarworks.uark.edu/etd/547

This Dissertation is brought to you for free and open access by ScholarWorks@UARK. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu, ccmiddle@uark.edu.

A DOMAIN SPECIFIC MODEL FOR GENERATING ETL WORKFLOWS FROM BUSINESS INTENTS

A DOMAIN SPECIFIC MODEL FOR GENERATING ETL WORKFLOWS FROM BUSINESS INTENTS

A dissertation submitted in partial
fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

By

Wesley Deneke
University of Arkansas
Bachelor of Science in Mechanical Engineering, 2003

August 2012
University of Arkansas

ABSTRACT

Extract-Transform-Load (ETL) tools have provided organizations with the ability to build and maintain workflows (consisting of graphs of data transformation tasks) that can process the flood of digital data. Currently, however, the specification of ETL workflows is largely manual, human time intensive, and error prone. As these workflows become increasingly complex, the users that build and maintain them must retain an increasing amount of knowledge specific to how to produce solutions to business objectives using their domain's ETL workflow system. A program that can reduce the human time and expertise required to define such workflows, producing accurate ETL solutions with fewer errors would therefore be valuable. This dissertation presents a means to automate the specification of ETL workflows using a domain-specific modeling language.

To provide such a solution, the knowledge relevant to the construction of ETL workflows for the operations and objectives of a given domain is identified and captured. The approach provides a rich model of ETL workflow capable of representing such knowledge. This knowledge representation is leveraged by a domain-specific modeling language which maps declarative statements into workflow requirements. Users are then provided with the ability to assertionally express the "intents" that describe a desired ETL solution at a high-level of abstraction, from which procedural workflows satisfying the intent specification are automatically generated using a planner.

This dissertation is approved for recommendation
to the Graduate Council.

Dissertation Directors:

_____
Wing-Ning Li

_____
Craig Thompson

Dissertation Committee:

_____
Gordon Beavers

_____
Rick Couvillion

DISSERTATION DUPLICATION RELEASE

I hereby authorize the University of Arkansas Libraries to duplicate this dissertation when needed for research and/or scholarship.

Agreed _____
Wesley Deneke

Refused _____
Wesley Deneke

ACKNOWLEDGEMENTS

I thank my advisors, Drs. Craig Thompson and Wing-Ning Li. Their tutelage provided me with guidance that was instrumental to this dissertation and their support helped keep me motivated throughout my graduate career. They have both made a profound, positive impact on my life.

I also thank Drs. Gordon Beavers and Rick Couvillion. Their advice over my academic career and assistance as committee members is greatly appreciated.

Finally, I thank my parents, James and Deborah Deneke, my wife, Vong Deneke, and my children, Scotty and Landric. The encouragement and assistance from my family during the course of this work has been a blessing. Without their support this dissertation would not have been possible.

Thank you all.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Problem

In both scientific and industrial communities and organizations, there is an ongoing need to employ sophisticated data processing techniques in order to manage and leverage the torrent of data created by modern information age technology [1]. This need has become so pervasive that the technical community has identified "big data" as a critical technical problem [2], [3], [4]. Scientists generate massive data sets from experiments and sensor feeds; retail stores track high volumes of customer sales and orders to vendors; and businesses record and maintain huge tables full of customer data.

Beyond simply retaining this data, organizations want to extract patterns or answers to queries to mine actionable information from the data. As illustrated in the system depicted in Figure 1, existing techniques for attaining such objectives often aggregate data from many heterogeneous data sources: databases, flat files, hierarchical files, and sensor feeds. These data sources are often large (e.g. billions of records). Commonly, their schemas change over time. The data sources may also be distributed: not stored in just one database management system or controlled by a single organization. Ultimately, the set of data processing tasks applied to the data from these diverse data sources is based on an organization's specific objectives.

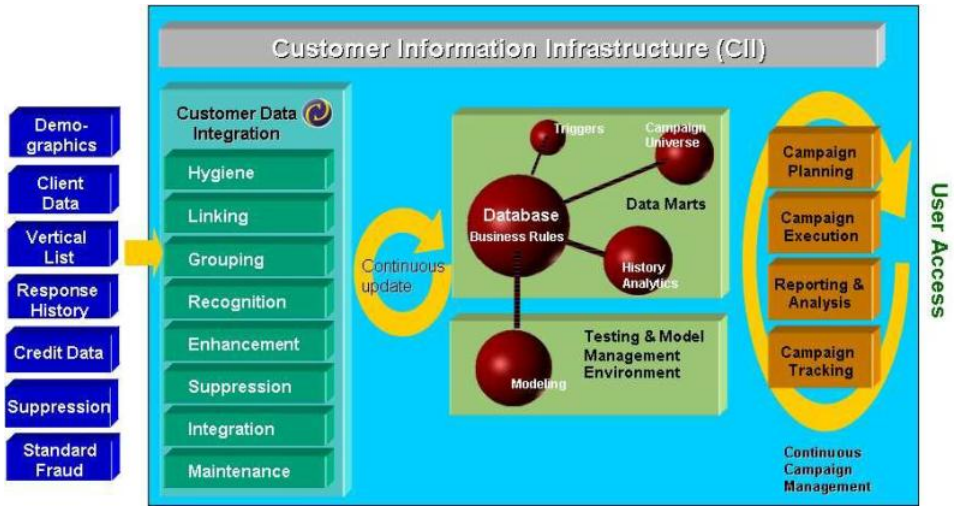

Figure 1: Depiction of a system for data processing (courtesy of Acxiom).

A widely adopted methodology for designing and executing such data processing tasks as reusable batch jobs is known as Extract-Transform-Load (ETL). In the ETL methodology, a solution is represented as a workflow wherein data sets flow from one node to another within a large directed graph of operations, starting with one or more data sources and ending with one or more data targets. An ETL workflow consists of the following elements:

- *Operators* – Reusable constructs that encapsulate the available data operations.
- *Data fields* – Individual data values that are extracted from data sources, flow in and out of operators, and are loaded into data targets.

Using these constructs, the workflow defines the data flow and the manner in which the data is processed. Given the properly constructed workflow graphs, commercial ETL workflow systems are able to produce solutions to data processing goals, which in turn provide solutions to business objectives. Several commercial ETL workflow systems (known as ETL Tools) are available today, including IBM InfoSphere DataStage [5], Microsoft SQL Server Integration Service [6], and Oracle Data Integrator [7].

The problem this dissertation addresses involves the construction of the input workflow graph. Currently, the specification of an ETL workflow is a manual process, involving a series of sequential phases that are conceptually similar to the Waterfall development model [8] (see Figure 2).

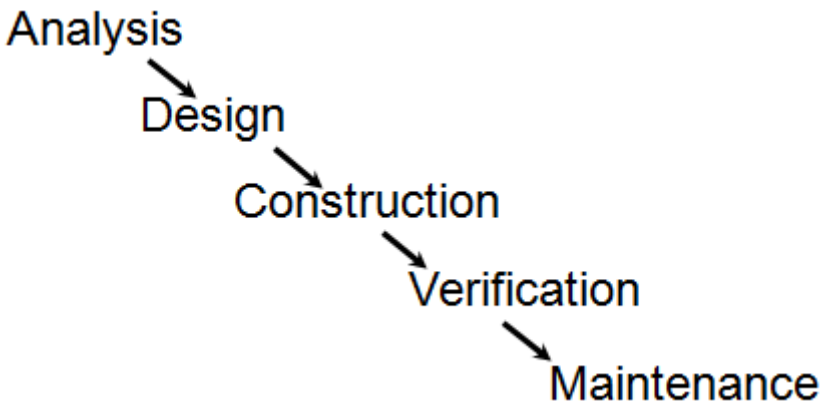

Figure 2: Phases of the Waterfall development model.

- *Analysis* – Define requirement specifications for the workflow to be developed, such as necessary behavior and imposed constraints. The requirements are typically gathered from the stake holders (aka non-expert users) who may only be able to convey the intended purpose in a high-level manner. In such cases, the technical requirements must be derived from the high-level business objectives in a manner appropriate within the context of the given domain.
- *Design* – Determine a workflow design that fulfills the requirement specifications. This must take into consideration the data sources available, the characteristics of this data, and the available operations in order to ascertain a plan for the solution based on the capabilities of the workflow system.
- *Construction* – Define the workflow's composition: the operators used, their sequence, the mapping of data fields between these operators. Today, construction is performed

directly through XML specification or ETL tools. XML specification of an ETL workflow involves encoding the operator calls and data field mappings into a XML-formatted file manually with a text editor (see Figure 3). Alternatively, ETL tools may be used, providing users with a visual editor that abstracts the underlying XML specifications. In these tools, workflow solutions are represented like flowcharts (see Figure 4). Operations are specified by dragging and dropping a desired operation's visual object into the solution's sandbox area. Data flow is mapped by individually connecting the appropriate data fields between the nodes of the workflow.

- *Verification* – Methodically test using sample data sets to ensure the workflow behaves according to the specified requirements. Verify absence of operator failures at run time due to improper mappings.
- *Maintenance* – Modifications to an existing workflow to include additional requirements.

```
<?xml version="1.0" encoding="utf-16"?>
<ProgramInterface xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <i_prj_c xmlns="http://www.WS.Request.com">PRJC</i_prj_c>
  <i_user_id xmlns="http://www.WS.Request.com">USERID</i_user_id>
  <i_flt_c xmlns="http://www.WS.Request.com">FLTC</i_flt_c>
  <i_div_c xmlns="http://www.WS.Request.com">DIVC</i_div_c>
  <i_rt_c xmlns="http://www.WS.Request.com">RTC</i_rt_c>
  <i_md_c xmlns="http://www.WS.Request.com">MDC</i_md_c>
  <i_cmd_c xmlns="http://www.WS.Request.com">CMDC</i_cmd_c>
  <i_svc_lvl_c xmlns="http://www.WS.Request.com">SVCLVLC</i_svc_lvl_c>
  <i_crr_c xmlns="http://www.WS.Request.com">CRRC</i_crr_c>
  <i_prl_f xmlns="http://www.WS.Request.com">PRLF</i_prl_f>
  <i_prl_eqp_unt_pfx xmlns="http://www.WS.Request.com" />
  <i_prl_eqp_unt_i xmlns="http://www.WS.Request.com" />
  <i_eqp_sub_cls_c xmlns="http://www.WS.Request.com">EQPSUBCLSC</i_eqp_sub_cls_c>
  <i_eqp_sub_cls_rqd_f xmlns="http://www.WS.Request.com">F</i_eqp_sub_cls_rqd_f>
  <i_eqp_len xmlns="http://www.WS.Request.com">LEN</i_eqp_len>
  <i_eqp_len_rqd_f xmlns="http://www.WS.Request.com">F</i_eqp_len_rqd_f>
  <i_eqp_typ_c xmlns="http://www.WS.Request.com">EQP</i_eqp_typ_c>
  <i_eqp_typ_rqd_f xmlns="http://www.WS.Request.com">F</i_eqp_typ_rqd_f>
  <i_tot_mil_cnt xmlns="http://www.WS.Request.com">MIL</i_tot_mil_cnt>
  <i_tot_pcs_q xmlns="http://www.WS.Request.com">PCS</i_tot_pcs_q>
  <i_tot_wgt xmlns="http://www.WS.Request.com">WGT</i_tot_wgt>
  <i_tot_vol xmlns="http://www.WS.Request.com">VOL</i_tot_vol>
  <i_nbr_stps xmlns="http://www.WS.Request.com">STPS</i_nbr_stps>
  <i_bil_cus_c xmlns="http://www.WS.Request.com">BIL</i_bil_cus_c>
  <i_sol_cus_c xmlns="http://www.WS.Request.com">SOL</i_sol_cus_c>
  <i_freight_stop_record xmlns="http://www.WS.Request.com">
    <i_stp_seq_nbr>1</i_stp_seq_nbr>
    <i_apt_beg_d>2010-08-04</i_apt_beg_d>
    <i_apt_beg_h>00.01.00</i_apt_beg_h>
    <i_apt_end_d>2010-08-04</i_apt_end_d>
    <i_apt_end_h>23.59.00</i_apt_end_h>
    <i_apt_typ_c>P</i_apt_typ_c>
```

Figure 3: A partial XML specification for an operator call in a workflow.

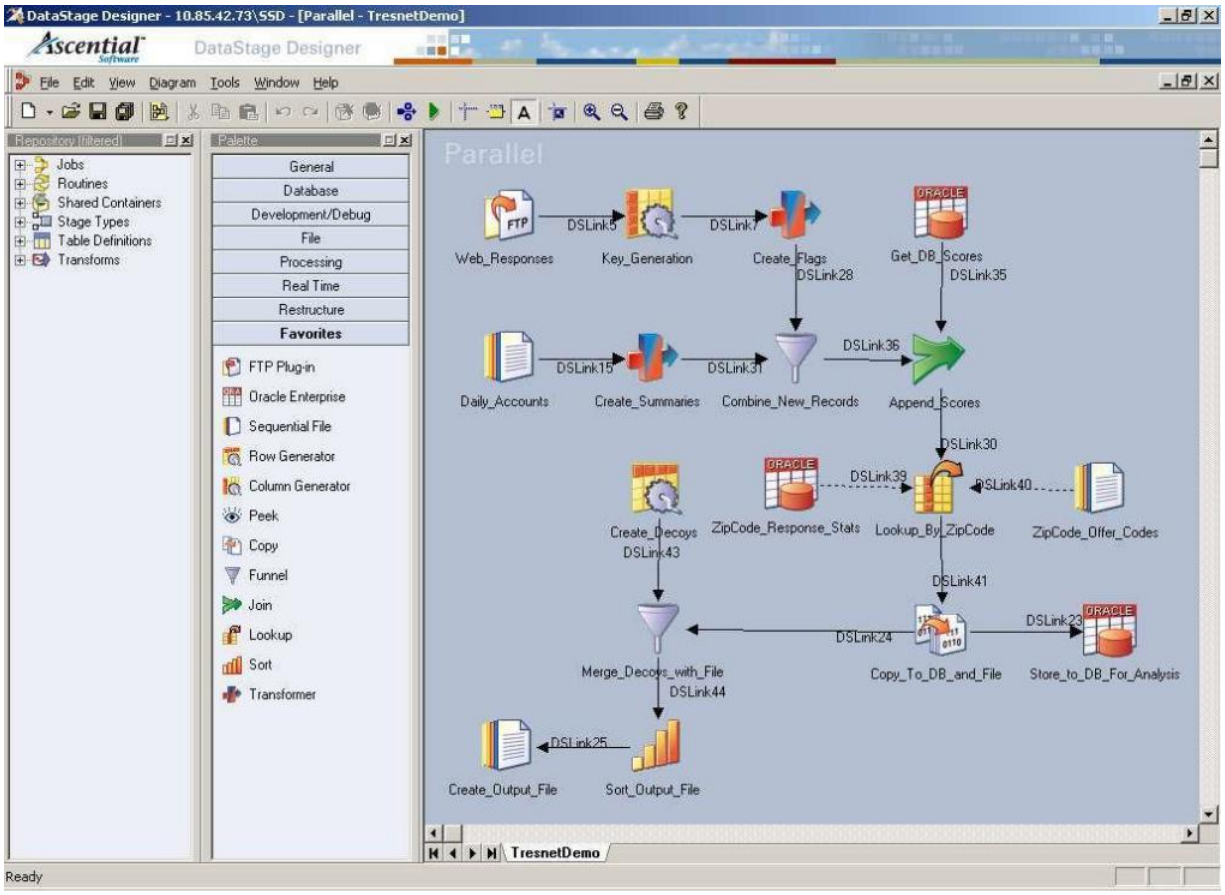

Figure 4: Using flowchart tools to specify a workflow.

As mentioned, ETL workflow specification is performed manually. While this may be sustainable for domains that deal with small, easily manageable data sets, it can become overwhelming for ETL domains that involve managing massive, distributed data sets, producing workflows that contain thousands of data fields that must all be mapped to flow properly through workflow that may require hundreds of operators. As the scale increases, there are so many considerations (e.g. which operators to include/exclude, what sequence of operators, which fields to map where) that the reliance on a human user becomes increasingly problematic. For workflows that involve many operators and data fields, a user performing XML specification can be required to type thousands of lines for a single solution attempt; an onerous task for a human. Even with today's drag-and-drop ETL workflow visualization and construction tools, the user is

responsible for selecting the proper set and ordering of operators, mapping each operator's and data source's/sink's parameters, and ensuring that each applicable requirement is considered. While feasible for simple workflows, performing such tasks manually for workflows with dozens of operators and hundreds parameters is time consuming and can take days or weeks.

In response, some approaches may hard code domain knowledge into macros or standard workflow templates. Similar to subroutines, macros and standard workflow templates may be defined and then later called, creating an invocation instance (particular call that binds a particular set of inputs, parameters, and outputs). In this way, commonly used sequences of operators can be mapped together and bundled as a reusable package. However, such solutions suffer from poor extensibility and must be continuously reconfigured.

Another aspect of the problem is the latent significance of domain knowledge in each phase of the specification process. ETL workflow specification often relies on the human user to retain virtually all of the knowledge necessary to produce a solution. For instance, to extract technical requirements from the business objectives described by stake holders, the user must be a domain expert: a user with a complete understanding of a given domain's workflow system and the solutions it is used to develop. Such expertise can also be required for the construction of ETL workflows; operators may have pre/post requisites or constraints that invalidate the workflow (cause runtime failures or fail to produce the expected result) if violated. However, it takes a considerable amount of time to train a domain expert. This domain knowledge is a challenge to identify and capture; much of it has been gained through experience, manifesting as intuition during workflow specification, thus making it difficult to impart the knowledge to others.

A final problem with the existing process is that specification of complex ETL solutions is error prone, requiring iterative approaches at a solution until all of the requirements are met. Integrating critical knowledge only from memory, human fallibility often results in erroneous workflows that fail to fully satisfy the requirements. Data processing goals may have numerous interpretations, making the requirements ambiguous. Consequently, manually composed workflow solutions must often undergo multiple iterations of modifications until a valid solution can be generated. When utilizing manual approaches, refactoring a workflow can become as lengthy as the construction of the original workflow. As a result, building and maintaining ETL workflow solutions can be a heavy investment of resources in terms of human time to specify workflows and processing time to process billions of records through a graph of complex transformations.

The problem that is the focus of this dissertation is how to automate the process of ETL workflow specification. No automated solution that can flexibly represent and enforce the necessary knowledge currently exists. Therefore, a program that can flexibly integrate domain knowledge to automatically generate domain-specific ETL workflow solutions from a set of processing goals would be provide significant benefits to the scientific and industrial workflow communities.

Before proceeding, it is important to note that the term domain (within this paper) is used to express a means of constraining the set of considerations. One notion is that a domain an industry, like retail, insurance, or finance. Another notion is a language restricted in functionality that handles just formatting or just matrix operations or just logic programming or just database queries.

1.2 Thesis Statement

This dissertation provides a solution to the ETL workflow specification problem by developing an extensible approach for creating domain-specific modeling languages that automate the generation of ETL workflows from business intents.

The thesis of this paper is as follows:

ETL workflow specification can be automated in an extensible manner by translating high-level statements of intent into a set of ETL workflow requirements and generating ETL workflow solutions that accomplish these specifications.

1.3 Approach

Given the limitations of existing ETL workflow specification solutions, the approach described herein seeks to automatically generate valid workflow solutions from high-level intent statements, providing:

- Better solution accuracy
- Lower required level of expertise
- Faster turn around
- Fewer errors

One key component of this approach is the model. As mentioned, ETL domain knowledge is critical to generate valid workflow solutions in the ETL workflow specification problem. For this reason, the proposed solution uses a model-based approach to capture this knowledge in a model. The constructs that implement this model provide a computational representation of ETL domain knowledge, enabling tasks that require this knowledge to be handled programmatically rather than manually. This permits the other key component, a

workflow engine, to leverage the model to automate workflow construction. The workflow engine is an implementation of an AI planner, which uses the model to generate valid workflow solutions from a given set of requirements represented as high-level intents.

This approach also places an emphasis on extensibility. The data sources, operations, and knowledge involved in ETL workflow specification can vary widely between domains: some may enhance address data, others may audit billing data, and yet others may analyze experimental results. These domains can also evolve, adding/modifying/removing operators and data sources in order to accommodate ever-changing processing goals. For these reasons, extensibility is considered important for the approach to adapt to current and future needs of ETL domains.

1.4 Organization

Chapter 1 defined the problem, objective, and approach of this dissertation. Chapter 2 provides background and key concepts associated with workflow, ETL, domain-specific languages, and models, as well as existing work in related disciplines, such as the areas of artificial intelligence planning and database SQL parsers. Chapter 3 outlines the conceptual design of the approach taken by this work to solve the ETL workflow specification problem. Chapter 4 describes implementation details of a prototype of the suggested approach. Chapter 5 covers the testing methodology, experimental results obtained from the prototype, and analysis of these results. Chapter 6 summarizes the conclusions, describes the contributions of this dissertation, and introduces areas of future research on the topic of ETL workflow specification. Finally, the appendices provide the operator and intent specifications used during testing, the expected workflow solutions for the test scenarios, and the workflow solutions generated by the prototype.

2. BACKGROUND AND RELATED WORK

This chapter provides background and related work on concepts relevant to the approach taken in this work. Section 2.1 describes the central concept workflow, focusing on extract-transform-load workflows. Sections 2.2 through 2.4 provide background from related areas: general purpose and domain-specific programming languages, relational databases, and artificial intelligence planning systems. Section 2.5 provides a literature survey on related work on automating workflow and plan generation.

2.1 Workflow and ETL

This section introduces the concept of workflow and describes the role workflow plays in delivering ETL solutions.

2.1.1 Workflow

Workflow is a concept that extends beyond the context of this problem domain, having a variety of specialized interpretations in differing contexts. Workflows are commonly used in a number of areas, including: employee procedures, AI planners, query optimization, web service composition, scientific data processing, business processes, and ETL. To generalize Aalst and Hee [9], a workflow is a conceptual model of a process. At a very high level, this model is used to describe the work that must be performed to achieve a particular result. Though abstract, work can be conceptualized as taking place as a series of changes, which in turn can be represented by a sequence of tasks and states. From this, a workflow can be described as a set of tasks connected in a specific sequence that may be represented in a sequential or parallel graph of tasks.

As an example, consider the work performed by a person traveling from their home to the airport. A person would need to prepare to leave their house and then drive to the airport. A workflow for this activity then could consist of a task for preparing to leave the house followed by a task for driving to the airport.

Often, the sequence in which tasks are performed in a workflow may be critical. For instance, a person cannot drive to the airport until after they have prepared to leave their house. Thus, any concrete model of a workflow may have to preserve this directional dependence. For this reason, the composition of a workflow is represented as a directed graph (digraph). A directed graph G is a pair (V, E), where V is a set of vertices and E is a set of ordered pairs of vertices that denote directed edges connecting vertex pairs. A directed edge e_i from vertex v_j to v_k would be denoted as $e_i = v_j \rightarrow v_k$. With respect to depicting a workflow, the vertices denote the tasks and the edges represent the sequence in which they are performed. In this way, a directed graph can capture the relationships between each task of a workflow from start to completion. To illustrate this, consider the previous example workflow for a person traveling from their home to the airport. The graph of this workflow would consist of two vertices, v_1 and v_2 and one edge e_1. Assuming that v_1 represents the task of preparing to leave the house and v_2 represents the task of driving to the airport, e_1 would be defined as $e_1=v_1 \rightarrow v_2$ and the resultant digraph would have a single path from start to completion (shown in Figure 5).

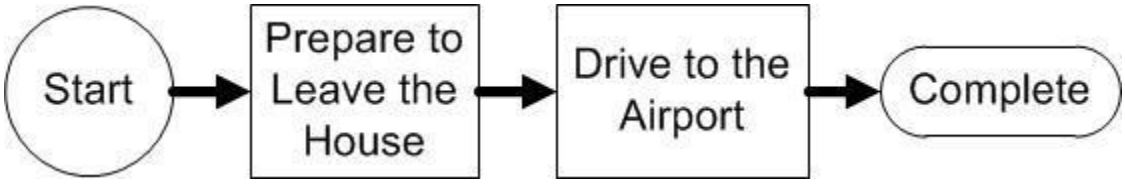

Figure 5: A simple workflow.

The composition of workflows can vary from simple to complex. This is due not only to the complexity of the process being modeled, but also the level of abstraction used to represent

the process. The work performed by a workflow may be represented at a high level of abstraction or a lower level of abstraction. In such a way, a workflow may be simplified to enhance comprehension by using more abstraction or it may be made more complex to improve accuracy by using less abstraction. This concept is comparable to the levels of abstraction present in programming languages. High-level programming languages, such as C#, are easier to comprehend as their instructions directly describe higher-level behavior. However, this at the cost of reduced control and understanding of what low-level actions these instructions produce. In contrast, low-level programming languages, such as assembly language, are able to describe a computing process in high detail as the instructions represent very basic computing actions. Consequently, programs written in assembly language are harder to comprehend as they can require many more instructions to specify higher-level behavior. In workflows, tasks are used to specify work behavior. High-level tasks may imply that how they are performed is unimportant or assume that knowledge on how they are performed is known. However, such tasks may be an over simplification of the work involved. Conversely, to provide the details necessary to reproduce a particular result, a workflow may instead be modeled using sequences of more basic tasks. Consider again the airport example. The tasks of this workflow can be considered non-specific, generalizing the steps a person follows to travel to the airport. Several actions and decisions a person must perform when preparing to leave their house can be abstracted by this single task, including: getting dressed, eating, finding the car keys, moving to the door, opening the door, etc. Thus, the workflow for a person traveling from their home to the airport could instead be considerably more complex, such as the example in Figure 6.

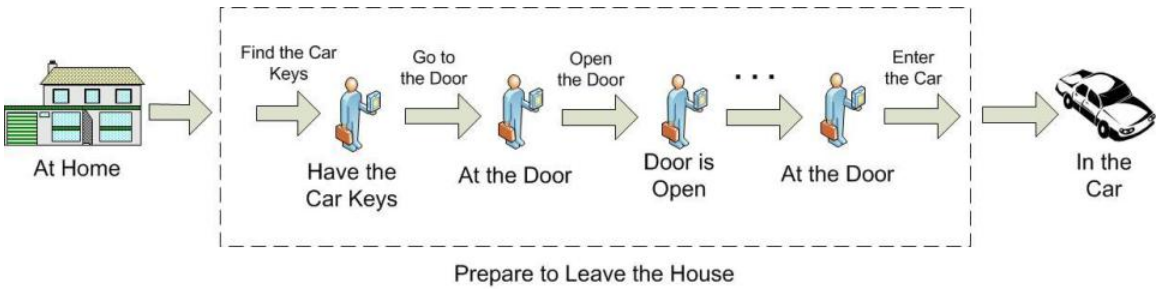

Figure 6: A second look at preparing to leave the house.

Workflows are commonly used to analyze existing processes and design new processes. These processes can range from human activities, such as the daily routine of an employee, to computing jobs, such as migrating data into a data warehouse. Yet, while a digraph can represent the composition of a workflow, this basic representation obscures some subtle details of a workflow's behavior conducive to analysis and design. Specifically, a digraph can depict what sequence of tasks to perform, but it fails to capture why the tasks must occur in this sequence. To study such behavior, a key observation is that the result accomplished by a workflow can be represented as a state: a specific set of conditions. From this, a workflow involves a sequence of state changes to attain the desired set of conditions (goal state). These changes to state are imposed by the tasks, where each task changes specific conditions in the workflow and may require specific conditions to be true before it can be performed. Using this state-based approach, workflows can be employed to advance the understanding of the processes being modeled.

This stateful representation of workflow can be depicted with a specialized form of a digraph, known as a state diagram. Formally, a state diagram S is a directed graph defined as: $S = (Q, \sum, \delta, q_0, F)$, where Q is a collection of states, $\sum$ is a collection of valid input symbols (known as an alphabet), and δ is a collection of mappings between an ordered pair of states and an input symbol that denote a state transition between the paired states caused by the input. A

state transition δ_i for the input symbol $\sum_t$ from state Q_j to Q_k would be denoted as $\delta_i : \sum_t \times Q_j \rightarrow Q_k$. In this definition, q_0 is the start state and F is the set of accepting states, where both $q_0 \epsilon Q$ and $F \subseteq Q$. Each state q_i is represented by a vertex of the graph and each state transition δ_i is represented by an edge of the graph connecting the two states for which there exists a transition. Applied to workflow, vertices represent the distinct states of the workflow and edges represent the transitions between states that available tasks, where each task $t_i \epsilon \sum$, can impose on the process.

To illustrate this, again consider the simple workflow from Figure 5. Here, it will be assumed that the person is starting at home, while the goal at the completion of this workflow is for the person to end up at the airport. Therefore, to achieve this goal the workflow must perform a sequence of tasks that moves the person from the state "At Home" to the state "At the Airport". Starting from at home, the available actions are to prepare to leave the house or drive to the airport. As a person cannot drive unless they are in the car, the first task performed must be to prepare to leave the house, after which the current state changes to "In the Car". However, since being in the car is not the goal state, another task must be performed. Now, the person cannot prepare to leave the house as they are no longer at home, but since they are in the car they can drive to the airport. The resultant state after this task is the goal state "At the Airport" and so the workflow is finished. As a state diagram, the set of distinct states $Q = \{Q_1, Q_2, Q_3\}$, where Q_1="At Home", Q_2="In the Car", and Q_3="At the Airport", the set of available tasks $\sum = \{t_1, t_2\}$, where t_1="Prepare to Leave the House" and t_2="Drive to the Airport", the set of state transitions $\delta = \{\delta_1, \delta_2\}$, where $\delta_1 = Q_1 \rightarrow Q_2$ and $\delta_2 = Q_2 \rightarrow Q_3$, the start state $q_0 = Q_1$, and the set accepting states $F = \{Q_3\}$ (refer to Figure 7).

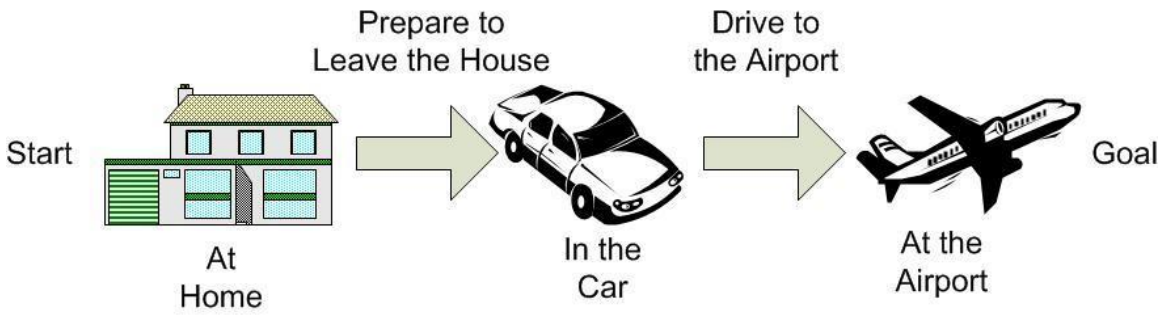

Figure 7: A state diagram for a person traveling to the airport.

2.1.2 Extract-Transform-Load Workflows

Many organizations must frequently extract data from one collection of data sources, transform the data through a complex collection of transformations (sometimes hundreds of transformations), and then load the data into target data sinks. Consider businesses that regularly move data from highly abstracted data models used in daily operations into business intelligence tables that are customized for reporting or analysis. To accomplish this, the desired data attributes must be selected and mapped to the proper columns in the target repository, including any necessary alterations. The core focus of another subset of such organizations involves building and maintaining knowledge-based models on customers from both internal and external data, requiring raw data to be pulled from diverse and commonly unreliable sources, transformed into useful customer knowledge, and persisted in a data warehouse. Extract-Transform-Load (ETL) refers to a category of data processing solutions that provide such organizations the ability to filter, clean, and persist diverse data sets. These solutions, characterized by sequences of extract, transform, and load operations, are assumed to be a sufficient and complete representation of the data processing operations necessary to direct and manipulate data from a collection of data sources to a collection of target repositories.

ETL solutions initially perform data extraction which entails pulling data from available data resources, such as databases and/or files. In particular, the data extraction phase is responsible for bringing data sets from diverse, possibly numerous resources together into a form of intermediate storage. From this representation, later data processing phases are provided a homogenous format in which all of the data may be accessed. Once data sets have been made available by data extraction processes, the data is transformed via some sequence of data transformation operations that alter all or a subset of the data using generic or domain-specific data processing logic. Generic transformations generally include: sorting, splitting, merging, translating, filtering, cleaning, validation, and enhancement. An example of a domain-specific operation is to transform a street address by adding its zip code. Using data transformations, data of heterogeneous quality levels is able to be brought to a particular standard and combined, while erroneous, unusable data is able to be pruned out. Hundreds of data transformations may be required. Once the data is in a required state, the final phase of ETL is to load the data into a target repository, which may include a file, database, data warehouse, or other form of data persistence, allowing any post-ETL processes to utilize the resultant data for a variety of applications.

ETL solutions are designed and implemented as workflows. ETL workflows consist of a sequence of tasks to denote the work that must be performed to achieve a particular result. In ETL, these tasks are known as operators. These operators are functional constructs that represent the specific processes used to extract data from sources, transform the data to meet operational needs, and load the data into repositories. These processes operate on data that is organized into a collection of logical partitions, known as data sets. These data sets consist of an unordered collection of structured data items known as data records. The structure of each data record is

defined by the data field schema of the containing data set. That is, each data record in a data set consists of the same collection of data fields, each with a data type and unique name.[1] Individual data values are thereby accessed by addressing the name of the containing data field for a given data record within the specified data set. From this, the directed edges of the graph of an ETL workflow represent the directional flow of data through the sequence of operators.[2] To illustrate this representation, consider a workflow that must pull raw name and address data from a customer file, clean it up, enhance it, and then load the useful data records into a data warehouse while capturing any pruned records in an output field, as depicted by the example in Figure 8.

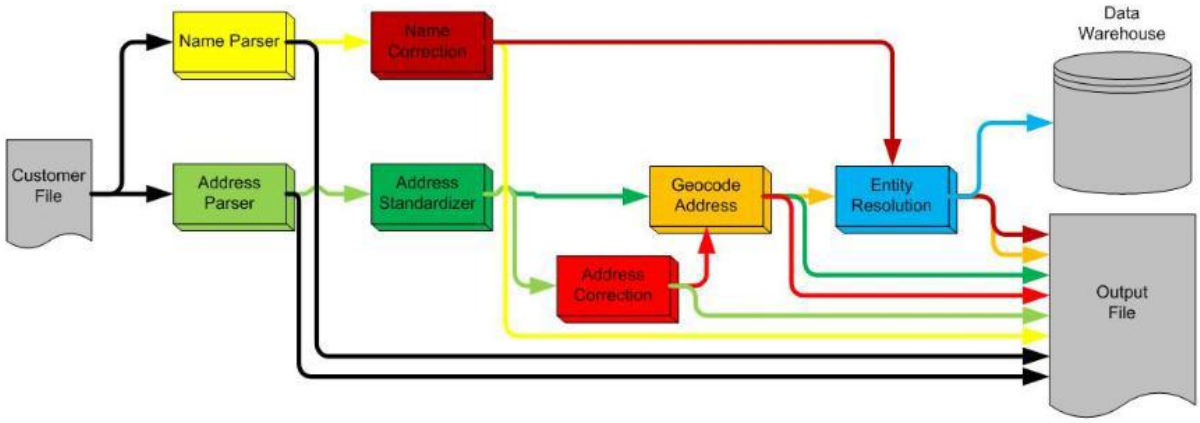

Figure 8: A complex workflow for data integration

Concerning the representation of operators, each operator is defined in executable modules to abstract the procedural logic and environmental details (client, server, grid, etc.)

[1] Conceptually, a data set containing records containing fields is strongly analogous to a relation in a relational DBMS that contains tuples that contain fields. Field names equate to attribute names. Values come from a domain.

[2] It is important to note that the graphs of ETL workflows are acyclic. For this reason, ETL workflows are more accurately described as directed-acyclic graphs (DAGs), where starting from any vertex v in a graph of an ETL workflow, there exists no path that ends up back at v after following any sequence of directed edges. This constrained structure ensures that operators are only performed a definite, pre-defined number of times during workflow execution, allowing ETL workflows to be run as reusable batch jobs instead of indefinitely persistent processes.

required to perform its prescribed data processing operation. For example, a common family of operators utilizes domain-generic logic to perform relational algebra-like transformations, splitting data into different sets based on user criteria (like SELECT), merging data sets on shared attributes (like JOIN), or performing sorting and aggregation (like ORDER BY, COUNT, etc.). Other operators may integrate domain-specific logic to influence operations such as cleaning, filtering, enhancing, and validating data based on the particular needs of the domain. Once defined, operators are independent, reusable modules that are not invoked until called during workflow execution, similar to a method or web service call, allowing operators to be used repeatedly. This enables organizations to create libraries of operators that encapsulate their standard data processing operations.

In addition to this underlying logic, each operator includes a collection of input fields and options, taken as input prior to workflow execution, and a collection of output fields are returned post-execution. Input fields are used to map which data fields from each data record in a data set that an operator should use when performing its data processing operations. These are specified through a mapping of a data field to an input field with a like data type. Options, on the other hand, govern the exact behavior of an operator, altering, for example, the value of an internal case statement in such a way that output of an operator is modified.[3] Once invoked, operators perform their defined data processing operations on the specified subset of data and return a collection of output fields. These return values denote data fields that were generated or altered as a result of operator execution. To illustrate, an example operator that performs the parsing of name data depicted in Figure 9.

[3] An option value may be part of a predefined set of values or it may be a variable such as a file name.

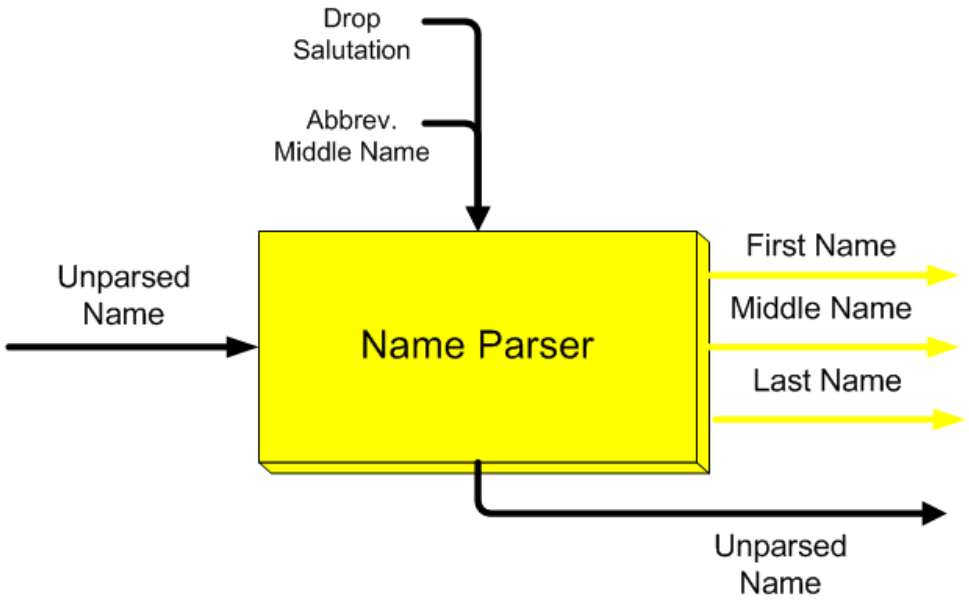

Figure 9: A depiction of a simple operator

In practice, ETL workflows are built by a process of interconnecting operators known as workflow specification. This process exists because there is no one universal workflow solution for all purposes. Instead, workflow specification is performed to find a way to build the sequence of operators for a given domain to meet their particular needs. Thus, the end result of workflow specification is to produce a workflow that maps (dirty, unenhanced) fields from data sources to (clean, enhanced) target repositories.

2.1.3 Lifecycle Phases of Workflow Specification

As mentioned in Chapter 1, workflow specification begins with an analysis phase. Adding more detail, during this phase, the desired behavior of the workflow to be developed is determined. These specifications would be gathered from stakeholders and would map to all of the technical requirements needed to build a workflow, indicating:

- data sources to include

- fields to extract from these sources
- operators to include or exclude
- field mappings, option mappings, and sequence of the included operators
- what fields to persist
- where to load the data for persistence

The people performing ETL workflow specification are typically those with extensive domain knowledge, known as domain experts. These domain experts are familiar with pertinent details such as the operators of the organization and the data of the particular domain. A stakeholder, which is a person or organization that has a vested interest in the outcome of the workflow solution, can be an external client or a business user that understands the business objectives and can describe (in a high-level manner) the intended purpose of the workflow, but lacks the technical expertise necessary to supply all of the necessary requirement specifications. The technical requirements must therefore be derived from this high-level description, prompting the stakeholder for additional information to resolve any ambiguous goals.

Also handled during this first phase of workflow specification is analysis of the characteristics of data sources. The data characteristics (e.g. layout, format, and quality) may be known and homogenous when pulled from trusted sources, but when pulled from a new or unreliable source (such as a file or other data source provided by the client), such characteristics are typically unknown. These data characteristics are considered requirement specifications as they influence the composition of the workflow. Selecting the appropriate operator to use to extract fields from a particular data source requires that the format of the data source is known (hierarchical file, flat file, relational table); reading the appropriate fields from a data source into a workflow requires that the layout of a data source is known; and determining what

transformations need to be performed on the fields to produce the desired result requires that initial quality is known (to a point).

The next phase of workflow specification is design. Consider that there can be many ways to build a workflow that fulfills the requirement specifications gathered during the analysis phase. Some may be difficult to distinguish, differing only by slight differences in field / option mappings or operator sequence. Others may include extraneous operations that exceed what is necessary to satisfy the requirements. The aim of the design phase is to produce a high level plan for a workflow based on the capabilities of the workflow system: which operators must be included and which can be excluded; which options require a specific setting and which can be ignored or defaulted. This reduces excess considerations during construction, which in turn is conducive to producing an accurate solution.

Next is the construction phase, during which the full definition of the workflow is specified. The decisions involved in the construction of an ETL workflow can rely heavily upon domain knowledge. There may be several operators available that perform similar operations or the requirements may necessitate that a custom operator is defined. Some operators may need to be performed in a specific order, while others can be performed at any time throughout the workflow. Further, there can be several ways to sequence the operators each of which may produce a different result. There may also be numerous fields in the data set that match the available input fields, where some come from the outputs of previous operators while others may come from the original data sources. A single change to the mapping of a single operator's input fields, however, can affect the result of the workflow. Similarly, the options of operators can be set in a multitude of permutations, each of which can affect the result of the workflow. Finally, selecting the appropriate sequence of transform operators and properly mapping each of its

parameters (parameters referring to the input fields and options) can be constrained by operational consideration, including:

- *Operator logic constraints* – The programmatic logic that implements ETL operators can constrain their usage. Specifically, this underlying logic can be based on assumptions about the data mapped into the operator's parameters. Failure to conform to these assumptions can cause the operator to error off or produce invalid results. There are numerous ways for such constraints to be embodied. As one example, consider an operator designed to parse name data expects a field with name data as input. If a field with zip code or phone number data is mapped, however, the expected result (a parsed name) cannot be guaranteed after execution. The logic can similarly restrict the data quality or format of input fields, such as an operator that performs geocoding assuming that the address data of input fields is spelled correctly. These constraints on operator parameters can also be conditional. Though ETL operators commonly have multiple parameters, and thus have numerous possible permutations of parameter mappings, only a fraction of these mappings may be valid. For example, from among multiple available input fields, it may only be valid to map only certain combinations of these input fields. This can be further influenced by the settings of options.
- *Workflow standards* – "Rules of thumb" that are the accepted practices for a domain in order to ensure consistent workflow solutions. These standards provide guidelines for the many decisions required when designing or altering an ETL workflow, including operator usage, workflow content, and field selection. Standards for operator usage may include pre/post requisite operators, such as always following operator A with operator B. Standards for workflow content may include the insertion of common operators, such as

an auditing/debugging operator at the end of the workflow. Finally, standards for field selection may be based on customer preferences or default to always choosing the "most recent" (in the context of the workflow) fields.

- *Business rules* – Constraints on workflow composition that are imposed from outside the domain. Business rules ensure that a domain's workflow solutions are in compliance with pertinent organizational, local, national, and even global guidelines. For example, legally imposed rules, such as contracts and laws, can restrict how data is used and distributed, both physically and digitally. Therefore, workflow design must be adjusted to identify and handle the subsets of restricted data in accordance with such constraints.

The final phase of workflow specification before a workflow can be considered "deliverable" (ready for production use) is verification. During this phase, the constructed workflow is tested on sample data sets. These sample data sets should be representative of the actual data the workflow is expected to process in order to verify that the workflow will behave according to the specified requirements when in production use. After analyzing the results produced from execution of the workflow on a sample data set, it may be discovered that there are missing specifications, ambiguous or contradictory goals, or operator failures at run time due to improper mappings. Depending on these results, the workflow may have to go back into construction to resolve mapping errors or even go all the way back to analysis if requirements have to be renegotiated with stakeholders.

2.2 General Purpose and Domain-Specific Languages

From heritage of programming languages, general purpose languages and domain-specific languages are relevant. General purpose languages are covered briefly in order establish

contrast with domain-specific languages, which is the approach taken in this dissertation as these languages allow users familiar with a given domain to express statements at the same level of abstraction as the problem domain.

2.2.1 General Purpose Programming Languages

General purpose programming languages include C, Java, C++, and Visual Basic. These languages have a specific syntax for specifying a collection of statements that can be executed by a computer. A set of these instructions would then be a program that can implement an algorithm. The power of a general purpose language is that they allow a person familiar with the language to solve a wide variety of problems; any problem with an algorithmic solution can be specified by the language. However, this generality comes at the cost of time. This is because these languages are broken down into low-level commands, to allow greater generalization. Many of these commands must be strung together into sub-routines to solve a problem at a higher-level of abstraction. The higher level the problem is the more complex the solution becomes.

2.2.2 Domain-Specific Languages

Domain-specific languages take an opposite approach to problem solving compared to general purpose languages. Rather than providing low-level commands to allow for a high degree of customization and to accommodate as many solutions as possible, domain-specific languages raise the level of abstraction to hide low-level details. This is accomplished by limiting the problem space to a specific, well-defined domain [10]. Given a well-defined domain, a domain-specific language should be less comprehensive, more expressive, and less redundant than a general purpose language. A domain-specific language is less comprehensive

because only concepts within its subject matter domain should be covered. The language should have excellent coverage of those concepts in the domain. Being more expressive refers to the language allowing users to express solutions for the given domain in terms of that domain. This allows users to understand what the intended purpose of a statement in a domain-specific language is. Lastly, less redundancy refers to the ability of the language to allow for a change in requirements with as little editing as possible. The smaller the redundancy, the less likely it is that errors can be introduced when changes are implemented.

Where most general-purpose languages are imperative languages, a domain-specific language can be declarative. "Declarative" refers to the ability of a language to express what is to be done rather than how to do it. SQL and Prolog are examples of declarative languages, allowing users to express what needs to be done, instead of specifying sequences of relational operations. For instance, the domain of SQL is that of relational databases and has clear, expressive statements. Those familiar with this domain can make statements in SQL without needing knowledge of computer systems.

2.3 Relational Database Management Systems

Relational database management systems (RDBMS) are relevant to this work because domain-specific modeling language for generating ETL workflows can borrow ideas from RDBMS queries, query optimization, integrity constraints, and database views. For instance, constraints on workflow composition are analogous to the concepts of integrity constraints in RDBMS and database views are also relevant to developing a high-level language that provides users with a restricted set of capabilities.

2.3.1 Relational Databases

ID	FirstName	LastName	Age
1	John	Anderson	35
2	Amy	White	22
3	Charlie	Brown	37
4	Karen	Kent	30
5	Edward	Jones	44
6	Jenny	Smith	26

Table 1: A relational database table containing personal data.

Relational database management systems provide a domain-specific language for data storage, query, and manipulation. RDBMS systems represent collections of data according to a perspective on the relational model, using the constructs of relations, tuples, and attributes to persist the structure and values of the data contents. Conceptually, the relational model assumes all data may be broken into atomic values that can be organized into a collection of domains (sets) that have relationships with one another that may be captured and represented. Mathematically, a relation is defined as a subset of Cartesian product over *n* domains of values. Each relation then is composed of a collection of elements known as a tuples that are each related by a shared set of ordered pairs known as attributes. Each tuple then is a function that maps an attribute to a data value. In relational databases, this mathematical model is refined to define relations more concretely as an unordered collection of like-structured data items, which are tuples. This shared tuple structure, known as a relation's schema, is based off of the attributes, metadata that define the unique name and data type of each element in a tuple.

Implementations of relational databases typically utilize a visual representation of a relation known as a table, which consists of columns and rows to represent sets of data elements (refer to Table 1). These basic relational database concepts are enhanced by or used to define additional concepts including derived relations, constructs to enforce data integrity, and a declarative means for data manipulation and retrieval. For more on relational databases, see [11].

2.3.2 SQL and Relational Algebra

Structured Query Language (SQL) is a database query language designed for retrieving and maintaining the data stored in relational databases [12]. Like other programming languages, statements in SQL can be given manually on the command-line or they can be embedded into a program that submits them at run time or during execution. SQL differs from a typical programming language, however, in that it is declarative instead of procedural. A SQL user specifies the properties of the data to be retrieved instead of the algorithm required to retrieve the data. Therefore, SQL abstracts the algorithms necessary to compute the desired data, allowing users to make simple queries even after only a brief introduction to the language. It is such declarative characteristics that are desirable in a DSML for generating ETL workflows.

Statements in SQL are made up of entities known as operators, which include select, project, union, join, etc. SQL operators perform a task involving one or more relations, producing another relation as a result. Take the basic operator select as an example. The select operator is given a reference to a specific table along with a condition, which can be in the form of a logical expression. Select then returns the subset of the table that satisfies the condition, meaning it can return a fraction of the table, none of the table, or the entire table. SQL statements can also utilize what are known as subqueries, where one or more SQL statements are

nested inside an outer statement. Subqueries introduce more complexity to SQL statements, but also allow the statements to be more expressive.

Once a SQL statement is submitted, a parser is used to verify the syntax and break the statement down by keywords into separate clauses. Each clause, upon execution, produces a table, which is then input into the next clause so it may be executed. The order in which clauses are executed, however, is not predetermined. There can be many ways to evaluate the same SQL statement. Therefore, each SQL statement is translated into an expression of an intermediate procedural language known as relational algebra. The query optimizer can then convert relational algebra expressions into equivalent expressions to try and find the fastest executing expression. This process, known as query optimization, is discussed in more detail in the next subsection. As noted by Thompson [13], however, it is interesting that when SQL is parsed and translated from the high level statements into procedural specifications, this intermediate form (called the query plan) is a kind of workflow. Similar to ETL workflows, in the relational query plan operators can be generic (relational algebra operations) or custom (user-defined functions) and the operands of these operators can be concrete relational tables (data sources) or virtual relational tables (outputs from earlier operations).

2.3.3 Query Optimization

After a SQL query is translated into a relational algebra expression, it goes through a process known as query optimization. Query optimization, performed by the aptly named query optimizer, is a process of generating query execution plans and performing cost-based analysis to determine a single plan that can evaluate the relational expression as efficiently as possible. A query execution plan, which contains concrete evaluation methods for each occurrence of an operator in the associated relational expression, is simply the set of steps necessary to perform

the query [11]. Optimization must be performed on a query because, as mentioned previously, there are typically a large number of ways to execute a query and each way has varying performance. It is the job of the query optimizer to determine the most efficient way to execute the query. The optimizer, however, does not look at every plan, but just a subset. Therefore, the choice may not be the overall best, but will at least be reasonably efficient based on the sample that was examined. Then after the plan is chosen, it can be passed to the query plan interpreter, which executes the query based on the plan.

To perform query optimization, each relational expression is first represented as a tree. The inner nodes of the tree encapsulate a single relational operator and the leaf nodes represent relations. The optimizer then applies heuristics and decides how to evaluate commutative and associative operators in order to generate a tree corresponding to an equivalent relational expression. The heuristics used take advantage of algebraic equivalences and are based on simple observations, such as "joining smaller relations is better than joining large relations". For example, utilizing the heuristic just mentioned, selections and projections can be pushed through joins and Cartesian products because this will create the smallest intermediate results, reducing I/O, and thereby hopefully reducing the cost of the resultant plan. Furthermore, the optimizer may introduce pipelining by interleaving two operational phases, in order to eliminate excess I/O. Then, the actual query execution plan is formed by augmenting the trees with specific methods for computing each operation. At this point, the DBMS then utilizes statistics and a mathematical model of query execution costs to assign weights to the tree, making it possible to estimate the cost of the plan. With the cost of each plan, a final decision on a query execution plan can be determined.

While possibly beyond the scope of this work, optimization of a workflow could be an important issue. Optimization could be performed on runtime, space, or even final quality of data (based on some metric). The plan generator, however, is of definite interest as once workflow specifications are gathered, the actual workflow must be built.

2.3.4 Integrity Constraints

During the design of a relational database, certain assumptions are made about both the structural properties of the data and the behavior of transactions in order to properly model the data of the application domain. Yet in typical database applications, users are continually committing transactions, relying upon the integrity of the data when they retrieve or report on it, but may be unaware of such guidelines. Therefore, to ensure changes made to the database do not incur a loss in data integrity, relational databases employ mechanisms known as integrity constraints. Integrity constraints are, in essence, rules that govern the usage of a database in order to uphold the integrity of the data. To illustrate, think about the state of a database as it relates to data integrity. An instance of a database can be considered in a legal state only if it satisfies all of its integrity constraints, meaning the integrity of the data is intact. It therefore follows that any change to the data that violates an integrity constraint would take the database to an illegal state if allowed to commit. Integrity constraints, thereby, can be considered to define the set of legal instances (i.e. states) of the data within the data model. Relational database management systems define integrity constraints as assertions (predicates), denoting the condition(s) that must be true for an integrity constraint to be satisfied. From this, any transaction that violates an integrity constraint should be rejected by the RDBMS in order to maintain data integrity.

From this concept, there are several kinds of integrity constraints that are commonly employed by relational databases to uphold the structural attributes inherent to a data model. One type is the entity constraint.[4] The purpose of this constraint is to enforce a relation's entity integrity by ensuring that each row can be uniquely identified. To achieve entity integrity in a database, each table is required to have primary key. This primary key is a set of columns that provides a unique (non-null) identity for each row. Thus, each "entity" (tuple) would be unique. Consequently, a database with entity integrity can have no rows with duplicate values for the primary key in a given table. As an example of this concept, assume that the "ID" column in Table 1 is the primary key. Attempting to insert a new row with any "ID" value between 1 through 6 would be rejected as (if allowed) it and the row sharing it's ID value could no longer be uniquely identified, thus violating entity integrity. However, if the primary key was a composite key of the "ID" column and the "FirstName" column, a new row with an "ID" value between 1 and 6 could be valid as long as the row with the shared "ID" value does not also share the "FirstName" value.

Another type of integrity constraint is the referential constraint.[5] This constraint is based on the concept of inter-table relationships, where columns of a given table may reference columns of other tables to represent an established relationship between their rows. To ensure referential integrity, the referential constraint requires that any value an element of a given column references from another table must exist as an element of the associated column in the referenced table. In relational databases, referential constraints are implemented as foreign keys, where a referenced column must be the key of the referenced table. Thus, to maintain the relationship between tables, a given row's foreign key cannot contain a value that does not exist

[4] The entity constraint is also known as the key constraint.

[5] The referential constraint is also known as the foreign-key constraint.

as the key of a row in the referenced table. To illustrate, consider Table 2, a relational table that contains payroll data for the individuals identified in Table 1. Referential integrity states that each value in the foreign key, "EmployeeID", must exist as a value in the key column, "ID", of a row from Table 1. Currently, the tables have referential integrity as both of Table 2's referenced values, 5 and 1, have associated rows in Table 1. However, if either row, 1 or 5, were deleted from Table 1, or a row with an "EmployeeID" of 7 or greater were inserted into Table 2, referential integrity would be violated. Thus, referential constraints ensure such transactions will be rejected.

ID	Division	Salary	EmployeeID
1	Sales	20,000	5
2	IS	45,000	1

Table 2: A relational database table with a foreign key.

A third type of integrity constraint is the domain constraint. Domain constraints define the legal values that may appear in the data items of a database in order to ensure domain integrity. To this end, each column must specify a data domain, a set of values that each element in the given column can contain, where all values in a given data domain must be of the same data type. This prevents a data row from being inserted into a table if any data elements are not congruent with the defined domain. For example, drawing again on the data from Table 1, the "Age" column represents an attribute of the data entity (in this case a person) that is a measurable quantity. As such, it makes sense when defining the table to declare the data type of this column as an integer, specifying that only whole numbers are acceptable values for this column. Then, if any transaction attempts to insert or update a row with any value other than a

whole number in the "Age" column, the value would be identified as invalid and the transaction would thereby fail.

Beyond enforcing the consistency of data's structure, integrity constraints may also be used to integrate conventions or business rules derived from the application domain. In this form, these integrity constraints are known as semantic constraints. To illustrate, consider the "Age" column from Table 1. As mentioned, this attribute represents the age of a person. Though the data type was defined as an integer in a previous example, an age can never be less than zero. Based on this logic, it can make sense to further refine the domain constraint on the "Age" column, asserting that the value must not only be of an integer data type, but must also always be greater than or equal to zero. In this way, values that don't make sense in the application domain will be omitted. Semantic constraints may also be used to control the progression of the data. That is, considering the current instance of a data element, restrict the next legal instances of the data element. For example, an application-specific assertion could be that an employee's salary, such as in Table 2, may not change more than 5% at a time.

2.3.5 Database Views

Where a query can be used to define a new relation from existing tables, a view is a named query that essentially defines a new virtual table. Views differ from relations in that views do not physically exist in the database. While relations are part of the physical schema, views are instead part of the external schema. Views are derived dynamically from the data stored in the database, and made accessible in a virtual table. Any changes to the underlying relations, such as transaction updates, are still reflected in the view. Views also still operate as a normal table, in that they allow queries and modifications, but provide the organizational benefit of only displaying/representing a subset of the actual data. Essentially, views provide abstraction

for a database, hiding the actual complexity of the data by simplifying multiple tables, partitioning data, or including extra data such as aggregation.

The concept of database views is of interest to the topic domain-specific modeling languages in the case of sub-setting the language. For instance, it may be necessary or desirable to restrict part of the language. Thus, the concept of database views may be extended to dynamically generate a domain-specific modeling language on a slightly smaller subset domain.

2.4 Search and Planning from Artificial Intelligence

Artificial intelligence provides a general approach for automating problem solving, using search and planning. These topics are all related to the automatic generation of workflow. As mentioned in section 2.1.1, a workflow can be thought of as a state diagram, thus the concept of search spaces may be extended to workflow. Furthermore, building a workflow is a matter of constraint satisfaction (business rules, pre/post conditions, pre/post requisites, etc). Finally, with the constraints specified, the concept of a planner can be used to search for a solution to the constraints and construct a workflow.

2.4.1 Search Spaces and Search Algorithms

Search in artificial intelligence requires two steps. First, a problem is represented as a search space. A search space is a term that refers to the set of all possible solutions to a problem. According to Coppin [14] "a search space is a representation of the set of possible choices in a given problem, one or more of which are the solution to the problem." Specifically, each choice in a given a problem can be represented as a state in a state-space diagram. Included in these states spaces would be an initial state where the problem starts, and a final state(s) which represents the goal(s). A solution to the problem would then be represented by one or more

paths through the diagram from the initial state to the goal state(s). Making a choice would be represented by a state transition in the state-space diagram, meaning that the result of an action that was taken changed the state from one state to another state. An action, however, is an abstraction of some underlying process in the problem. Generally, there is a set of actions that can be performed while in a given state. The process of planning is used to determine which of these actions should be performed to make an appropriate state transition, bringing the current state closer to the goal.

The second step is to employ a search algorithm to evaluate the search space for a solution. The search space representation is used because searching is computationally feasible. However, there are various methods that can be used, each of which is more effective in certain cases.

Search methodologies are categorized based on their control strategy (the order in which they expand nodes in a search space. One family of search methodologies is known as exhaustive search. Exhaustive searches are also known as blind or brute-force methods because they do not take the specific nature of the problem into account, and thus blindly make decisions, trying every possibility until a solution is found. Examples of commonly known exhaustive search methods are the depth-first, breadth-first, and best-cost first search algorithms. While incorporating no knowledge abstracts the solution so it may be applied to wider range of problems, exhaustive searches tend to have a much larger search space, and thus have low asymptotic runtime efficiency. Conversely, informed search methods employ heuristics to incorporate knowledge of the problem, reducing the size of the search space. A heuristic allows the search method to make assumptions about paths that have not yet been explored to determine

which action should be taken next. This allows informed search methods to reduce the amount of time searching, but does make the method applicable to a smaller range of problems.

For both blind and informed searches, the method can be further classified as data-driven or goal-driven. Data-driven searches start from an initial state and take actions to move forward in the search space until the goal is reached. This process is also known as forward chaining. Goal-driven searches, on the other hand, use backward chaining, where the search starts at the goal state and work backwards towards the initial state.

2.4.3 Constraint Satisfaction

Constraint satisfaction problems are a sphere of problems where the goal state must be reached satisfying some set of constraints. Specifically, if certain characteristics of the state can be represented as variables, then the constraints enumerate the possible values these variables may take. Therefore, a solution must assign a value to each variable that satisfies each constraint.

A typical method used to solve constraint satisfaction problems is a search. However, basic methods such as an exhaustive search to find a path through the search space would be too inefficient in this case, because the constraints can be evaluated in any order. Instead methods such as backtracking are employed. The backtracking method is a recursive algorithm that incrementally assigns values to the variables as long as the problem remains solvable, but backtracks by unassigning one or more variables if the problem becomes unsolvable based on the constraints. If all of the variables get assigned and no backtracking occurs, meaning that each constraint is still satisfied, then a valid solution has been reached.

2.4.4 STRIPS

One example of an AI planner is STRIPS [15], which used an automated planning approach that uses a means-end analysis strategy to determine the differences between the current state and the goal state to select the appropriate action to move closer to the goal state. This strategy was designed as a problem solver for robots, to autonomously plan the sequence of actions a robot should take for navigation or to rearrange objects. The actions that can be taken are represented by a set of operators, that each have preconditions and postconditions associated with them. A precondition is a constraint that must be satisfied before the action can be performed. A postcondition is the "effect" of performing the action; what becomes true after performing the action and what becomes no longer true. For more on preconditions, postconditions, and predicate logic, see [16].

A STRIPS instance starts off in some initial state that is described by a set of true conditions. The instance is also passed the set of conditions used to describe the world for this instance, the set of available operators, and a set of true and false conditions to describe the goal state. To develop a plan, the STRIPS program would then first compare the current world model to the conditions of the goal state. If these conditions matched then the problem would be solved. Otherwise the program would select an operator to lessen the differences between the current state and the goal state. Once an acceptable plan was developed, the robot could execute the plan and should satisfy the goal conditions. However, if an unexpected effect occurred during execution, a form of exterior interference or something that was not described as an effect in the operator conditions, then the plan may not succeed.

2.5 Automatic Generation of Workflows

Relatively little literature ties workflow as developed by the ETL community to artificial intelligence planning. The former community generates workflows manually. The latter automates plans that, in effect, are workflows. Some communities have considered how the two areas can fit together, which is central to this dissertation. In particular, the relational community allows operations involving data stored in relational databases to be expressed through a high-level, declarative language, SQL. Statements in SQL are automatically mapped to query plans (which are workflows). However, a query planner is not able to map SQL statements to custom operations, only to generic set operations. Query optimization also does not incorporate domain knowledge, relying upon the user specifying a query to be knowledgeable of the data model.

Sun [17] sought to automatically generate workflows as a solution to the "highly manual and cumbersome effort" required to produce a workflow in the existing process. The approach taken employed a domain ontology to incorporate user specified constraints into the workflow generation process. All possible workflows were built from the ontology and a questionnaire then allowed users to specify constraints that would filter results until only a single solution was remaining. Beyond the generation of workflows, there is little in common with the approach taken by this dissertation. The research had no methodology for capturing domain knowledge, building the ontology, or extending the ontology (the ontology was hard-coded). Determining a solution by generating all possible workflows and iteratively pruning the excess solution through a decision tree is also distinctly different from the DSML approach and a hindrance to extensibility. This would be infeasible to apply to domains with complex workflows.

Zhang et al [18], captured domain knowledge from experts in the form of patterns. The approach was designed to be domain independent by associating these patterns with the task,

scenario, and solution. However, the approach does not fully generate a workflow, but instead just makes suggestions, leaving domain experts in the process. It cannot produce unique workflow solutions, but merely regurgitate what it has already seen. Furthermore, since it only gathers patterns, there is no way to extract the knowledge from the system.

Ambite and Kapoor [19] represents domain knowledge using predicate logic then generates a workflow through a planner, using a best-first search to build a workflow solution to a given user request. While based on a domain ontology, there was no methodology for building or extending this ontology. Also, rather than having an intuitive interface for users to specify goals, the user specifications are complex and technical, preventing non-experts from using it. Finally, the workflows the planner was capable of building were only a simplification of workflow, having only a single way to connect and order operators.

Previous research by Xiao, Thompson, and Li [20, 21, 22] on our own project employed a planner, capable of generating all of the possible workflows to transform a given initial set of input fields to a specified set of output fields. Though this research included a knowledge representation, it was only able to capture a small portion of the domain knowledge. Constraints on workflow composition due to operator logic, workflow standards, and business rules could not be represented in an extensible manner. Finally, the interface was not high-level, but instead technical, expecting specifications in the form of fields rather than statements that express the user's intent.

3. APPROACH

3.1 High Level Design

The automation of ETL workflow specification should meet several design criteria in order to effectively supplant the current manual approach. These requirements include:

- Significantly reduce human time required to produce complex ETL workflows.
- Enable less technically adept people to define such workflows.
- Improve the correctness of ETL workflow specification.
- Facilitate the maintenance of ETL solutions over time.
- Be extensible to new ETL domains.
- Meet current and evolving requirements.

To meet these requirements, this dissertation proposes an extensible framework for the creation of domain-specific modeling languages that enables users to express the intent of a desired ETL solution at a high-level of abstraction and automatically generate workflows satisfying such specifications.

3.2 Scope

Before proceeding, it is important to note that the design is limited in scope by the assumptions outlined in this section. With respect to data, it is assumed that the data inputs are flat files of records (similar to relations in relation database systems) and that the fields of data resources have already been properly characterized. As the problem addressed by this work is focused on the construction of ETL workflows, concerns such as field identification and interrogation are considered out of scope. However, the automated determination of the characteristics of fields from an arbitrary data source is a problem that that has been automated

by a recent dissertation in the related area of layout inferencing [23]. The discovery, selection, and accessing of data resources is similarly considered out of scope. Data sources and sinks are assumed to be given, either by an earlier process or predetermined by the domain. Furthermore, while it is acknowledged that a workflow may involve multiple, heterogeneous data sources, for ease of presentation it is assumed that there is only a single data source, with homogenous data and known state. This limitation just means that the current ETL workflow automation system does not take into account whether data it is operating on is local or remote, in one or many repositories, or will suffer from more or less network latency.

With respect to operators, the discovery or generation of new operators is considered out of scope. Therefore, the collection of available operators is assumed to be given by the domain and each has well-defined functionality with a known set of inputs and outputs.

3.3 Representing ETL Domain Knowledge

As described in Chapter 2, ETL workflow specification is a process complicated by multiple interplaying factors. Not only are the valid configurations of ELT solutions dependant on a number of low level details, such as the collection of available operators, the underlying logic of these operators, and the characteristics of the data, but also they are similarly influenced by high level concerns, such as adhering to standards and business rules. For such reasons, to ensure that the workflow solutions produced have the correct structure and composition to yield the intended results, mapping a language to workflow specifications and automatically generating valid workflows necessitates that such knowledge be captured in a formal representation.

In developing such a representation, it must be considered that this influencing knowledge is neither uniform across domains nor guaranteed to remain static within a given

domain. Accordingly, a knowledge representation must be based on core concepts shared by ETL domains, modeled at a sufficient level of abstraction to facilitate extensibility. To this end, recall from Chapter 2 that ETL workflows are composed of two fundamental elements: fields and operators. This is true despite variations in the knowledge influencing the specification of a domain's ETL solutions. The direction of the approach therefore is to represent the domain knowledge that determines the composition and structure of an ETL workflow as characteristics of these elements.

3.3.1 Fields

A key assertion of this approach is that the data each field contains can be described at a higher level of abstraction. Currently, the definition of fields is done in a domain-generic manner. Each field is defined with two basic properties: a name and a data type. The name property (e.g. firstName or fieldX) uniquely identifies each field. The data type property (e.g. varchar or int) classifies allowable values of each field. However, for the purpose of ETL workflow specification, fields cannot be differentiated based on such domain-generic properties [24]. As the targets and products of the operations in a workflow, though the data of a field is being transformed as a result of these operations, the field's name and data type remain static. Neither of these properties can express any knowledge about a field at the level of abstraction of the given domain, such as a first name versus a last name or raw address data versus an address in a standardized format. Domain concepts such as these enable the significance of each field to be expressed clearly and thereby handled appropriately within the context of the given domain. Therefore, it is critical that such granular knowledge of ETL data be captured and represented explicitly. This is accomplished in the approach by modeling a domain's fields according to the characteristics known as content type and state attributes.

Content type refers to the classification of fields based on the high-level concepts a domain uses to categorize data. The content type of a field, of which a field may only have one at any instance, describes the type of data the field contains using the appropriate domain concept, which would be a noun such as “first name”, “phone number”, “zip code”, etc. Each content type abstracts a collection of data properties, as determined by the defining domain concept, allowing fields that contain related data to be associated based on content type. For example, the fields UserField0 and Contact in Figure 10 can both be characterized as sharing a content type of “Name”. Similarly, UserField1, UserField2, Address1, and ZipID can all be characterized as sharing a content type of “Address”. This is synonymous with the concept of classes from object-oriented analysis and design, where classes are types defined according to the problem domain, allowing numerous instances of a class, declared as objects, to share an explicit relationship as they share the collection of properties abstracted by the defining class [25].

UserField0	UserField1	UserField2
James Love	1 Main St	AR

Address1	ZipID	Contact
55 North St	37345	Kris Mir

Figure 10: Name and address data records.

State attributes, on the other hand, refer to distinguishable qualities that data may possess. While similar to content type, rather than classifying data into types, state attributes denote

known or asserted facts about the state of the data a field contains. Consider that the state of data may not be specified by an atomic quality, but instead be a composite of multiple qualities. Each of these qualities is an abstraction of data properties described by a domain concept, typically using an adjective or verb such as validated, filtered, standardized, corrected, etc. Accordingly, the state attributes of a field, of which there may be zero or more, represent the various aspects of the state of a field's data. To illustrate, refer again to the data fields depicted in Figure 10. The address data in field UserField1 could be known to be in a standard format and thereby can be characterized with the "Standardized" state attribute. After being run through a particular operator that geocodes addresses, this field could additionally be characterized with a "Geocoded" attribute.

Incorporating these abstractions, the resulting conceptual model of ETL fields can be described as shown in Figure 11, where - in addition to the existing name and data type - each field is characterized by a content type and collection of state attributes. It therefore follows that an implementation of this model must define the set of content types and state attributes. As these values are domain-specific, formalizations of concepts a domain uses to describe properties of their data, this step is left up to the domain modelers.

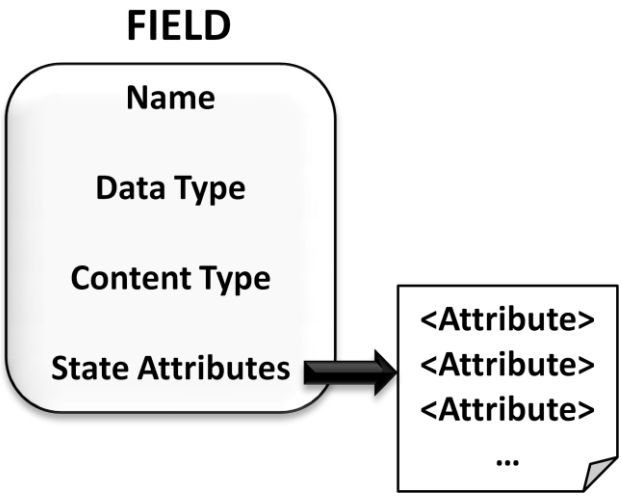

Figure 11: Attributed field model.

3.3.2 Operators

Operators are the elements of ETL that perform work. Specifically, the data processing operations that operators enact when executed are what produce and manipulate the data contained within fields. Yet, to ensure an operator produces a particular result, mapping the inputs of operators must be compliant with the properties of each operator. As operators are essentially programs that accept parameters and produce return values, such properties of operators constraining their usage and determining the results they produce can be expressed using concepts of programming languages: preconditions and postconditions.

3.3.2.1 Operator Preconditions

The preconditions of ETL operators are assertions that must be true prior to an operator's execution to guarantee that the operator will perform its defined data processing operations in an intended manner. Assuming operators are independent constructs, the only variables of an operator on which to base such assertions are those introduced by their collection of inputs. Operator preconditions then must represent each assertion as a requirement that a parameter

mapping must satisfy to be considered valid.[6] For instance, the operator depicted in Figure 12 could require that the input field "Input 1" is always mapped, while mapping the input field "Input 2" is optional and "Input 3" can only be mapped if both of the other input fields are mapped.[7] Such stipulations on how inputs may be utilized conjunctively are just one example to illustrate the variability of the constraints that a representation of operator preconditions must capture. Regardless of the origins of such conditions, the case where an argument is mapped only to "Input 1" must be recognized as valid, while the case where arguments are mapped to just "Input 1" and "Input 3" must be recognized as invalid.

Figure 12: ETL operator with inputs and outputs.

The representation of operator preconditions is handled using predicate expressions. As predicate expressions, assertions that must all be true to satisfy a particular precondition are composed as a logical conjunction. If there are multiple of these conjunctive groups of assertions, denoting more than one distinct precondition for an operator, they are combined as

[6] A parameter mapping, as it applies to an operator, refers to the mapping of fields in the workflow's data set to the input fields of the operator. As an option is also considered a parameter of an operator, a parameter mapping can also refer to specifying the settings of the operators options.

[7] To clarify the usage of the term "mapped", for an input field of an operator this refers to the input field parameter having an argument field from the workflow's data set specified (i.e. mapped).

logical disjunctions. An operator's precondition may then be evaluated based on the truth of these underlying predicates.

To illustrate, consider again the requirements on input mapping combinations presented in the previous example. The precondition of this operator can be expressed as depicted in Figure 13, where the presence of an input's name indicates the input requires an argument, while the absence indicates the use of the input is not allowed. Given an input mapping that supplies an argument only to "Input 1" or one with arguments for both "Input 1" and "Input 3", this expression of the operator's precondition would correctly evaluate to true and false respectively.

```
"Input 1"
|| ( "Input 1" && "Input 2" )
|| ( "Input 1" && "Input 2" && "Input 3")
```

Figure 13: Expression of an operator's preconditions.

Building on this basic representation, the precondition of an operator must additionally specify the properties a field must possess to be a valid argument for a particular input field. As these candidate arguments are in fact fields, which - from the field model presented earlier – are characterized on such properties by means of content type and state attributes, these abstractions may similarly be utilized to specify required or disallowed conditions on input fields. Mapping an operator's input fields can then be evaluated by comparing the content type and state attributes of the arguments against those defined in the precondition. For example, consider determining an argument mapping for the operator in Figure 14. In this simple example, the operator has just a single input, the input field named "Input 1". The preconditions specified for this input indicate assertions that an argument must contain either street address data that is in a standardized state or full name data and that is in a corrected state. Selecting arguments from the

fields available in Figure 10, the only content matches would be the fields Address1, UserField1, Contact and UserField0. However, the precondition on field state cannot be satisfied, as both street address fields lack a standardized attribute and both full name fields lack a corrected attribute. As a result, it can be easily determined from the given precondition that none of these fields are valid arguments.

"Input 1"{ **Content =** Street Address; **State=** Standardized; }
|| "Input 1"{ **Content =** Full Name; **State=** Corrected; }

Figure 14: ETL operator with preconditions.

The other type of input for ETL operators to be considered is options. Recall that the arguments for options are selected from a predefined set of values. Accordingly, the cases when each of these value settings is a valid argument must be specified as part of an operator's precondition. An additional consideration, however, is that each option setting may alter an operator's execution behavior. Such alterations may in turn affect an operator's precondition, introducing additional assertions on the operator's inputs, both input fields and other options. These assertions are therefore represented as preconditions on option settings. The evaluation of an operator's input mapping would then be the conjunction of the operator's precondition and the preconditions of each option based on its setting. Using the operator depicted in Figure 15 to illustrate, assume that the inputs "Option 1" and "Option 2" can both have a value Y or N. From the operator's precondition, it can be determined that "Option 1" and "Option 2" cannot both be set to Y at the same time. Furthermore, if "Option 2" is set to Y, an argument for "Input 1" must

be in a standardized state to be considered valid. It can thereby be ensured that not only are an operator's option settings valid, but also that the arguments mapped to an operator's input fields are in compliance with these settings.

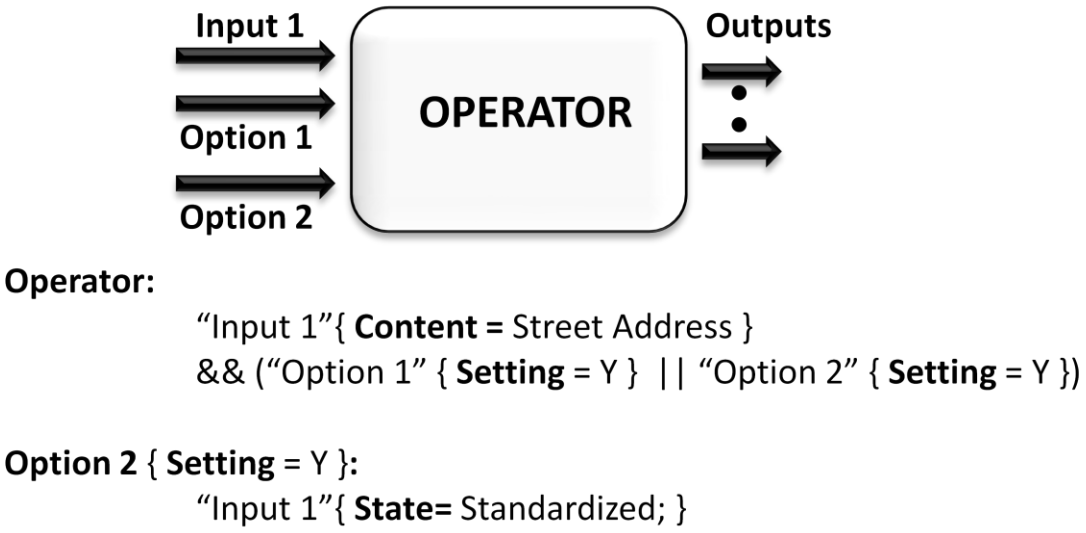

Figure 15: ETL operator with an option-level precondition.

3.3.2.2 Operator Postconditions

The postconditions of operators are assertions that will be true after an operator executes, provided that the preconditions are satisfied. The dependence upon the satisfaction of the preconditions logically follows from the fact that an ETL operator's execution is undefined should the preconditions be violated, and thereby the condition of the results produced cannot be guaranteed. As an understood consequence of the execution of an ETL operator is that in carrying out its encoded data processing operations in a valid manner, an operator produces a particular result, these results can be persisted in a workflow's dataset by generating and appending new fields - as defined by the collection of output fields - or alternatively can overwrite the data of existing fields that were mapped into the operator as arguments. Such knowledge is critical to the process of workflow specification, as the projected condition of the data at any point in an ETL workflow must be known to determine which operators should

follow or if the goals of the solution have been satisfied. The role of an operator's postcondition, therefore, is to specify the particular qualities the transformed data has gained, lost, or been produced with; in essence, to describe the "effect" of the operator's execution.

Complementary to the preconditions of operators, the representation of operator postconditions employs the enhanced representation of fields to specify the condition of each affected field following operator execution. Through the specification of the appropriate characteristics - an implementation specific consideration to be handled by a domain's modelers - the diverse transformations produced by ETL operators may not only be captured, but also described in terms that are compatible with that of preconditions. First is content type, which specifies what category of data is to be written to each field in a postcondition after operator execution, providing a domain concept to describe the data produced rather than an ambiguous data type. For example, assuming the effect of the operator depicted in Figure 16 can be described as parsing the input into a first name and last name, with an input mapping that satisfies the given precondition, the defined result is that "Output 1" will be populated with first name data and "Output 2" will be populated with last name data. This knowledge can therefore be encoded by means of the postcondition shown.

Precondition:

"Input 1"{ **Content =** Full Name; }

Postcondition:

"Output 1"{ **Content =** First Name; }
"Output 2"{ **Content =** Last Name; }

Figure 16: ETL operator with a postcondition.

State attributes, on the other hand, are tied to each field in a postcondition to express (via domain concepts) the state in which the data values of new fields will be produced or existing fields will be transitioned to following operator execution. The specification of state attributes denotes alterations (additions or deprecations) to make to an indicated field's collection of state attributes. For example, consider an operator that performs an in-place transformation (i.e. writes transformed data back to the fields used as arguments), such as the one shown in Figure 17. The effect of this operator - assuming an input mapping that satisfies the given precondition - can be described as standardizing city, state, and street address data. To represent this data transformation, after valid execution the "Standardized" attribute is appended to the collection of state attributes of each transformed field, as specified in the postcondition. The resultant state of an argument for this operator would therefore be a composite of any attributes existing prior to the operator's execution with this "Standardized" attribute.

Precondition:

"Input 1"{ **Content =** Street Address; }
&& "Input 2"{ **Content =** City; }
&& "Input 3"{ **Content =** State; }

Postcondition:

"Input 1"{ **Content =** Street Address; **State=**Standardized; }
"Input 2"{ **Content =** City; **State=**Standardized; }
"Input 3"{ **Content =** State; **State=**Standardized; }

Figure 17: ETL operator with a postcondition that include state attributes.

A final consideration with the postconditions of operators is that, similar to preconditions, there may be multiple distinct cases. Specifically, different input mappings – for both input fields and options – may illicit changes to operator behavior, thereby producing result sets with different characteristics. The operator depicted in Figure 18, for example, has two distinct preconditions for its single input field, one accepting full name data while the other accepts street address data, each with different requirements on data state. Assuming the operator's execution is defined for both cases, given an input mapping that satisfies one precondition, the postcondition of the operator may be different than if given an input mapping that satisfies the other precondition. It therefore follows that the representation of postconditions must facilitate such multiplicity. To handle this, each postcondition is simply tied directly to the specific precondition(s) that it is defined for, such as shown in Figure 18.[8] This granular precondition to

[8] In this form, the disjunction of operator preconditions is implied.

postcondition association thereby formally defines the particular mapping of input fields and option settings required to produce a desired result.

Precondition:
"Input 1"{ **Content =** Street Address; **State=** Standardized; }
Postcondition:
"Input 1"{ **Content =** Street Address; **State=**Verified; }

Precondition:
"Input 1"{ **Content =** Full Name; **State=** Corrected; }
Postcondition:
"Input 1"{ **Content =** Full Name; **State=**Filtered; }

Figure 18: ETL operator with multiple preconditions and postconditions.

3.4 A Domain-Specific Modeling Language for Generating ETL Workflows

The described model establishes a representation of ETL domain knowledge. This representation captures the knowledge needed to compose operators and fields into valid workflows. From this foundation what remains is designing the framework for a domain-specific modeling language capable of leveraging this knowledge to automate ETL workflow specification. With the direction being to allow users to specify the desired composition of an ETL workflow at a high level of abstraction, consider as a basis how a query statement in SQL abstracts the details of the relational algebra used to perform the retrieval of desired data items. Each constituent term of a statement maps to lower-level concepts, providing a partial to full

specification. This translated specification is then used by the query optimizer to produce the query's execution plan and return the desired data results. Similarly, combining an engine capable of automating the process of workflow construction with a declarative language that allows users to express the desired results (intents) that can be interpreted into technical requirements by the engine can allow the appropriate composition of workflows (in terms of operators and input mappings) to be inherent from specifications written in this language. To this end, two components have been designed: a workflow engine and an intent language.

3.4.1 Workflow Engine

As implied, the workflow engine must be capable of generating workflows absent human interaction. Therefore, applying concepts of AI planning to ETL workflow, this engine maps the process of workflow construction as a search space problem. Recall from Chapter 2 that the search space problem requires three basic pieces of information: an initial state, a goal state, and the collection of available state transitions. Mapping these concepts to ETL, the state that data processing workflows are concerned with altering is that of the data; taking the given source data and producing the specified result. From this, the initial state is represented by the state of the source data being extracted, while the goal state is represented by the specified target state of the data. State transitions then are manifested by the distinct data transformations of operators. The workflow engine may therefore generate ETL workflows autonomously by determining the sequence of state transitions (i.e. operators and input mappings) necessary to move from the initial state to the goal state.

The design details of this engine are based on concepts incorporated from the knowledge representation. Data state is represented through the state attributes of the fields. With the state of source data being assumed as known, the initial state of a workflow is well-defined. As data

passes through operators, each operator's postcondition is applied to the appropriate fields, allowing each field to maintain a heritage of its state as it flows through a workflow. Postconditions also enable the workflow engine to select only the operators, input fields, and option settings necessary to attain the workflow's goal specifications. Preconditions then are evaluated against the current state of the workflow's fields, ensuring input mappings are valid. Finally, using forward or backward chaining, the workflow engine may arrange the appropriate operator's into the proper sequence to attain the goal state.

While the concrete planning algorithm employed by the workflow engine is left up to the implementation, available strategies include:

- *Depth first* – The depth first strategy attempts to reach the goal state from the initial state by evaluating as deep as possible upon each available path before backtracking. When traversing a step for the first time, all of the next possible steps are determined before advancing or backtracking. If either there are no next steps available, the strategy backtracks to the most recent unexplored step. If the goal state has been reached, the algorithm can either exit to return only a single solution or backtrack to the most recent unexplored step in order to find all solutions. Otherwise, if there are next steps available, the strategy advances to the first unexplored next step from the current step.
- *Breadth first* – The breadth first strategy attempts to determine every possible solution for reaching the goal state from the initial state. When traversing a step for the first time, all of the next possible steps are determined. Rather than directly advancing to one of these steps, however, each of the next steps is considered in a first in, first out (FIFO) manner, giving the evaluation of any neighbors of the current

step precedence. If the goal state has been reached or there are no next steps, the breadth first strategy simply continues to evaluate in according to FIFO until there are no steps left to evaluate.

- *A** – This best-first strategy is similar to depth first, attempting to reach the goal state from the initial state by evaluating as deep as possible upon each available path before backtracking. The difference, however, is that the A* strategy does not simply expand the first unexplored next step. Instead the next step to advance to is determined based on cost so far and selecting the unexplored next step with the least cost. Note that the assignment of cost is implementation specific.

Before moving on to present the final component, it must be noted that it is possible for a particular goal state to be attained by multiple, distinct ETL workflows. Either due to ambiguity of the goal state or flexibility of operator constraints, the following cases may result:

- The same collection of operator's arranged in different sequences.
- Multiple operator's with redundant functionality that are simply substituted for one another.
- Alternative distinct input mappings for the same sequences of operators.

These resultant workflows may in fact be equivalent – a determination to be handled by the governing domain – and thereby returning any of these solutions would be satisfactory. However, the converse that one or more are not equivalent may also be possible. In such cases, a domain may need to re-evaluate the domain models or goal specifications in order to pare down the set of resultant workflows to only the proper solutions. This topic of workflow equivalence is described further in Chapter 6 but incorporating it in the workflow engine is left to future work.

3.4.2 Intent Language

The workflow engine described provides a planner capable of generating ETL workflows. With the initial state assumed known and operators (plus associated domain knowledge) defined through the knowledge representation, the remaining dependency is a specification of the goal state. These goal specifications are difficult to understand in their raw, computational form; a specific set of fields, each with a specific content type and a specific set of state attributes to represent a goal data state. This computational form is necessary for the planning logic of the workflow engine, but such a technical means of goal specification would still be cumbersome to expert users and would prove overly complex for casual users. Reducing the complexity of technical specifications through the development of higher level abstractions is not uncommon in computing. As cited by Schmidt [26], there have been several innovations (in particular higher level languages) over the past fifty years that capture "design intent", enabling users to express solutions using terms from their domain of interest rather than in computing oriented terms. The purpose of the final component of the approach, the intent language, is to serve as such an abstraction, providing an intuitive interface that even users with minimal expertise can understand and utilize to specify data processing goals.

The focus of this language is to allow users to specify (at a high level of abstraction) the intent of what is desired; hence the name "intent language". Conceptually, the terms that allow users to express their intent should be drawn from the subject matter domain, using the appropriate domain concepts to describe each (desired) data processing goal of the given domain. The value of drawing on domain concepts for the language's terms is that it provides users with familiar terminology and allows users to understand what their specifications are expressing. From this conceptual design, the interface of the intent language can take many forms. The

interface can be visual having a dashboard with buttons, check boxes, and/or provide drag drop functionality (similar to ETL tools) for user interaction. The interface can also be command-line oriented, accepting typed instructions similar to programming languages. The languages terms and interface, however, are left to be determined by the implementation.

In order to constructively generate the workflow that satisfies a specified goal, however, this language must be capable of being translated from high-level specifications down to technical specifications. The core design of this language therefore involves the definition of mappings between high-level and low-level specifications; i.e. intent specifications and goal state specifications. Each term in the language must be mapped to its corresponding specifications of goal state, where these specifications of goal state are based upon the implementation of the workflow engine. A statement in this language consisting of one or more terms could then be translated into all of the requirements needed by the workflow engine to generate the workflows that satisfy the goals this statement represents.

4. IMPLEMENTATION

4.1 Overview

An implementation of the workflow representation and planning algorithm presented in the previous chapter was constructed to provide a functional prototype. From an architectural perspective, the focus was to provide extensibility through the separation of concerns. Decoupling the ETL domain knowledge from the logic of the prototype enables the knowledge to be updated to reflect changes in a domain (e.g. new operators, content types, and state attributes) or to fit new domains without requiring changes to the prototype's logic. Use of abstraction within the core logic permits components such as the workflow engine to be enhanced to accommodate new/custom rules. To this end, the prototype's architecture is logically structured as multiple layers, as depicted in Figure 19. At a technical level, the source code of this prototype was developed in C#. The programming techniques used during development followed object-oriented design principles in order to keep the source code both legible and extensible [27]. As a result, the implementation is divided among a collection of modular objects that compose the prototype's object model, shown in Figure 20. The remainder of this chapter explains the details of this implementation.

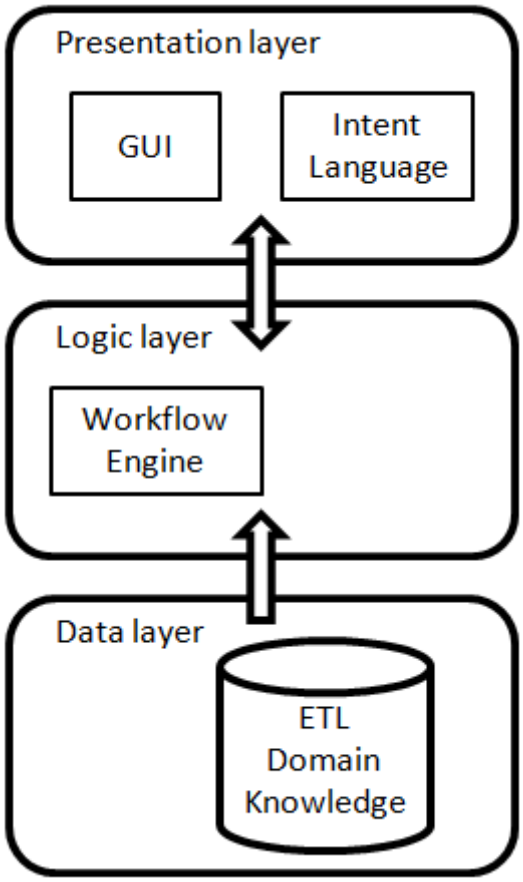

Figure 19: High-level architecture of the prototype.

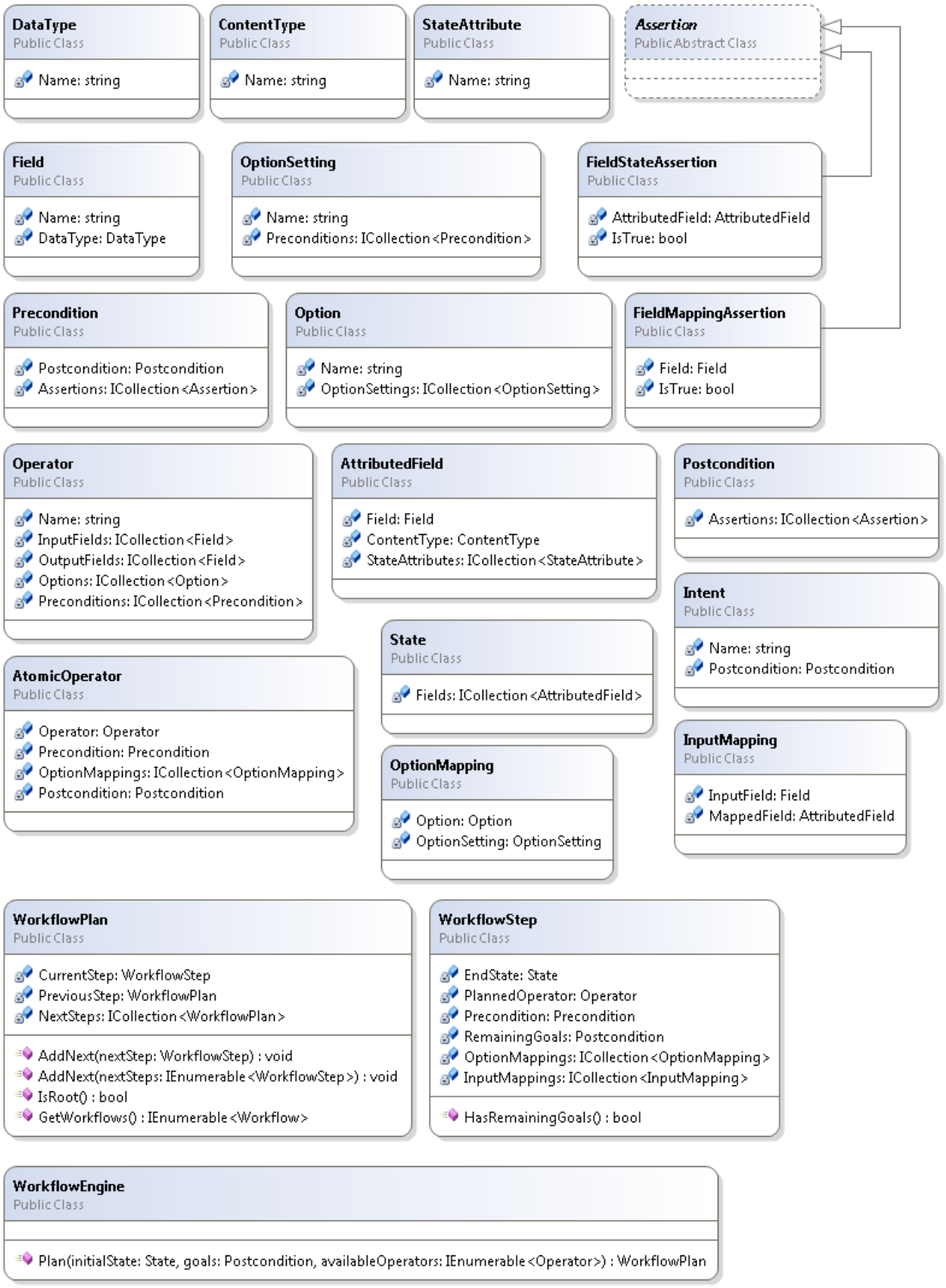

Figure 20: UML diagram of the prototype's object model.

4.2 Infrastructure

Before proceeding with the description of the prototype's implementation, it is necessary to first explain some fundamental aspects of the prototype's infrastructure, which encapsulates the persistence of the knowledge representation.

4.2.1 State

First is the concept of state and how its various applications were realized in the prototype. State, as it applies to both data fields and workflows, was described in Chapter 3. For data fields, state was represented via the state attributes on attributed fields. The state of a workflow was described as the state of all of the fields in the workflow at a particular instance, and was represented accordingly by a collection of attributed fields. As it applies to goals, however, state must be able to be specified in both an inclusive and an exclusive manner; that is, data state that should be produced and data state that should not be produced. For this reason, the state specified by goals was represented by postconditions. This is elaborated on more at the end of section 4.2.2.

4.2.2 Assertion Types

Next, one design requirement to implement the approach was the definition of concrete assertion types. In the approach chapter, both preconditions and postconditions were described as being composed of assertions. However, the base concept of an assertion is abstract; an assertion's definition changes depending on the criteria it is designed to check. While this abstraction allows the approach to be flexible, concrete definitions are necessary for an implementation. For this reason, two assertion types were designed: field mapping assertions and field state assertions.

The field mapping assertion was designed to represent constraints on mapping combinations of input fields; assertions which are found in preconditions. Recall from Chapter 2 that though an operator can have multiple input fields, only specific combinations of these fields being mapped in conjunction may be valid. As part of a precondition to express these valid combinations, a given field can be considered required, disallowed, or optional. When required, an argument field must be mapped to the input field for the constraint to be satisfied. Conversely, when disallowed, an argument field must not be mapped to the input field to satisfy the constraint. The final case, optional, indicates the absence of a constraint, where it is acceptable to either map or not map an argument field to the input field. For a representation of such constraints, the definition of the field mapping assertion consists of a reference to a field and a Boolean value (see Figure 20). The referenced field specifies which input field the assertion is for. The Boolean value modifies the evaluation of the assertion, representing whether the assertion should evaluate to true when the specified field is mapped or not mapped.

To illustrate the use of the field mapping assertion, consider the operator in Figure 21. Assume that a field mapping is required for “Input 1”, a field mapping is disallowed for “Input 2”, and “Input 3” is optional. These constraints would be expressed in a precondition by including a field mapping assertion for “Input 1” with the Boolean value set to true, a field mapping assertion for “Input 2” with the Boolean value set to false, and including no field mapping assertion for “Input 3”. This example precondition would then evaluate to true in two cases: when an argument field is mapped to “Input 1”, an argument field is not mapped to “Input 2”, and an argument field is mapped to “Input 3”; or when an argument field is mapped to “Input 1”, an argument field is not mapped to “Input 2”, and an argument field is not mapped to “Input 3”. Note that the optional case is represented implicitly through the absence of an assertion. An

alternative to represent this explicitly could split the single precondition into two preconditions, both with the same assertions for “Input 1” and “Input 2”, but one with a field mapping assertion for “Input 3” with the Boolean value set to true and the other with a field mapping assertion for “Input 3” with the Boolean value set to false.

Figure 21: An example operator with multiple inputs.

The field state assertion was designed to represent constraints involving the state of a field’s data. The definition of the field state assertion consists of a reference to an attributed field and a Boolean value (see Figure 20). This assertion type, however, was designed to be employed differently depending on whether the instance of the assertion is part of a precondition, postcondition, or intent.

For preconditions, a field state assertion is used to represent constraints on the state of a field’s data for mapping to an operator’s input field. As described in Chapter 2, the data mapped into an operator’s input fields can be required to exhibit certain characteristics; characteristics that, from the model presented in Chapter 3, are represented by content type and state attributes. Field state assertions are used as part of a precondition to express the content type required, the state attributes required, and the state attributes disallowed for a given input field. The referenced attributed field specifies which input field the assertion is for, which content type is required, and which state attributes are either required or disallowed. Similar to the field

mapping assertion, the Boolean value modifies the evaluation of the field state assertion. When the Boolean value is set to true, the assertion only evaluates to true when a field mapped to the input field of the assertion has both the same content type specified by the assertion and each of the state attributes (or a superset of the state attributes) specified by the assertion. When the Boolean value is set to false, the assertion only evaluates to true when a field mapped to the input field of the assertion has none of the state attributes specified by the assertion.

Using the operator in Figure 21 as an example, assume that "Input 1" requires the content type C1 and the state attribute A1, while "Input 2" requires the content type C2 and the state attribute A2 but disallows the state attribute A3. These constraints would be expressed in a precondition through three field state assertions: one using an attributed field for "Input 1" with the content type C1 and the state attribute A1, with the Boolean value set to true; another using an attributed field for "Input 2" with the content type C2 and the state attribute A2, with the Boolean value set to true; and a final one using an attributed field for "Input 2" with the content type C2 and the state attribute A3, with the Boolean value set to false. This example precondition would then evaluate to true when a field with the content type C1 and having at least the state attribute A1 is mapped to "Input 1", and a field with the content type C2, having at least the state attribute A2, but not having the state attribute A3 is mapped to "Input 2".

For postconditions, a field state assertion is used to represent the results produced by an operator's execution: the addition of a new field containing data with a specific state or changes to the state of an existing field's data. To express the addition of a new field as part of a postcondition, the referenced attributed field specifies all of the characteristics of this new field: the field's name, data type, content type, and state attributes. For this case, the Boolean value of the field state assertion is not used in the evaluation of the assertion. Next, to express changes to

an existing field as part of a postcondition, the referenced attributed field specifies which field the assertion is for, what the content type of this field is changed to, and what changes are made to the field's state attributes. For this case, the Boolean value of the field state assertion modifies the evaluation of the changes made to the field's state attributes. When the Boolean value is set to true, the specified state attributes are added to the existing field. This addition of state attributes is performed as a merge, where the existing field's resultant set of state attributes has only one of any given state attribute. When the Boolean value is set to false, the specified state attributes are removed from the existing field. Any state attributes in the assertion not found on the existing field are ignored.

To illustrate the designed use of the field state assertion as part of a postcondition, consider the operator in Figure 22. First, assume that this operator adds the new field "Output 1" with the content type C1 and the state attribute A1 and changes the state of "Input 1" to the content type C2 and adds the state attribute A2. These changes would be expressed in a postcondition through two field state assertions: one using an attributed field for "Output 1", with the content type C1 and the state attribute A1, with the Boolean value set to true; and another using an attributed field for "Input 1", with the content type C2 and the state attribute A2, with the Boolean value set to true. The evaluation of this postcondition would add the new field "Output 1" with the content type C1 and the state attribute A1, and, assuming an argument field mapped to "Input 1" with content type C2 and the state attribute A3, the field "Input 1" would have content type C2 and the state attributes A2 and A3. Next, assume that this operator changes the state of "Input 1" to the content type C1 and adds the state attribute A1, but removes the state attribute A2. These changes would be expressed in a postcondition through two field state assertions: one using an attributed field for "Input 1", with the content type C1 and the state

attribute A1, with the Boolean value set to true; and another using an attributed field for “Input 1”, with the content type C1 and the state attribute A2, with the Boolean value set to false. In the case of an argument field mapped to “Input 1” with content type C2 and the state attribute A2, the evaluation of this postcondition would change the field “Input 1” to have content type C1 and only the state attribute A1. In the case of an argument field mapped to “Input 1” with content type C2 and the state attribute A1, the evaluation of this postcondition would leave the field “Input 1” the same having the content type C1 and only the state attribute A1.

Figure 22: An example operator with a single input and output.

Finally, for intents, field state assertions represent conditions in the data of the workflow that are required to be present or not present after workflow execution; i.e. the technical requirements that must be met by the end state of a workflow to satisfy a given goal. As stated, these conditions can be required to be present (inclusive) or required not to be present (exclusive). However, it is not assumed that the specific names of each field in the end state of a workflow that satisfies a given goal are known. Since the names of fields are specific to the data source or operator on which they are defined, that would require that each data source and operator needed to produce a given goal were predetermined in the definition of each intent. Therefore, these conditions are not asserted on specific fields, but instead on specific content types. Content types are a characteristic that categorize fields based on domain concepts. This is

useful in the specification of goals because it generalizes the specification – for a category of fields rather than a specific field – and permit the specifications to be more expressive as they are based on domain concepts. As part of an intent, the referenced attributed field specifies the content type the assertion is for and which state attributes are either required to be present or not present. Once again, the Boolean value modifies the evaluation of the field state assertion. When the Boolean value is set to true, the specified state attributes must be present in the end state of a workflow. It is not required that the specified state attributes are all on a single field with the same content type as the assertion or that only the specified state attributes are present on fields with the same content type as the assertion, only that each of the specified state attributes is present on any of the fields with the same content type as the assertion. When the Boolean value is set to false, the specified state attributes must not be present in the end state of a workflow. None of the specified state attributes can be present on any field with the same content type as the assertion.

As an example of the field state assertion's designed use as part of an intent, consider a field state assertion for the content type C1 and state attribute A1 with the Boolean value set to true, a field state assertion for the content type C2 and state attributes A2 and A3 with the Boolean value set to true, and a field state assertion for the content type C1 and state attribute A4 with the Boolean value set to false. This intent's assertions would evaluate to true for any workflow state that has a field with the content type C2 and state attribute A2, a field (possibly the same field as the first) with the content type C2 and state attribute A3, a field with the content type C1 and the state attribute A1, and no field with the content type C1 and the state attribute A4.

4.2.3 Storage

Another design requirement for the prototype's infrastructure was a solution for storage of ETL domain knowledge. The storage mechanism employed by the prototype was a relational database, SQL Server Enterprise 2008 [28]. The choice of a relational database is discussed as part of future work in Chapter 6 to overcome limitations imposed by space complexity. Using this storage mechanism, the physical data model was implemented following a table-per-type approach. Each entity (i.e. type) defined by the conceptual model was implemented as a table in the database. This implementation also employed normalization to keep the representation general purpose. Accordingly, foreign keys and association tables were incorporated to capture the one-to-one/one-to-many/many-to-many relationships between the entities. The table definitions are depicted by the diagram in Figure 23.

To explain this table structure further, consider the AttributedFields table from Figure 23. As described by the approach, an attributed field is an enhanced representation of a field. Where the base definition of a field consists of just a name and data type, an attributed field introduces a content type and a set of state attributes. However, while the base definition of a field is expected to remain static (at least within a domain) the content type and state attributes of a field may vary, such as throughout a workflow. Accordingly, fields are stored in the Fields table, as their definitions are reusable, and the AttributedFields table provides an association of fields to content types and state attributes. Since each attributed field has only one associated field and content type, these are represented by foreign key relationships to the Fields and ContentTypes tables. But, since an attributed field can have any number of state attributes, this many-to-many relationship is represented using the association table AttributedFieldStateAttributeAssociation.

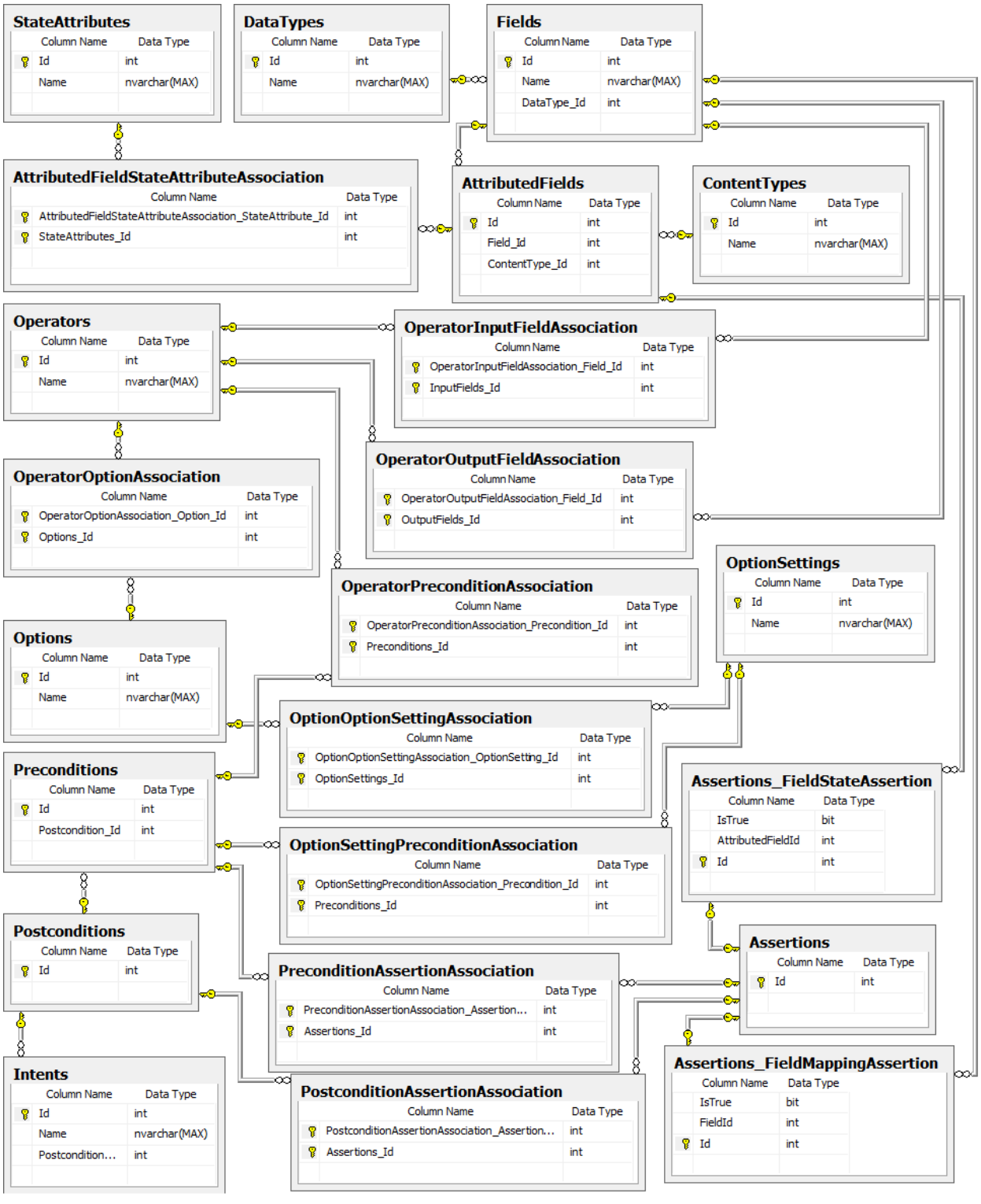

Figure 23: Physical table structure.

4.2.4 Data Access

With the ETL domain knowledge stored in relational tables, an additional requirement was to make this data accessible to the rest of the prototype. As the prototype's design was object-oriented, the objective was to encapsulate the raw relational data using structured objects; a computational form that was both extensible and organizationally convenient. Rather than developing custom SQL to query the ETL domain knowledge and logic to convert query results to objects, an object-relational mapper (ORM) was used. An ORM is a framework that abstracts the persistence of database entities, handling the retrieval and hydration of these entities into objects. Data access was therefore handled by the chosen ORM, Entity Framework [29].

Then, to address the architectural objectives, the repository pattern was employed to abstract these data access details from the other components of the prototype. The repository pattern is a design pattern that abstracts the persistence of data, exposing only a collection-like interface for access to the objects that encapsulate the desired data [30]. This extra layer of abstraction allows the rest of the prototype's implementation to remain unaware of the database integration and even the utilization of an ORM, further enhancing the flexibility of the prototype.

4.3 Prototype Execution

Building upon the details of the infrastructure is the implementation of the core components of the prototype; the intent language and the workflow engine.

4.3.1 GUI

The front end of the prototype was implemented as a GUI application, developed using Windows Presentation Foundation (WPF) [31]. As the sole entry point for the prototype, the GUI was designed to provide the following functionality:

- Specification of data processing goals (i.e. intents).
- Specification of the initial data state.
- Invocation of the workflow engine.
- Display of the generated workflow solutions.

Using this functionality, the GUI is a visual-based DSML, providing users with a graphical interface to compose statements in the intent language and submit them to the workflow engine to generate their desired workflow solutions.

4.3.1.1 Intent Specification

The interface for intent specification was provided as a dashboard in the GUI (refer to Figure 24). Intents added to this visual component are displayed as rows, where each row consists of two columns; one column is configured to display the intent's name and the other is a checkbox. The check state of each checkbox indicates whether or not the associated intent is being specified by the user; checked indicates include, not checked indicates not included.

Upon startup, the intents defined for the given domain are loaded from the database. As this control is scrollable, the number of intents the GUI displays is not constrained and so can be variable. Users then specify intent statements by scrolling through the list of intents and selecting the checkboxes by the intents that describe the data processing goals they want satisfied by the resultant workflow solutions. The resultant goal state specified by the user is then determined through the amalgamation of all of the checked intents. Each assertion from the postcondition of each intent is merged to produce a single specification of the goal state. Note that it is currently possible for a user to specify goals that are redundant – the goals of one intent are equivalent to or a subset of the goals of another – or conflicting – the goals of one intent

cannot be satisfied if the goals of another intent are satisfied. The identification and subsequent handling of these relationships between intents, however, is left to future work.

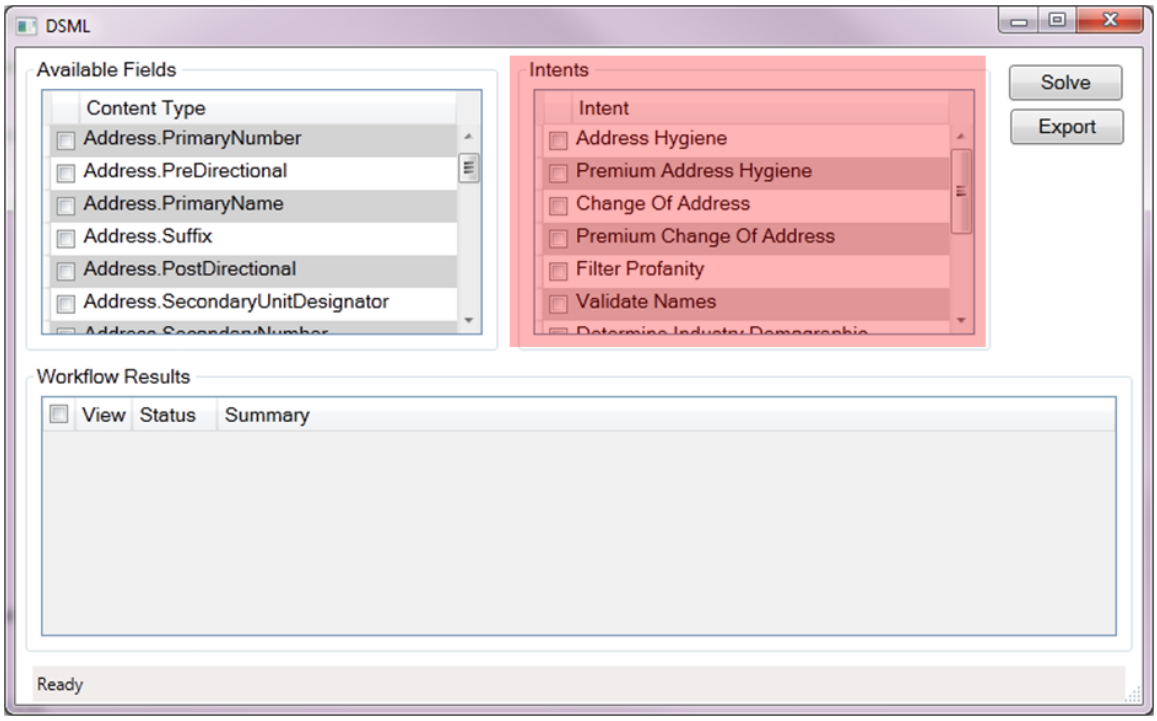

Figure 24: Intent specification in GUI.

4.3.1.2 Initial State

As specified in the approach, the determination of data state from an input file is out of the scope of this work, and it is instead assumed that the initial state of the source data is known. However, for the purposes of functional testing, it was necessary to provide a means of specifying an initial state for use in the invocation of the workflow engine.

To this end, an interface was provided in the same manner as intent specification; a scrollable dashboard that uses checkboxes for specification of the initial state (refer to Figure 25). Since what could be specified in the grid had to be representative of different possible data fields that were known to be valid for the given domain, all of the domain's content types and

state attributes were used. The content types were the items in the dashboard and a dialog allowed state attributes to be added to any row.

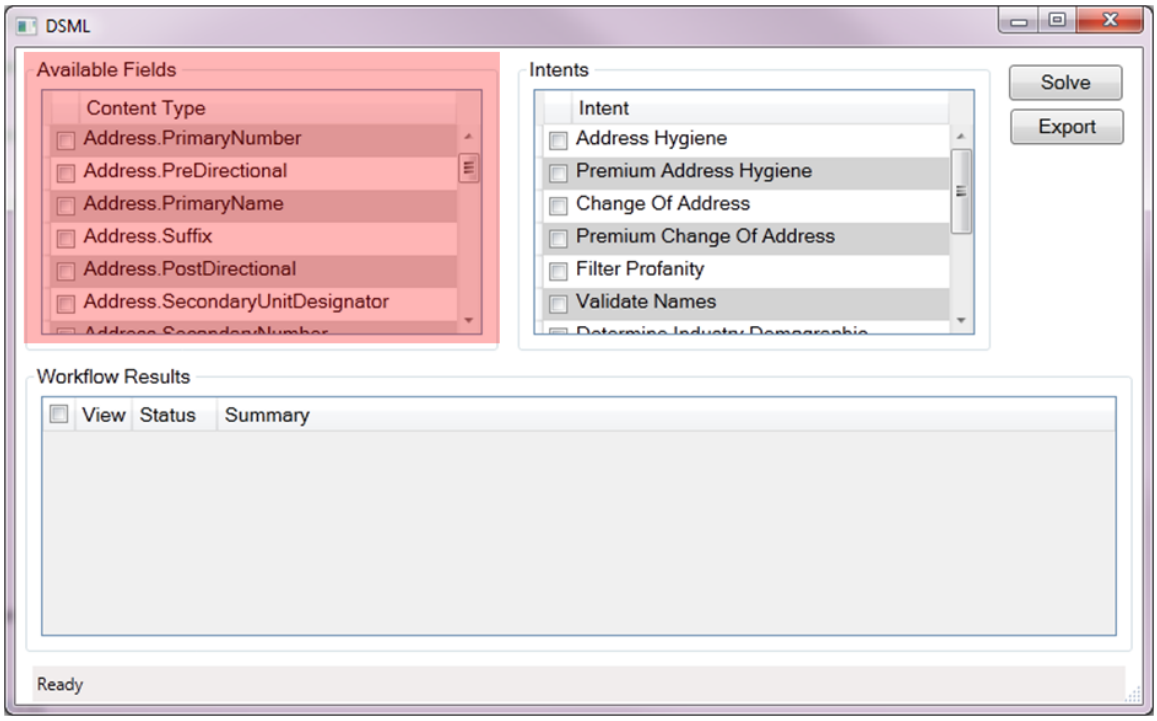

Figure 25: Initial state specification in GUI.

4.3.1.2 Workflow Engine Invocation

Invocation of the workflow engine is triggered via the "Solve" button. Upon clicking the "Solve" button, the user's specifications are validated to ensure both an initial state and a goal state has been specified. If this validation is passed, the domain's operators are loaded from the database and the specified initial state, goal state, and collection of available operators are passed to the workflow engine to begin the generation of ETL workflows.

4.3.2 Workflow Engine

Upon invocation, the workflow engine begins execution of the process flow shown in Figure 26. Adhering to the aforementioned design standards, the interface to the prototype's

workflow engine was implemented as a service. Specifically, the service itself is thin, devoid of the actual logic that implements each task in the workflow planning process, instead directing the appropriate business objects in the domain layer to carry out these tasks. The advantage to this implementation is that the workflow engine's logic is loosely coupled and so is highly flexible. That said, the implementation of the workflow engine's process flow follows a depth first strategy, working to find all workflows (as opposed to just a single workflow) to achieve the specified goal state from the specified initial state using the specified collection of operators.

As the workflow engine executes, a plan is generated to reach the goal state from the initial state. This plan, referred to as the workflow plan, is a collection of steps. Starting from the initial state, there can be multiple operator/mapping choices that transition the workflow closer to the goal state from the current step in the workflow plan. Following the depth first strategy described in Chapter 3, the planning algorithm evaluates as deep as possible upon each available path, where a path is a series of sequential steps in the workflow plan. The evaluation upon a given path ends when either the goal state has been reached or there are no more next steps available. At this point, the prototype's workflow planning algorithm backtracks, returning to the previous step in the workflow plan to evaluate other unexplored paths. Once all paths have been explored, execution of the workflow engine ends and the workflow plan contains all workflows that reach the goal state from the initial state. While this is a high level description, the remainder of this section presents further detail on the implementation of this planning algorithm.

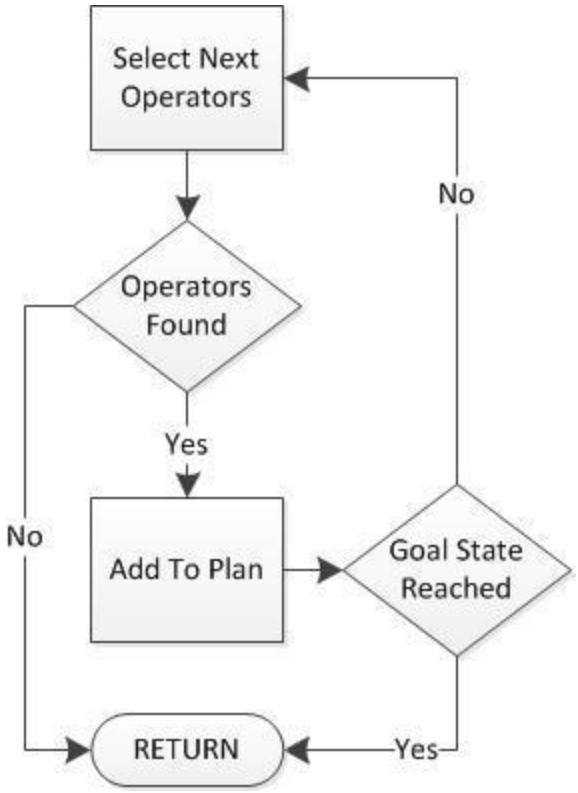

Figure 26: Process flow in the prototype's workflow engine.

4.3.2.1 Planning Tree

Tracking the workflow plan is an integral part of the prototype's workflow planning algorithm. When the current step in the plan reaches the goal state or has no unexplored next steps available, the workflow engine must be able to backtrack to a previous step in order to continue the evaluation for solutions. Also, once all paths in the workflow plan have been explored, the workflow engine must be able to extract each of the distinct paths that reach the goal state from the plan, as each of these paths represents a workflow solution. For these reasons, the workflow planning tree was implemented in the prototype to track the workflow plan.

As a tree-based data structure, the workflow planning tree is composed of a collection of connected nodes. Each node represents a step in the workflow plan, with the root being the

starting point of the workflow and the leaf nodes being the possible last steps based on the path taken through the tree. To allow the workflow to traverse up and down the tree, each node has a reference to a single parent (null for the root) and potentially multiple children. Furthermore, each node maintains details about the step in the workflow plan they represent. One such detail is the operator planned for that specific step in the workflow, as well as the option and input mappings planned for this operator to allow the exact workflow to be extracted from the tree. Additional details are the end state and remaining goals after the operator is executed, which are used by the workflow engine to plan the next possible steps in the workflow.

When execution of the workflow engine begins, the root of the workflow planning tree is initialized using the initial state as the end state and the goal state as the remaining goals. Starting with the root as the current node, the workflow engine determines which operators should be the next steps in the workflow plan and adds them as the children of the current node. The workflow engine then performs the same logic on the children of the current node in a recursive manner. If no next steps can be found for the current node or the current node has reached the goal state, recursion backtracks the workflow engine to the parent node and the next child is evaluated.

4.3.2.2 Goal Matching

As stated, the determination of which operators should be the next steps in the workflow plan selects only operators that bring the workflow plan closer to the goal state. Finding such operators is referred to as goal matching. During goal matching, all of the available operators are checked to find postconditions that produce any of the remaining goals of the current step in the planning tree. This is determined by comparing the assertions in each postcondition to each of the assertions in the remaining goals.

The logic that performs this comparison has multiple criteria to consider. One criterion is that the content types of the assertions (which are field state assertions) must match; if the content types differ, then the postcondition's assertion does not produce the specified goal. Another criterion is that no assertions can conflict. If the postcondition produces a state attribute that is disallowed by the specifications of the goal state, then it is not a goal match. The final criterion is also dependent upon the workflow's current state; i.e. the end state of previous node in the planning tree. If the content type already exists in the workflow's current state, then the postcondition's assertion must additionally share at least one state attribute with the goal state's assertion. This serves two purposes. First, by accepting a goal match if the content type does not exist in the workflow's current state, this allows the workflow engine to handle cases when a particular field does not exist at the start of the workflow, but must instead be produced by an operator during the workflow. Second, by disallowing a goal match if the content type does already exist in the workflow's current state but no state attributes are shared, this prevents false positives on operator's that are not actually brining the workflow closer to the goal state.

4.3.2.3 Input Matching

Even though an operator is a goal match, it should not be added to a workflow plan if no valid input mapping exists. To determine if an input mapping is valid, the input mapping has to satisfy the appropriate precondition of the operator. Here, a precondition is considered "appropriate" only if it produces the postcondition that was determined to be a goal match; however, this determination is not the responsibility of precondition validation, but is instead handled by a separate step that is discussed later. Moving on, finding a valid input mapping is performed using the fields from the end state of the current step in the planning tree. First, each of the field mapping assertions of the precondition is checked. If a field cannot be found for a

required field mapping assertion, a valid input mapping is not available. If a field is available for each of the required inputs, then the field state assertions are checked. If all of the conditions are satisfied, meaning the required content types match and there are no conflicts with the required and disallowed state attributes, then the field is considered a match. If a match is found for each of the required inputs, the operator is determined to have a valid input mapping.

4.3.2.4 Workflow Planning

Operators that are found to be both a match to the remaining goals and have a valid input mapping are considered candidates for the next step in the workflow plan. As described, each candidate is added to the planning tree as a child of the current node in the tree; growing the workflow plan. Once added to the workflow plan, these new children must have their end states and remaining goals evaluated in order to continue with the workflow planning logic.

First, the end state is determined by applying the postcondition of the planned operator to the fields from the input mapping. Applying a postcondition is performed per assertion. For a given assertion, the input mapping is checked for matching fields. If no matching fields are found, the asserted attributed field is added to the end state. Otherwise, the state attributes of the attributed field from the input mapping are merged with those from the assertion. The resultant end state consists of all of the fields from the previous state with the postcondition of the operator applied to the fields in the input mapping.

After the end state is known, the remaining goals can be determined. This involves taking the difference between the (now known) end state of the child node and the remaining goals of the current node. This logic takes the field state assertions from the postcondition that are true, finds the fields in the end state that match (based on content type) and takes the difference of the assertion with the matched fields. The result is a new assertion has all of the

state attributes that were in the old assertion except those that are present in the attributed field. If there are no state attributes left in the assertion, it is dropped from the result. Otherwise, it is added to the resultant postcondition. If no matching fields are found, the existing assertion is added to the resultant postcondition, as none of its asserted goals have been satisfied. What is left is a postcondition containing all of the assertions that have not yet been satisfied by the current state of the workflow.

4.3.2.5 End Conditions

Though the workflow engine attempts to plan the next steps of the workflow plan recursively, each recursive invocation checks to see if the goal state has been reached. This end condition is checked by evaluating the remaining goals of the current step in the workflow planning tree. One case indicative of the goal state being reached is when there are no remaining goals. The other case is when the only remaining goals are exclusive goals; having all inclusive goals satisfied and no exclusive goals violated indicates the goal state has been reached.

4.3.2.6 Operator Decomposition

A characteristic of ETL operators noted in previous chapters was that an operator can produce more than one distinct result depending on the inputs. That is, a single operator can have multiple preconditions, each of which can produce a different postcondition. To determine which operators will transition the workflow closer to the goal state, as is performed by the prototype, the workflow engine must know the distinct postconditions of each operator. Similarly, to know the input mapping required to produce a desired postcondition, the workflow engine must know the distinct preconditions of each operator. While these values can be determined on demand during the workflow planning process, the computational overhead associated with computing the distinct preconditions and postconditions of every operator each

iteration through the workflow planning logic becomes excessive. Therefore, an optimization introduced by the prototype that is performed before workflow planning begins is a process referred to as operator decomposition.

Operator decomposition is a process that breaks up a single, complex operator into many atomic operators; atomic referring to the construct having only a single precondition and postcondition. To decompose an ETL operator into atomic operators, the canonical form of all of the operator's preconditions, including the options, must first be determined. Canonical form refers to the standard form of presentation [32]. In the prototype, each precondition is a conjunctive expression and therefore a collection of preconditions tied to an operator or option setting is already in canonical form. What is required is to merge the operator's preconditions with the preconditions of the options. Note that during this merging, preconditions from two different option settings for the same option are not merged, because an option can only have a single setting mapped at a time. From the merged canonical form, the maxterms are the operator's discrete preconditions.

To demonstrate, consider an operator with the preconditions: (P1&&P2) || (P1&&P3). If the operator has only single option with two settings, one with no precondition and the other with the precondition P4, the merged canonical form would be: (P1&&P2) || (P1&&P3) || (P1&&P2&&P4) || (P1&&P3&&P4); four maxterms, meaning four discrete preconditions. If there were a second option, similarly with two settings, one with no precondition and the other with the precondition P5, the merged canonical would be: (P1&&P2) || (P1&&P3) || (P1&&P2&&P4) || (P1&&P3&&P4) || (P1&&P2&&P5) || (P1&&P3&&P5) || (P1&&P2&&P4&&P5) || (P1&&P3&&P4&&P5); eight maxterms, meaning eight discrete preconditions.

Of course, merging preconditions involves merging the underlying assertions. Assertions of the same type for the same field should be merged into a single assertion. For example, a precondition containing field mapping assertion for field F1 that is true being merged to a precondition also with a field mapping assertion for field F1 that is true should produce a single field mapping assertion for field F1 that is true in the resultant, merged precondition. Similarly, a precondition containing field state assertion for field F1 with state attribute A1 that is true being merged to a precondition also with a field state assertion for field F1 with state attribute A2 that is true should produce a single field state assertion for field F1 with both state attributes A1 and A2 that is true in the resultant, merged precondition. However, if two assertions are conflicting, such as asserting that a field both can and cannot be mapped or that a state attribute is both required and disallowed, it indicates that that both preconditions cannot be satisfied in conjunction. Such preconditions are excluded from the result.

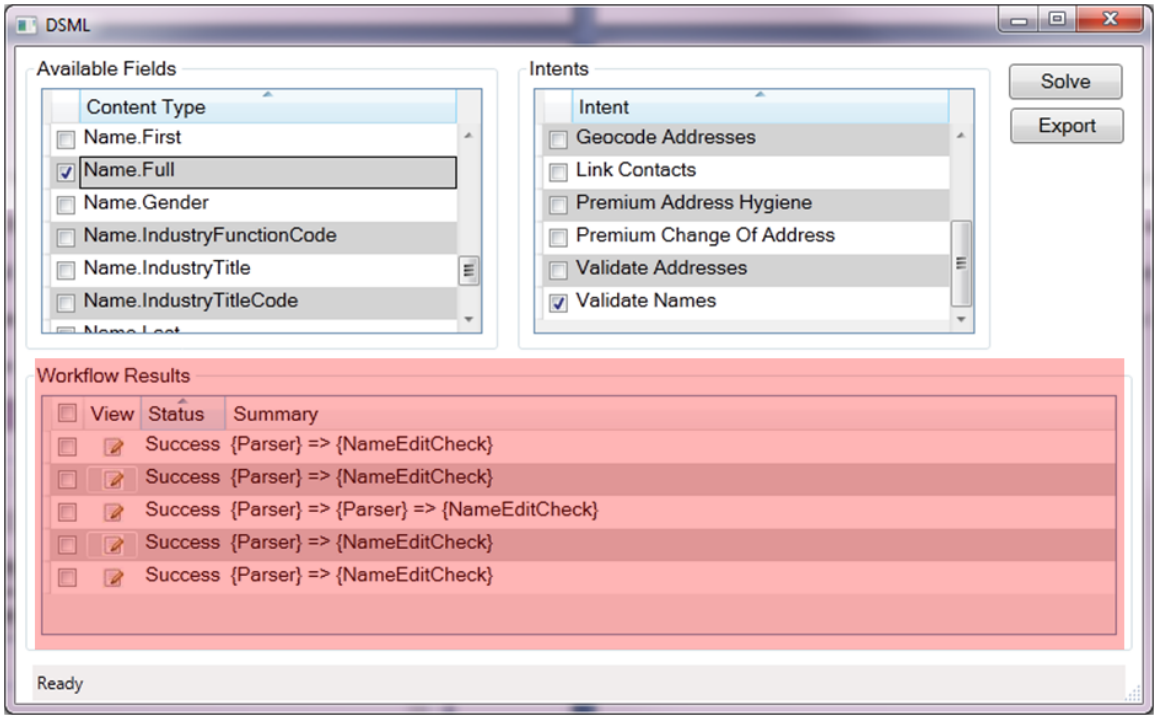

Figure 27: Workflow results in GUI.

4.3.3 Results

When execution of the workflow engine has finished, the workflow plan is returned to the GUI. The GUI then traverses the workflow planning tree to extract the solutions generated by the workflow engine, finding all paths that start at the root and end in a leaf node. Workflows that satisfied all the goals are given the status "Success", while those that failed to satisfy all the goals are given the status "Failure". A summary and the status of each workflow is displayed in the workflow results area (see Figure 27). Clicking the "View" button on a workflow displays a dialog window that presents the user with the details of the workflow (see Figure 28). The user may also click the "Export" button to save a specific workflow or all of the workflows to file. An example of an exported workflow can be viewed in Appendix C.1.

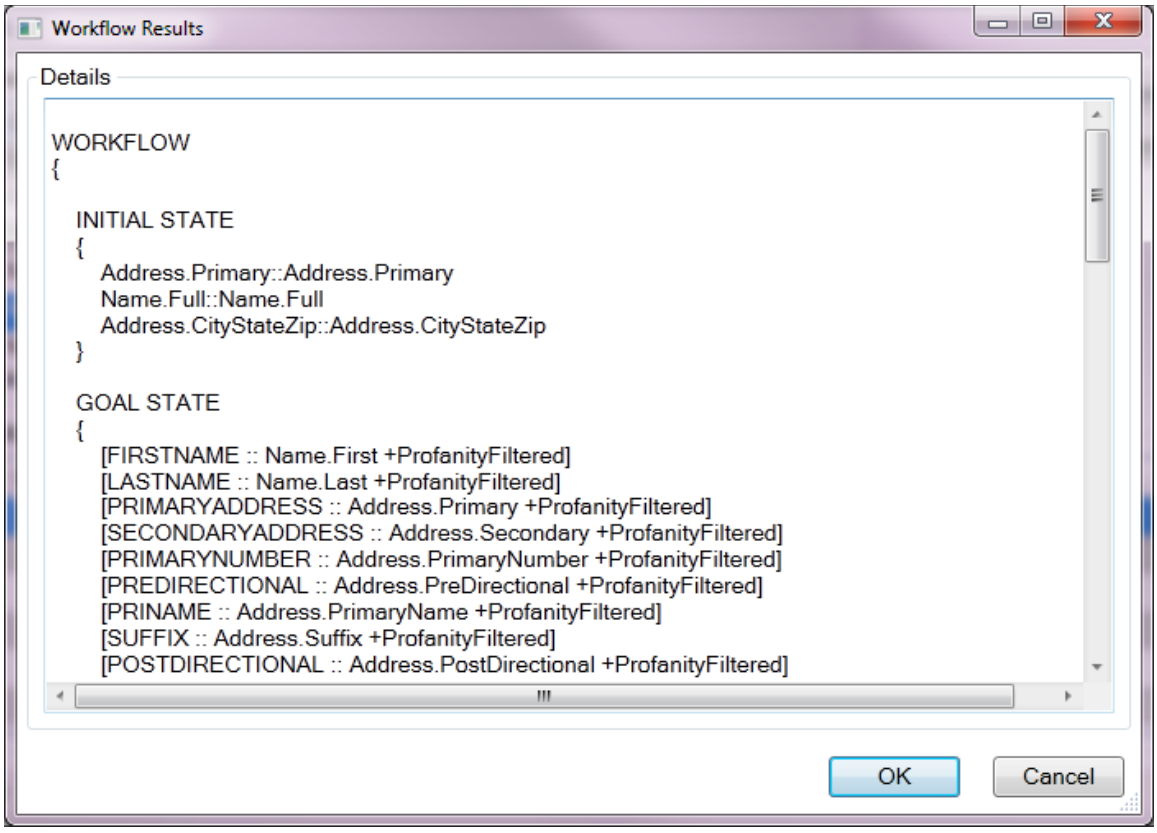

Figure 28: Workflow details in GUI.

5. VERIFICATION AND VALIDATION

5.1 Overview

Preliminary verification of the prototype was performed concurrently with the prototype's development through unit testing. The aim of unit testing is to isolate the smallest testable components of an application (i.e. units) in order to verify the behavior of these components individually [33]. While infeasible to define unit tests that cover all possible cases, unit testing helped ensure that functionality such as operator decomposition, goal matching, input mapping, determining remaining goals, and applying postconditions satisfied their design specifications (described in Chapter 4) before beginning system testing.

Once the components of the prototype were verified, testing focused primarily on validation: confirming that the software fulfilled its intended use [34]. System testing of the prototype was scenario-based, modeling a target domain and defining test scenarios. These test scenarios were then executed and their output was analyzed to demonstrate the validity of the prototype. The remainder of this chapter describes the target domain, details of modeling this domain, test scenarios selected, results obtained from these scenarios, and analysis of these results.

5.2 Target Domain

In order to have realistic scenarios to draw on for validation of the approach, an industrial partner was selected as the target domain. The services of this domain focus on improving the quality of their clients' data. Their clients are typically businesses that collect and store large amounts of customer relationship management (CRM) data, such as names and addresses of individuals and/or businesses. For various reasons, this data is often incomplete (names without

a mailing address), redundant (multiple entries for the same individual or business), or incorrect (the wrong mailing address associated with a customer's name). Given such deficiencies in data quality, the data processing goals of clients are typically related to business productivity. For example, a client that mails promotions to their customers may seek to correct erroneous addresses, update addresses for customers who have moved, or merely filter the customer data to remove redundancy.

Over the years, this domain has developed a large library of custom operators targeted at servicing their clients' goals. Many of these operators have a sizeable number of input fields, options, and output fields, and each is capable of producing a variety of transformations depending on the mapping of inputs. Due to the complexity of both the operators and targeted goals, the workflows that are produced to transform a client's raw customer data into the desired form can involve up to hundreds of these operators and thousands of data fields (e.g. source fields, input fields, output fields). Furthermore, construction of these workflows is performed manually and is constrained by a number of workflow standards and business rules that are only fully known and understood by domain experts. Consequently, these expert users must apply their knowledge of the operators and constraints to determine which operators to use, how their options should be set, and what sequence they should be composed in to attain the client's desired goals. Then an appropriate field must be selected from all of the available fields in the workflow and mapped to each input field of each of the operators. Finally, the workflows must be tested and refactored a number of times until it is verified that all of the operator mappings are valid and the workflow produces the result.

To populate the prototype's model for testing, an in depth study was performed on the target domain's workflow system. This study involved learning operator definitions and

business cases, then leveraging the domain experts to capture the various facets of associated domain knowledge. From this study, select operators and intents were modeled specifically for validation testing. In order to better understand the results obtained from testing, the following subsections describe these operators and intents that were modeled.

5.3 Operators

5.3.1 Catalog of Operators

From the target domain, eight operators were selected for use in testing. Note that the actual operators are more complex than those presented in this dissertation. However, details were left out so that this work can remain valuable without being overwhelming. Brief descriptions of these operators are as follows:

- *AddressEditCheck* – This operator interrogates address fields to determine if the contents are blank or invalid. Accordingly, the accepted inputs are name and address fields. By default, no inputs are required; instead, input requirements are dependent on the options.
- *AddressEnhance* – This complex operator provides a range of functionality that includes: address standardization and hygiene, delivery point validation, delivery sequencing, appending geocoding data, and updated data based on changes of address. Much of this functionality is enabled or disabled through options. Parsed address fields are required as inputs, where the setting of each option may further constrain the inputs.
- *AddressSelect* – This select operator has a well-defined criterion for selection: selecting the field with the most current address data from the inputs. Though the inputs are not specified by default, workflow standards provide an adequate definition for modeling.

- *ContactLink* – This operator cross references a knowledge base to find contact data that is determined to be related to a given record. Though not required by the operator, domain knowledge dictates that the address input fields have passed through AddressSelect first.
- *IndustryCode* – This operator determines an industry-based demographic code for an individual. To perform this determination, the operator requires only the professional title as input. A business name can also be included as an optional input to enhance the accuracy of the operator. The determined codes are appended to the data record as part of the operator's output.
- *NameEditCheck* – Using encoded rules, this operator determines if name fields contain data errors or invalid data. Either parsed or unparsed name data (but not both) is accepted as inputs. The operation is performed in place on the input fields, so no new fields are created as outputs.
- *Parser* – The primary function of this operator is to parse name and address data, accepting combinations of parsed and unparsed name and address fields as inputs. After execution, the operator outputs a new set of parsed name and address fields. A secondary function, which is controlled by an option, is filtering out any profane words from the output fields.
- *PremiumAddress* – This is an example of an operator with overlapping functionality of another operator, AddressEnhance. PremiumAddress performs address hygiene, as well as finding current addresses based on change of address. Transformations are only performed based on the options, but require address data that has been through ContactLink first.

5.3.2 Example Operator Model

This section describes the detailed specifications of the AddressEnhance operator, which are shown in Appendix A.2. The specification starts with the basic information about the operator. The names of each of the input fields and output fields were taken directly from the target domain's operator definition. All fields were defaulted to the "string" data type, but since data types were inconsequential in the prototype, these details were left out of the field specifications.

The next part of the specification is the preconditions. There were two preconditions defined at the operator level. Both involve only basic, parsed address fields. In both, the PrimaryAddressLine field is required, specified by the field mapping assertion. The expected content of this field was determined to be primary address data, accordingly specified by the content type Address.Primary in the field state assertion. In addition to the PrimaryAddressLine, one precondition requires the City and State inputs fields, the former expecting the field content to be city data, defined as Address.City, and the latter expecting state data, defined as Address.State. The other operator level precondition requires only the Zip input field in addition to the PrimaryAddressLine. As specified in the field state assertion, the expected content of the Zip input field is zip code data, represented by the content type Address.Zip. Note that neither of the preconditions requires any state attributes.

As an aside, note the naming convention of content types. The name of a content type is a hierarchy of categories of data content. A content type starts with a general category, such as "Name" or "Address", then is followed by a more specific subcategory of this general category, such as "First" or "Primary". The reason for using this convention is to enhance readability.

Following each precondition is the specification of the associated postcondition. Both preconditions share the same postcondition, consisting of two collections of parsed address fields. These are fields added to each data record as the result of the operator's execution. Each of these fields has been standardized and cleaned according to USPS guidelines, represented by the UspsStandardized state attribute. Each, except for the address flags, has also had the address checked as a valid point of delivery, represented by the DeliverabilityValidated state attribute.

Next in the specification are the options. The first option, Perform Delivery Sequencing, has two settings: Y or N. When set to Y, the operator will enhance the address data with delivery sequences; this is specified by the DeliverySequenced state attribute. The next option, Perform Change of Address, similarly has Y and N settings. Setting this option to Y enables the operator to find updated addresses based on change of address data. These updated address fields are populated only in the output fields specified in the postcondition. Additionally, this is represented by the AddressChange state attribute. Finally, the PerformGeocoding option enables geocoding of the address data, specified in the postcondition through the Geocoded state attribute.

5.4 Intents

While operators were known constructs, determining which intents should be modeled was more involved. The methodology devised for the elicitation of intents started by using the base content types defined for the domain from the operators; "name" and "address". Next, high-level descriptions of the ways in which a field's data can be modified or enhanced by the available operators were determined (e.g. "cleaned", "standardized", and "corrected"). Finally, the intents were mapped back to the technical requirements (operators and options) that produce the given transformations.

5.4.1 Catalog of Intents

Using the methodology described above, the data processing goals selected for testing were drawn only from those achievable using the chosen operators. As they are defined, intents can map to 1 or more operators. The descriptions of the intents are as follows:

- *Address hygiene* – Standardizes and corrects address data. This can entail applying substitutions, such as standardizing "123 Main Street" with "123 Main St.", or fixing errors, such as correcting "12 3 Main" with "123 Main St.".
- *Premium address hygiene* – Standardizes and advanced correction of address data. Premium address hygiene can handle more extreme errors than the non-premium version (and is therefore more costly to use), such as correcting "Main S" to "123 Main Street".
- *Change of address* – Use change of address data to find current addresses. This data is pulled from USPS change of address databases. For example, by referencing the USPS database, it can be determined that "John Doe" who lived at "123 Main St." now lives at "1 Walnut Ave.".
- *Premium change of address* – Uses proprietary change of address data to find current addresses, finding matches for older address data that the USPS database can't match.
- *Filter profanity* – Expunges profane words from name and address fields.
- *Validate names* – Flags name fields that contain data that is not a valid name. For example, a name field that is blank or contains "12345" would be flagged so that the data is not used to mail a contact.
- *Determine industry demographic* – Appends a code indicating the industry-based demographic of an individual. This could indicate the contact is a director of marketing,

executive vice president of sales, or another indicator useful in targeting the mailing campaign.

- *Validate addresses* – Checks the components of address data for common errors. Flagging addresses that are blank or that have invalid postal codes (e.g. less than 5 numbers or containing letters) avoids improper addressing of mail.
- *Delivery sequencing* – Appends sequence codes for mailing routes. Sequencing addresses for mailing routes saves money for mailing campaigns.
- *Geocode addresses* – Adds latitude and longitude coordinates for addresses.
- *Link contact* – Associates contact data to related records in the knowledge base. Though this, it can be discovered that "Jim Doe" who lived at "123 Main St." is actually the same person as "John Doe" that now lives at "1 Walnut Ave.".

5.4.2 Example Intent Model

This section describes the process of defining the intent "Change of Address". First, the purpose of this new intent was determined. The purpose was described as using change of address data to find current addresses. This description was reduced in order to be succinct while still capturing the domain concept, resulting in the intent's name "Change of Address". Next, the technical specifications required to accomplish this intent in the target domain's workflow system were gathered. From interviewing the domain experts it was determined that change of address data is used to find current addresses when the "Perform Change of Address" option is set to "Y" on the AddressEnhance operator. The postcondition of this option setting was analyzed, finding that the AddressChange state attribute indicates that a given attributed

field has had change of address logic applied to it. The AddressChange state attribute was therefore needed as part of mapping the intent.

An additional requirement was introduced from workflow standards. Whenever the change of address logic is run in the AddressEnhance operator, it is standard practice in the target domain to follow up with the AddressSelect operator. This standard is to ensure the most current address (between the before and after change of address logic) is captured. Capturing the most current address is indicated by the CurrentAddressSelected state attribute, which is appended by the postcondition of the AddressSelect operator. The CurrentAddressSelected state attribute was therefore also needed as part of mapping the intent.

Finally, the content types of the fields each of these state attributes should be present on at the end of the workflow were determined. These content types were a subset of the address content types and included "Address.Primary", "Address.City", and "Address.Zip". These requirements were combined to produce the definition of the "Change of Address" intent (shown in Appendix B.3). Translating this intent to workflow requirements specifies that the address data should pass through AddressEnhance with the Perform Change of Address option enabled and through AddressSelect.

Before moving on, note that "Change of Address" is distinct from "Premium Change of Address" in that they are not considered interchangeable by the target domain. In fact, it is common to have a single workflow constructed to satisfy both intents.

5.5 Test Scenarios

Actual business cases from the target domain were analyzed to choose scenarios appropriate for testing. As part of this analysis, the business cases considered were limited to those that fit within the scope of this paper, involving no generic operators and no repetition of

the same operator. That said, the business cases selected originate from a client planning to perform a mailing campaign. To improve the productivity of this campaign, the client seeks to enhance their customer's contact data. Such improvements include:

- Hygiene on address data for accurate mailing addresses.
- Enhancing the contact data with delivery sequencing to receive mailing discounts.
- Classification of customers through industry demographics, allowing the campaign to be targeted accurately.
- Checking for changes of address to update data that is out of date.
- Cross-referencing a knowledge base to identify duplicate contacts.
- Identifying and flagging records with invalid name and/or address data.

5.5.1 Test Scenario 1

The first scenario is a basic mailing enhancement package. The initial state consisted of unparsed data; full name, unparsed primary address, and unparsed city state zip. The selected intents were "Determine industry demographic", "Address hygiene", and "Validate names". A workflow generated for this scenario is shown in Appendix C.1.2. The expected workflow, which was produced by the target domain, is the following sequence:

- Parser => IndustryCode => AddressEnhance => AddressSelect => NameEditCheck

5.5.2 Test Scenario 2

The second scenario is an advanced mailing enhancement package. As before, the initial state consisted of unparsed data; full name, unparsed primary address, and unparsed city state zip. The selected intents were "Determine industry demographic", "Address hygiene", "Link contacts", "Validate addresses", and "Validate names". Due to the length of the results for this

scenario, these results were not included in the dissertation. However, the expected workflow was once again produced by the target domain and is the following sequence.

- Parser => IndustryCode => AddressEnhance => AddressSelect => ContactLink => NameEditCheck => AddressEditCheck

5.6 Analysis

In both scenarios, the solutions generated were 100% accurate for the specified goals. All of the workflows generated had a composition of operators matching the expected result and all of the workflows generated also successfully achieved the specified intents. The number of workflows generated was 384 for the first scenario and 7680 for the second scenario. Initially, this appears to be an overwhelming number of solutions. To better analyze these solutions, however, an enhancement was made to the GUI to consolidate the results based on the end state of the last step in each workflow for a given scenario. Workflows with the same set of attributed fields (same fields with the same content types and the same state attributes) were consolidated. After consolidation, only 16 distinct workflows were left for each scenario. The difference between these 16 workflows (in both scenarios) were the following options; "Perform Change of Address", "Perform Delivery Sequencing" and "Perform Geocoding" on the AddressEnhance operator and "Filter Profanity" on the Parser operator. Compared to the expected result, however, all of these workflows were valid. In the expected workflows, these options are set to a default, which can be changed if the client desires. While the user did not specify these goals, they also did not exclude them either. Since the intents were under-constrained, this lead to the overflow of workflows.

For another comparison, the distinct operator sequences were also determined. There were 12 distinct operator sequences for the first scenarios, shown in Appendix C.1.1, and 120

distinct operator sequences for the second scenario, shown in Appendix C.2.1. All contained the same operators (for the given scenario) and each sequence was valid as they all produce the same output state. The initial step of each workflow was to parse the unparsed data. This is due to the preconditions of all of the later operators requiring parsed name and address data. After this step, the sequence of operators differed only by the placement of a couple operators. Under closer scrutiny, this is because the operators' preconditions were satisfied early in the workflow and the goal state was not affected by the placement of the operator; therefore, there were multiple, equivalent placements. For example, the position of NameEditCheck was inconsequential as, after parsing, no operators would affect the name data.

For ease of use, ignoring the specification of the initial state (since it is actually assumed known), three intents were all that was needed to generate all 384 workflows and 5 to generate 7680 workflows. Each workflow for the first scenario involved 5 operators, 8 options, 84 input fields that had to be mapped, and 90 output fields. Each workflow for the first scenario involved 7 operators, 9 options, 116 input fields that had to be mapped, and 91 output fields. As a user, it was not required to have to select each of these operators, sequence them (note that order was important on some operators), or handle any field or option mappings. Due to the consolidation enhancement to the GUI (mentioned at the start of this section), the number of workflows for a user to consider was reduced. However, it was also determined that ease of use could be improved further by providing users with a means of distinguishing between the workflows generated. An enhancement was made to the GUI to rank the workflows based on distance from the user's specified goal state. All of the workflows satisfied the goals, but some performed more transformations than others, resulting in the differences in the end states of the workflows. The distance enhancement quantified these differences by calculating the number of distinct

content types and state attributes that appear in a given workflow's end state but are not in the user's specified goal state. The workflow with the minimum calculated distance is the solution that most tightly bounds the user's goal specifications.

Similar to ease of use, the prototype demonstrated reduced time to produce workflows. Solutions were generated in less than one minute for both scenarios. Also, having to select only three or five intents (depending on the scenario) and click "Solve" was able to produce all of the solutions. As opposed to the manual process, which (for 384 workflows) would require 32,256 input fields and 3,072 options to map by hand (either in an ETL tool or in XML); for 7680 workflows, this would require 890,880 input fields and 69,120 option to map by hand (a large increase).

The workflow engine's design prevents it from constructing workflows that violate known constraints. There can always be cases when a constraint was neglected (as part of modeling) but once identified and defined, such errors will be avoided. In contrast to the manual process, there is no chance of errors (once again this assumes proper modeling). To validate this, the workflows generated were inspected to find the presence of any known composition errors, but none were found (as expected from the unit testing verification).

The prototype was able to capture all of the necessary domain knowledge. It was flexible enough to handle the known constraints of the target domain. The architecture of the prototype is the best evidence of the extensibility. However, this could be reinforced by future work to apply this approach to a different domain.

6. CONCLUSIONS

6.1 Summary

ETL workflow specification is the process of composing operators and fields into a workflow that satisfies a particular set of requirements. When these requirements are specified at a high-level of abstraction, domain knowledge becomes critical to perform this process as such knowledge drives the translation of high-level specifications into lower-level technical requirements needed to build the workflow. Field and option mappings, operators to include or exclude, and operator sequence are all characteristics of an ETL workflow that affect the result a workflow produces and so must be determined to build a workflow that satisfies the given requirements.

This dissertation has presented a novel approach (compared to the existing solutions surveyed in Chapter 2) for automating ETL workflow specification consisting of a generic model to abstract and capture ETL domain knowledge, a declarative language to allow users to express requirements at a higher level of abstraction, and a planner to automatically generate workflows that produce a specified goal state. Based on the framework defined by this approach, a functional prototype was implemented to demonstrate that the proposed approach conforms to the objectives of the thesis.

Testing the prototype on a real-world domain provided a demonstration that both the human time and expertise needed to produce an ETL solution were reduced through use of the DSML, having only to click a few buttons to generate the solutions for a given test scenario. The results obtained from testing were then analyzed to demonstrate that, according to the specified intents, the solutions produced were both accurate and error free.

Finally, extensibility was exhibited by the architecture of the prototype, in the decoupling of ETL domain knowledge from the workflow generation and intent translation logic, and the implementation of the physical data model, kept normalized to allow the knowledge representation to allow for enhancements.

In summary, this dissertation has demonstrated its objective that:

ETL workflow specification can be automated in an extensible manner by translating high-level statements of intent into a set of ETL workflow requirements and generating ETL workflow solutions that accomplish these specifications.

6.2 Contributions

The contributions of this dissertation include a model that captures ETL domain knowledge, a planning strategy that incorporates this knowledge for the automatic generation of workflows, and a language that maps workflow requirements to higher-level abstractions called "intents."

To capture ETL domain knowledge and how data is transformed as a result of workflow execution, the proposed model has extended the usual representation of fields to include field attributes to encode the content type semantics and state of data, expressing knowledge that was previously obfuscated to all but domain experts and implicit in workflow execution. Using the extended field representation and propositional logic, operators were enhanced with preconditions and postconditions, providing a computational representation of the input requirements of operators, the distinct results produced by their execution, and how options impact their behavior.

The modeling framework transforms ETL workflow specification to an intent declaration and treats workflow planning as a search space problem to automatically generate the

corresponding workflows. The proposed workflow planning algorithm provides an informed search strategy, incorporating domain knowledge to reduce the search space to only operators that transition the workflow closer to the goal state and have a valid input mapping from the available workflow fields. To optimize this execution of this planning strategy, a way to normalize a compound operator into a collection of simpler operators was also developed.

This work has also established a mapping between technical and higher-level specifications of data processing goals. This mapping enables the definition of domain-specific languages that can be used to express an ETL solution using high-level terms that translate to workflow specifications.

Finally, as a demonstration of the feasibility of the modeling framework, the approach has been implemented in a prototype and used successfully to model a real industrial customer data integration domain.

6.3 Future Work

This section provides directions for further research on the topic of automating ETL workflow specification. Some topics are straightforward extensions in scope that could build upon the approach presented in this paper, while others fall far enough outside of the scope of this dissertation that significant enhancements would be necessary.

6.3.1 Operator Verification

The model of ETL domain knowledge defined in the approach is not limited to being used solely for workflow generation. This model includes the preconditions and postconditions of ETL operators; knowledge necessary to verify the usage of these operators in a workflow. A possible application for future work would be to use this model to store the operator

specifications of an ETL domain and apply this knowledge (whether as an overlay for an existing workflow tool or built into a new workflow tool) to verify that a workflow is syntactically well-formed, either during or after workflow construction. The benefit of such an application would be comparable to compile-time type safety in programming languages [35]. Users could be assured that the ETL workflow solutions they have constructed are structurally sound, avoiding errors that would not arise until runtime if any operator mapping was invalid. This could also help with the maintenance of ETL workflows, identifying errors with operator usage as a domain's workflow system matures and changes over time.

6.3.2 Correctness

Consider that even after an ETL workflow has been decided upon (whether as part of a manual or automated process) it is difficult to prove that the workflow is "correct". Here, the term correct is used in the sense of verification that a workflow satisfies each of its originating business intents. This use is comparable to proving the correctness of programs [36], [37]. The difficulty in proving the correctness of traditional, manually-specified ETL workflow arises from the dissociation between the business intents and the workflow specifications: business intents are high level (and often not specified explicitly) while workflow specifications are low level. With today's manual workflows, verification is informal, relying upon the experience of domain experts and/or iterative testing to accept a workflow as a fit solution. However, this provides no guarantee of correctness. The knowledge of a domain expert is not absolute and testing cannot feasibly have complete coverage. To guarantee correctness, a formal verification is needed. While an attempt can be made utilizing statistics of the workflow's performance on the actual data and relating that back to the intents, this would provide only a shallow affirmation and could not be considered a proof in the strict sense. A contribution of this dissertation, however, was

the definition of formal specifications for operators and intents; preconditions and postconditions. Given such formal specifications, future work could research using formal verification techniques to derive a proof of workflow correctness for a given workflow intent.

6.3.3 Equivalence

As was noted in Chapter 3, it is possible for a particular goal state to be attained by multiple, structurally distinct ETL workflows. Possible cases included:

- The same collection of operators arranged in different sequences.
- Multiple operators with redundant functionality that are simply substituted for one another.
- Numerous distinct input mappings for the same sequences of operators.
- Extraneous option settings for the same sequences of operators.

Of particular interest for further research was the observation that one or more such workflows may be considered equivalent; being structurally different but producing equivalent results. Determination of workflow equivalence would enable such seemingly distinct ETL workflows to be consolidated, alleviating unnecessary computational overhead in the planner and refining the result set for the user by pruning the excess choices. This concept of equivalence is considered analogous to its application in query optimization. As surveyed in Chapter 2, a query optimizer uses identity statements to recognize and reduce equivalent query plans or substitute operations in a plan to produce the same result at a cheaper cost. Research into discovering such identities for ETL workflow could allow for the determination of workflow equivalence.

6.3.3 Optimization

Complex ETL workflows can contain hundreds of operations. The more complicated a domain's ETL workflow system becomes, the more difficult workflow construction becomes. Another concern for such domains, however, is the increasing difficulty of workflow optimization. Optimization of a workflow refers to seeking to improve the workflow in some sense, such as:

- Reducing the financial cost to the client.
- Reducing the cost to the business in employee hours.
- Reducing the cost to the business in operations performed.
- Minimizing workflow runtime.

While optimization provides several potential benefits, it becomes infeasible even for a domain expert to manually optimize complex workflows. Workflow optimization is therefore considered an interesting future topic to explore.

Optimization would require equivalence relations for workflow operations similar to algebraic transformations like the commutative, associative, and distributive rules. These could be used to transform workflows from one form to another, similar to the way relational database query plans are optimized. An additional need would be to enhance the approach presented in this work with a cost model. Domain users could then assign more or less cost to particular aspects of ETL workflows. Using this cost model, a simple use case would be for the workflow solutions produced to include ranking in terms of these configurable costs. A more complex use case would be to incorporate the cost-based optimization directly into the workflow engine, affecting the operators selected, option mappings, and input field mappings based on the cost configuration.

6.3.4 Data Heritage

An additional aspect of dealing with complex ETL workflows is that it is difficult to track the heritage of the data. As the data fields pass through various operators of a given workflow, the values may or may not be transformed. Statistics in the form of percentage changed can be gathered, but it is more difficult to track the exact effect of each operator on each data field. Still, traceability of data heritage can be beneficial towards accurate pricing of solutions, licensing, and even just internal debugging. Accordingly, data heritage is another area for future research.

As an extension of the concepts described in this dissertation, one approach could be to use attributed fields during workflow execution to determine the heritage of data fields through an ETL workflow. Specifically, instead of only utilizing attributed fields during the planning of ETL workflow, extend the functionality of the operators to append meta-data in the form of state attributes to each data field according to the actual transformations that were performed to the value of each data field during execution of a workflow. The post-execution output of the workflow could then be used to trace the exact path each data field took through the operators of an ETL workflow.

6.3.6 Generic Set Operators

In defining the scope of this dissertation, it was asserted that operators must have well-defined functionality with a known set of inputs and outputs. Consequently, generic set operators, such as SELECT and UNION, fell outside this scope as they could accept variable inputs and would return variable outputs based on variable criteria. For example, address data could be mapped into a select operation configured to only return addresses that are in the US in

one instance, while in another instance name data could be mapped into a select operation configured to only return female names. However, such operations can be needed to support the control of data flow in an ETL workflow.

Adding support for such operators, while challenging, may be similar to relational database query planning and optimization. This area is well understood by the database community. The ETL operators in this dissertation can be viewed as domain-specific mappings that could augment relational operators. Our ETL operators can be viewed as variants of a generic MAP operator that sequences through the tuples in a set and applies a user-defined or domain-specific function (which may be parameterized with options) to input fields to produce output fields.

Adding support for such operators may also be similar to the application of generics in programming languages. With generics, the actual content type of inputs would not need to be specified in the assertions, but would instead be inferred from the input mapping. The target of the operation's criteria would then similarly be inferred from the input mapping. For example, if address data is mapped to the input, then the operation's criteria would have to be on this address data. This leaves the actual specification of the criteria, which would have to in the form of state attributes to hold significance to the workflow engine, or would require enhancements to both the model and workflow engine.

6.3.7 Intermediate Goals

During execution of its planning algorithm, the workflow engine evaluates available operators against the remaining goals to select candidates for the next steps in the workflow plan. These candidates are restricted to operators with postconditions that intersect the assertions of the remaining goals. Other operators are excluded to ensure that the workflow plan is composed

only of candidates that bring the workflow progressively closer to the goal state. A consequence of this focused planning criteria is that the workflow engine cannot accommodate intermediate data processing goals; requirements that are not covered by the goal state but may need to be fulfilled in order to satisfy the preconditions of operators that transition the workflow closer to the goal state.

In an attempt to expand the possible solutions the workflow engine can produce, the planning algorithm could be enhanced to use an approach such as backtracking. At a high level, the backtracking logic could initiate when the workflow engine reaches a "dead end" in planning. The preconditions of the operators that are needed to transition closer to the goal state would then be set as an intermediate goal state and the workflow engine would return to solving the original goal state once the intermediate goal state is achieved. Further, this logic could be performed recursively to accommodate scenarios that may require multiple intermediate goal states.

Of course, a drawback of backtracking would be the increased complexity for the workflow engine. But a challenge to consider if exploring any such enhancement would be how to limit the evaluation of intermediate goals; allowing the workflow engine to attempt to accommodate such scenarios, while preventing it from doing so infinitely. One possibility to explore is to give such goals higher cost and favor best cost expansions for plan generation.

6.3.8 Input Mappings

A limitation of the prototype workflow engine is that it only attempts to find a single, valid input mapping when adding an operator to the workflow plan. While sufficient for the purposes of this dissertation, future data processing goals could necessitate that the set of input mappings being considered be broadened. One extreme would be to consider all valid input

mappings of a particular operator as next possible steps in the workflow plan. Potential drawbacks, however, would include an overflow of workflow solutions being returned from the workflow engine for the user to choose from, as well as increased computational complexity. A satisfactory solution would therefore have to discern which input mappings are considered "meaningful" with respect to the given domain and data processing goals, in order to achieve a balance between the potential drawbacks and benefits.

6.3.9 Goal Indexing

When considering ETL workflow generation for domains with complex workflow systems, a concern is performance. With respect to this concern, a potential area for performance improvement in the prototype would be in the logic that searches for operators that will transition the workflow closer to the goal state. As explained, the prototype evaluates the remaining goals against every postcondition of every available operator. The performance of evaluating in such a manner, however, can degrade significantly as the number of goal specifications and possible postconditions increase. Research into a more efficient and more scalable means of finding operators that (partially) satisfy a goal state could therefore be beneficial.

One possible approach to providing this enhancement would be a means of indexing the postconditions of the available operators for faster lookup. For example, if each postcondition was indexed by content type and state attributes, finding goal matches would require far fewer lookups. The performance gain of such an approach could then be evaluated against the cost of building and storing the index.

6.3.10 Caching

Besides degradation of runtime performance, another cost that is of increasing concern as ETL workflow systems become more complex is memory consumption. As the complexity of a domain's workflow system increases, the size of the data utilized by the workflow engine's planning algorithm (e.g. data fields, operators, and planning tree) increases. In the prototype's current implementation, this data is kept in main memory. Consequently, the workflow complexity that the prototype can handle is limited by the size of main memory.

A potential strategy to remove this memory limitation is to leverage the prototype's integration with a relational database. While only domain knowledge is currently stored in the database, additional tables could be added to provide a caching mechanism for the workflow engine. Candidate data for caching could include the details of the workflow plan and even the specifications of the decomposed operators. Such an approach would allow the prototype to solve problems even for more workflow systems, with the tradeoff being performance for retrieval and persistence of this data.

6.3.11 Nested Intent Statements

The primary function of the intent language was abstraction of low level data processing goals using high level terms/statements to allow even casual users to understand and employ the language to generate ETL workflows. Though this level of functionality was sufficient for the scope of this work, continued research focused on the intent language could identify additional opportunities to expand its functionality and improve its ease of use.

One candidate functionality enhancement is for the language to support nested intent statements. Nesting of intent statements refers to the ability to combine intents in a hierarchical

form, consisting of inner and outer intents that form a comprehensive intent statement. This can be formally defined according to the following grammar:

- The set of terminal symbols $\sum$ = { I, (,) }
- The set of non-terminal symbols N = { S }
- S is the start symbol
- The set of production rules P:
    1. S => I
    2. S => I (S)

In this formal grammar, the terminal symbol *I* represents an intent statement and the parentheses establish the inner/outer nested hierarchy of intent statements.

The purpose of this enhancement is to allow users to compose more precise statements of their intent. One example of the need for such elevated precision is the statement "run edit checks after geocoding addresses". In this example, the user is specifying a directive on the order of operation, that geocoding addresses before running edit checks would not attain their data processing goals. Another example is the statement "only geocode addresses that are in the US". This example specifies a directive on data flow, requiring data records to be subdivided (based on some criteria) to produce the intended result.

Research on this topic can leverage existing work that allows similar directives to be specified based on statement composition. SQL supports nesting in the form of sub-queries, where the outer query can utilize the results of the inner query. Also, Menu-based Natural Languages enables users to form statements by chaining commands together [38].

6.3.12 Intent Relationships

Another enhancement to the intent language involves discerning relationships among intents. Recall that a single intent may encapsulate a set of workflow requirements. Among all of the intents defined for a domain, one or more intents may be related based on the workflow requirements they are mapped to. These relationships can include:

- Implicit – The workflow requirements of one intent are a sub-set of another; therefore, one intent is implicitly specified whenever the other intent is.
- Conflict – Both intents cannot be satisfied by the same workflow. For example, one intent asserts that the goal state has a particular state attribute, while the other asserts that the goal state cannot have that state attribute.

Recognition of such relationships among intents can provide a more powerful, easier to use language. Applied to the GUI of the prototype, when an intent is selected that has conflicts with other available intents, these conflicting intents could be disabled to prevent the user from choosing an unattainable goal state. Similarly, implicit intents can be visually indicated whenever an intent that encompasses their workflow requirements is selected to both prevent redundancy and provide clarity for the user.

6.3.13 Result Filtering

A noted problem of ETL workflow specification was that requirements can be ambiguous, requiring a domain expert to iteratively refactor an ETL workflow until a satisfactory solution is agreed upon. While such manual refactoring is addressed by the approach of this paper, intent statements that are ambiguous can similarly produce a larger number of possible solutions than desired by the user. It therefore follows that a beneficial future enhancement

would be a means of discerning the differences between workflow solutions, allowing users to filter the results generated by the workflow engine.

Towards such an enhancement, consider an approach using the goal state specified by the user as a baseline for comparison. Comparing this planned goal state against the actual goal state of the workflow engine's results, each workflow solution will likely satisfy additional goals that were not part of the planned goal state. These excess goals then provide the means to differentiate and prune workflows from the result set. Incorporating this idea into the prototype, a resolution screen could be added to the GUI that allows a user to refine the result set by selecting from the additional goals to use as a filter.

REFERENCES

[1] No Author, "Data, Data Everywhere," The Economist, February 25, 2010, URL: http://www.economist.com/node/15557443

[2] "XDATA," DARPA Research Project Announcement, March 29, 2012.

[3] "Core Techniques and Technologies for Advancing Big Data," NSF Program Solicitation, March 29, 2012.

[4] President's Council of Advisors on Science and Technology "Designing a Digital Future: Federally Funded Research and Development in Networking and Information Technology," Report to the President and Congress, December 2010.

[5] IBM Corporation, *InfoSphere DataStage*, URL: http://www-01.ibm.com/software/data/infosphere/datastage/, Accessed: July 04 2012.

[6] Microsoft, *SQL Server Integration Services*, URL: http://msdn.microsoft.com/en-us/sqlserver/cc511477.aspx, Accessed: July 12, 2012.

[7] Oracle Corporation, *Oracle Data Integrator*, URL: http://www.oracle.com/technetwork/middleware/data-integrator/overview/index.html, Accessed: July 12, 2012.

[8] Royce, W., "Managing the Development of Large Software Systems," *Proceedings of IEEE WESCON*, pp. 1-9, Los Angeles, CA, August 1970.

[9] Aalst, W., Hee, K., Workflow Management: Models, Methods, and Systems, MIT Press, Cambridge, MA, March 1, 2004.

[10] Kelly, S., Tolvanen, J., Domain-Specific Modeling: Enabling Full Code Generation, Wiley-IEEE Computer Society Press, March, 2008.

[11] Kifer, M., Bernstein, A., Lewis, P., Database Systems An Application-Oriented Approach, Pearson Education, 2006.

[12] Chamberlin, D., Boyce, R., "SEQUEL: A Structured English Query Language," *SIGFIDET Workshop on Data Description, Access, and Control (SIGFET '74),* New York, NY, 1974.

[13] Thompson, C., Li, W., Deneke, W., Eno, J., "Towards a Domain-Specific Modeling Language (DSML) for Customer Data Integration (CDI)," *Conference on Applied Research in Information Technology*, Acxiom Laboratory for Applied Research, Conway AR, February, 2007.

[14] Coppin, B., Artificial Intelligence Illuminated, Jones and Bartless Publishers, 2004.

[15] Fikes, R., Nilsson, N., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, 2, (3-4), 189-208, 1971.

[16] Barwise, J., "An Introduction to First-Order Logic," chapter in: Handbook of Mathematical Logic, Chapter A.1, 6-47. Elsevier, Amsetdam, The Netherlands, 1977.

[17] Sun, T., "An Intelligent Wizard for Automatic Workflow Modeling," *IEEE International Conference on Systems, Man, and Cybernetics*, pp. 2742-2746, Taipei, China, October 8-11, 2006.

[18] Zhang, S., Xiang, Y., Shen, Y., Shi, M., "Knowledge Modeling and Optimization In Pattern-Oriented Workflow Generation," *Twelfth International Conference on Computer Supported Cooperative Work in Design*, pp. 636-642, Xi'an, China, April 16-18, 2008.

[19] Ambite, J., Kapoor, D., "Automatic Generation of Data Processing Workflows for Transportation Modeling," *Proceedings of the Eighth Annual International Conference on Digital Government Research: Bridging Disciplines & Domains*, Philadelphia, PA, May 20-23, 2007.

[20] Xiao, Z., Thompson, C., Li, W., "A Practical Data Processing Workflow Automation System in Acxiom Grid Architecture," *International Workshop on Workflow Systems in Grid Environments (WSGE06)*, Changsha, China, October 21-23, 2006.

[21] Thompson, C., Li, W., Xiao, Z., "Workflow Planning on a Grid," *Architectural Perspectives column, IEEE Internet Computing*, pp. 74-77, January-February, 2007.

[22] Xiao, Z., Thompson, C., Li, W., "Automating Workflow for Data Processing in Grid Architecture," *International Conference on Information and Knowledge Engineering (IKE '06)*, Las Vegas, NV, June 26-29, 2006.

[23] Phillips, R., Benham, P., Li, W., Beavers, G., Thompson, C., "Automating File Schema Recognition Via Content-Based Oracles," *International Conference on Information and Knowledge Engineering (IKE '08)*, Las Vegas, NV, July 14-17, 2008.

[24] Eno, J., Thompson, C., Li, W., Deneke, W., "Enhanced Workflow Service Modeling," *Conference on Applied Research in Information Technology*, Acxiom Laboratory for Applied Research, Conway AR, February, 2008.

[25] Booch, G., Object-oriented Analysis and Design with Applications (3rd Ed.), Addison-Wesley, Upper Saddle River, NJ, 2007.

[26] Schmidt, D., "Model-Driven Engineering," *IEEE Computer*, February, 2006.

[27] Martin, R., Clean Code: A Handbook of Agile Software Craftsmanship, Prentice Hall, Upper Saddle River, NJ, August 11, 2008.

[28] Microsoft, *SQL Server*, URL: http://www.microsoft.com/sqlserver/, Accessed: July 12, 2012.

[29] Microsoft, *Entity Framework*, URL: http://msdn.microsoft.com/en-us/library/bb399572.aspx, Accessed: July 12, 2012.

[30] Evans, E., Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley, Upper Saddle River, NJ, 2004.

[31] Microsoft, *Windows Presentation Foundation*, URL: http://msdn.microsoft.com/en-us/library/ms754130.aspx, Accessed: July 12, 2012.

[32] Harris, D., Harris, S., Digital Design and Computer Architecture, Morgan Kaufmann, San Francisco, CA, March 16, 2007.

[33] Martin, R., Martin, M., Agile Principles, Patterns, and Practices in C#, Prentice Hall, Upper Saddle River, NJ, July 30, 2006.

[34] "IEEE Standard for Software Verification and Validation," *IEEE Std 1012-2004*, pp. 1-11-, June 8, 2005.

[35] Freeman, A., Introducing Visual C# 2010, pp. 22-23, Apress, New York, NY, November 24, 2010.

[36] Gries, D., The Science of Programming, Springer-Verlag, New York, NY, 1981.

[37] Hantler, S., King, J., "An Introduction to Proving the Correctness of Programs," *ACM Computing Surveys (CSUR)*, vol. 8, pp. 331-353, New York, NY, September 1976.

[38] Tennant, H., Ross, K., Saenz, R., Miller, J., Thompson, C., "Menu-Based Natural Language Understanding," *21st Meeting of the Association for Computational Linguistics (ACL)*, MIT, June, 1983.

[39] Deneke, W., Li, W., Thompson, C., "State Driven Semantic Modeling of Operators in ETL Workflow," *Journal of Computing Sciences in Colleges*, vol. 25, no. 5, pp. 27-33, May 2010.

[40] Deneke, W., Eno, J., Li, W., Thompson, C., Talburt, J., Loghry, J., Nash, D., Stires, J., "Towards a Domain-Specific Modeling Language for Customer Data Integration Workflow," *Third International Conference on Grid and Pervasive Computing*, vol. 0, pp. 49-56, Kunming, China, May 25-28, 2008.

APPENDIX A: MAILING ENHANCEMENT OPERATOR SPECIFICATIONS

A.1 AddressEditCheck

```
AddressEditCheck
{

   INPUT FIELDS
   {
      FIRSTNAME
      MIDDLENAME
      LASTNAME
      BUSINESSNAME
      PRIMARYADDRESS
      SECONDARYADDRESS
      CITYNAME
      STATEABBREV
      ZIPCODE
      ADDRESSFLAGS
   }

   OUTPUT FIELDS
   {
   }

   PRECONDITIONS
   {
   }

   OPTIONS
   {

      EditCheckAddresses
      {

         OPTION SETTINGS
         {

            N
            {

               PRECONDITIONS
               {
               }
            }

            Y
            {

               PRECONDITIONS
               {

                  PRECONDITION
```

```
                {
                  [PRIMARYADDRESS].REQUIRED
                  & [PRIMARYADDRESS :: Address.Primary ]
                  & [CITYNAME].REQUIRED
                  & [CITYNAME :: Address.City ]
                  & [STATEABBREV].REQUIRED
                  & [STATEABBREV :: Address.State ]
                  & [ZIPCODE].REQUIRED
                  & [ZIPCODE :: Address.Zip ]

                  POSTCONDITION
                  {
                    [PRIMARYADDRESS :: Address.Primary +AddressEditChecked]
                    [CITYNAME :: Address.City +AddressEditChecked]
                    [STATEABBREV :: Address.State +AddressEditChecked]
                    [ZIPCODE :: Address.Zip +AddressEditChecked]
                  }
                }
              }
            }
          }
        }
      }
    }
```

A.2 AddressEnhance

```
AddressEnhance
{

   INPUT FIELDS
   {
      PRIMARYADDRESSLINE
      SECONDARYADDRESSLINE
      BUSINESSNAME
      CITY
      STATE
      ZIP
      URBANIZATIONNAME
      FIRSTNAME
      MIDDLENAME
      LASTNAME
      LASTNAMESUFFIX
      TITLECODE
      GENDERCODE
   }

   OUTPUT FIELDS
   {
      AAPRINUMBERBEFORECOA
      AAPREDIR1BEFORECOA
      AAPRINAMEBEFORECOA
      AASUFFIX1BEFORECOA
      AAPOSTDIR1BEFORECOA
      AASECUNITDESBEFORECOA
      AASECNUMBERBEFORECOA
      AAOUTPRURBBEFORECOA
      AAOUTBUSADDRESSBEFORECOA
      AACITYNAMEBEFORECOA
      AASTATEABREVBEFORECOA
      AAZIPCODEBEFORECOA
      AAADDONCODEBEFORECOA
      AAOUTPRIMADDRESSBEFORECOA
      AAOUTSECADDRESSBEFORECOA
      AAADDRESSFLAGSBEFORECOA
      AAPRINUMBERAFTERCOA
      AAPREDIR1AFTERCOA
      AAPRINAMEAFTERCOA
      AASUFFIX1AFTERCOA
      AAPOSTDIR1AFTERCOA
      AASECUNITDESAFTERCOA
      AASECNUMBERAFTERCOA
      AAOUTPRURBAFTERCOA
      AAOUTBUSADDRESSAFTERCOA
      AACITYNAMEAFTERCOA
      AASTATEABREVAFTERCOA
      AAZIPCODEAFTERCOA
      AAADDONCODEAFTERCOA
      AAOUTPRIMADDRESSAFTERCOA
      AAOUTSECADDRESSAFTERCOA
```

```
    AAADDRESSFLAGSAFTERCOA
    COANEWZIPCODECHAR
    COANEWADDONCODECHAR
    COANEWHOUSENUMBER
    COANEWSTREETPREDIR
    COANEWSTREETNAME
    COANEWSTREETSUFFIX
    COANEWSTREETPOSTDIR
    COANEWUNITDESIGNATOR
    COANEWSECONDARYNUM
    COANEWCITY
    COANEWSTATE
    COANEWURBANIZATIONNAME
  }

  PRECONDITIONS
  {

    PRECONDITION
    {
      [PRIMARYADDRESSLINE].REQUIRED
      & [PRIMARYADDRESSLINE :: Address.Primary ]
      & [CITY].REQUIRED
      & [CITY :: Address.City ]
      & [STATE].REQUIRED
      & [STATE :: Address.State ]

      POSTCONDITION
      {
        [AAPRINUMBERBEFORECOA :: Address.PrimaryNumber
+UspsStandardized+DeliverabilityValidated]
        [AAPREDIR1BEFORECOA :: Address.PreDirectional +UspsStandardized+DeliverabilityValidated]
        [AAPRINAMEBEFORECOA :: Address.PrimaryName +UspsStandardized+DeliverabilityValidated]
        [AASUFFIX1BEFORECOA :: Address.Suffix +UspsStandardized+DeliverabilityValidated]
        [AAPOSTDIR1BEFORECOA :: Address.PostDirectional +UspsStandardized+DeliverabilityValidated]
        [AASECUNITDESBEFORECOA :: Address.SecondaryUnitDesignator
+UspsStandardized+DeliverabilityValidated]
        [AASECNUMBERBEFORECOA :: Address.SecondaryNumber
+UspsStandardized+DeliverabilityValidated]
        [AAOUTPRURBBEFORECOA :: Address.UrbanizationName
+UspsStandardized+DeliverabilityValidated]
        [AAOUTBUSADDRESSBEFORECOA :: Address.Business +UspsStandardized+DeliverabilityValidated]
        [AACITYNAMEBEFORECOA :: Address.City +UspsStandardized+DeliverabilityValidated]
        [AASTATEABREVBEFORECOA :: Address.State +UspsStandardized+DeliverabilityValidated]
        [AAZIPCODEBEFORECOA :: Address.Zip +UspsStandardized+DeliverabilityValidated]
        [AAADDONCODEBEFORECOA :: Address.Zip4 +UspsStandardized+DeliverabilityValidated]
        [AAOUTPRIMADDRESSBEFORECOA :: Address.Primary
+UspsStandardized+DeliverabilityValidated]
        [AAOUTSECADDRESSBEFORECOA :: Address.Secondary
+UspsStandardized+DeliverabilityValidated]
        [AAADDRESSFLAGSBEFORECOA :: Address.Flags +UspsStandardized]
        [AAPRINUMBERAFTERCOA :: Address.PrimaryNumber +UspsStandardized+DeliverabilityValidated]
        [AAPREDIR1AFTERCOA :: Address.PreDirectional +UspsStandardized+DeliverabilityValidated]
        [AAPRINAMEAFTERCOA :: Address.PrimaryName +UspsStandardized+DeliverabilityValidated]
        [AASUFFIX1AFTERCOA :: Address.Suffix +UspsStandardized+DeliverabilityValidated]
        [AAPOSTDIR1AFTERCOA :: Address.PostDirectional +UspsStandardized+DeliverabilityValidated]
```

```
        [AASECUNITDESAFTERCOA :: Address.SecondaryUnitDesignator
+UspsStandardized+DeliverabilityValidated]
        [AASECNUMBERAFTERCOA :: Address.SecondaryNumber
+UspsStandardized+DeliverabilityValidated]
        [AAOUTPRURBAFTERCOA :: Address.UrbanizationName
+UspsStandardized+DeliverabilityValidated]
        [AAOUTBUSADDRESSAFTERCOA :: Address.Business +UspsStandardized+DeliverabilityValidated]
        [AACITYNAMEAFTERCOA :: Address.City +UspsStandardized+DeliverabilityValidated]
        [AASTATEABREVAFTERCOA :: Address.State +UspsStandardized+DeliverabilityValidated]
        [AAZIPCODEAFTERCOA :: Address.Zip +UspsStandardized+DeliverabilityValidated]
        [AAADDONCODEAFTERCOA :: Address.Zip4 +UspsStandardized+DeliverabilityValidated]
        [AAOUTPRIMADDRESSAFTERCOA :: Address.Primary +UspsStandardized+DeliverabilityValidated]
        [AAOUTSECADDRESSAFTERCOA :: Address.Secondary +UspsStandardized+DeliverabilityValidated]
        [AAADDRESSFLAGSAFTERCOA :: Address.Flags +UspsStandardized]
      }
    }

    PRECONDITION
    {
      [PRIMARYADDRESSLINE].REQUIRED
      & [PRIMARYADDRESSLINE :: Address.Primary ]
      & [ZIP].REQUIRED
      & [ZIP :: Address.Zip ]

      POSTCONDITION
      {
        [AAPRINUMBERBEFORECOA :: Address.PrimaryNumber
+UspsStandardized+DeliverabilityValidated]
        [AAPREDIR1BEFORECOA :: Address.PreDirectional +UspsStandardized+DeliverabilityValidated]
        [AAPRINAMEBEFORECOA :: Address.PrimaryName +UspsStandardized+DeliverabilityValidated]
        [AASUFFIX1BEFORECOA :: Address.Suffix +UspsStandardized+DeliverabilityValidated]
        [AAPOSTDIR1BEFORECOA :: Address.PostDirectional +UspsStandardized+DeliverabilityValidated]
        [AASECUNITDESBEFORECOA :: Address.SecondaryUnitDesignator
+UspsStandardized+DeliverabilityValidated]
        [AASECNUMBERBEFORECOA :: Address.SecondaryNumber
+UspsStandardized+DeliverabilityValidated]
        [AAOUTPRURBBEFORECOA :: Address.UrbanizationName
+UspsStandardized+DeliverabilityValidated]
        [AAOUTBUSADDRESSBEFORECOA :: Address.Business +UspsStandardized+DeliverabilityValidated]
        [AACITYNAMEBEFORECOA :: Address.City +UspsStandardized+DeliverabilityValidated]
        [AASTATEABREVBEFORECOA :: Address.State +UspsStandardized+DeliverabilityValidated]
        [AAZIPCODEBEFORECOA :: Address.Zip +UspsStandardized+DeliverabilityValidated]
        [AAADDONCODEBEFORECOA :: Address.Zip4 +UspsStandardized+DeliverabilityValidated]
        [AAOUTPRIMADDRESSBEFORECOA :: Address.Primary
+UspsStandardized+DeliverabilityValidated]
        [AAOUTSECADDRESSBEFORECOA :: Address.Secondary
+UspsStandardized+DeliverabilityValidated]
        [AAADDRESSFLAGSBEFORECOA :: Address.Flags +UspsStandardized]
        [AAPRINUMBERAFTERCOA :: Address.PrimaryNumber +UspsStandardized+DeliverabilityValidated]
        [AAPREDIR1AFTERCOA :: Address.PreDirectional +UspsStandardized+DeliverabilityValidated]
        [AAPRINAMEAFTERCOA :: Address.PrimaryName +UspsStandardized+DeliverabilityValidated]
        [AASUFFIX1AFTERCOA :: Address.Suffix +UspsStandardized+DeliverabilityValidated]
        [AAPOSTDIR1AFTERCOA :: Address.PostDirectional +UspsStandardized+DeliverabilityValidated]
        [AASECUNITDESAFTERCOA :: Address.SecondaryUnitDesignator
+UspsStandardized+DeliverabilityValidated]
```

```
            [AASECNUMBERAFTERCOA :: Address.SecondaryNumber
+UspsStandardized+DeliverabilityValidated]
            [AAOUTPRURBAFTERCOA :: Address.UrbanizationName
+UspsStandardized+DeliverabilityValidated]
            [AAOUTBUSADDRESSAFTERCOA :: Address.Business +UspsStandardized+DeliverabilityValidated]
            [AACITYNAMEAFTERCOA :: Address.City +UspsStandardized+DeliverabilityValidated]
            [AASTATEABREVAFTERCOA :: Address.State +UspsStandardized+DeliverabilityValidated]
            [AAZIPCODEAFTERCOA :: Address.Zip +UspsStandardized+DeliverabilityValidated]
            [AAADDONCODEAFTERCOA :: Address.Zip4 +UspsStandardized+DeliverabilityValidated]
            [AAOUTPRIMADDRESSAFTERCOA :: Address.Primary +UspsStandardized+DeliverabilityValidated]
            [AAOUTSECADDRESSAFTERCOA :: Address.Secondary +UspsStandardized+DeliverabilityValidated]
            [AAADDRESSFLAGSAFTERCOA :: Address.Flags +UspsStandardized]
        }
    }
  }

  OPTIONS
  {

    PerformDeliverySequencing
    {

      OPTION SETTINGS
      {

        N
        {

          PRECONDITIONS
          {
          }
        }

        Y
        {

          PRECONDITIONS
          {

            PRECONDITION
            {
              NONE

              POSTCONDITION
              {
                [AAPRINUMBERBEFORECOA :: Address.PrimaryNumber +DeliverySequenced]
                [AAPREDIR1BEFORECOA :: Address.PreDirectional +DeliverySequenced]
                [AAPRINAMEBEFORECOA :: Address.PrimaryName +DeliverySequenced]
                [AASUFFIX1BEFORECOA :: Address.Suffix +DeliverySequenced]
                [AAPOSTDIR1BEFORECOA :: Address.PostDirectional +DeliverySequenced]
                [AASECUNITDESBEFORECOA :: Address.SecondaryUnitDesignator +DeliverySequenced]
                [AASECNUMBERBEFORECOA :: Address.SecondaryNumber +DeliverySequenced]
                [AAOUTPRURBBEFORECOA :: Address.UrbanizationName +DeliverySequenced]
                [AAOUTBUSADDRESSBEFORECOA :: Address.Business +DeliverySequenced]
                [AACITYNAMEBEFORECOA :: Address.City +DeliverySequenced]
                [AASTATEABREVBEFORECOA :: Address.State +DeliverySequenced]
```

```
            [AAZIPCODEBEFORECOA :: Address.Zip +DeliverySequenced]
            [AAADDONCODEBEFORECOA :: Address.Zip4 +DeliverySequenced]
            [AAOUTPRIMADDRESSBEFORECOA :: Address.Primary +DeliverySequenced]
            [AAOUTSECADDRESSBEFORECOA :: Address.Secondary +DeliverySequenced]
            [AAADDRESSFLAGSBEFORECOA :: Address.Flags +DeliverySequenced]
            [AAPRINUMBERAFTERCOA :: Address.PrimaryNumber +DeliverySequenced]
            [AAPREDIR1AFTERCOA :: Address.PreDirectional +DeliverySequenced]
            [AAPRINAMEAFTERCOA :: Address.PrimaryName +DeliverySequenced]
            [AASUFFIX1AFTERCOA :: Address.Suffix +DeliverySequenced]
            [AAPOSTDIR1AFTERCOA :: Address.PostDirectional +DeliverySequenced]
            [AASECUNITDESAFTERCOA :: Address.SecondaryUnitDesignator +DeliverySequenced]
            [AASECNUMBERAFTERCOA :: Address.SecondaryNumber +DeliverySequenced]
            [AAOUTPRURBAFTERCOA :: Address.UrbanizationName +DeliverySequenced]
            [AAOUTBUSADDRESSAFTERCOA :: Address.Business +DeliverySequenced]
            [AACITYNAMEAFTERCOA :: Address.City +DeliverySequenced]
            [AASTATEABREVAFTERCOA :: Address.State +DeliverySequenced]
            [AAZIPCODEAFTERCOA :: Address.Zip +DeliverySequenced]
            [AAADDONCODEAFTERCOA :: Address.Zip4 +DeliverySequenced]
            [AAOUTPRIMADDRESSAFTERCOA :: Address.Primary +DeliverySequenced]
            [AAOUTSECADDRESSAFTERCOA :: Address.Secondary +DeliverySequenced]
            [AAADDRESSFLAGSAFTERCOA :: Address.Flags +DeliverySequenced]
        }
      }
    }
  }
}

PerformChangeOfAddress
{

  OPTION SETTINGS
  {

    N
    {

      PRECONDITIONS
      {
      }
    }

    Y
    {

      PRECONDITIONS
      {

        PRECONDITION
        {
          [FIRSTNAME].REQUIRED
          & [FIRSTNAME :: Name.First ]
          & [LASTNAME].REQUIRED
          & [LASTNAME :: Name.Last ]

          POSTCONDITION
```

```
  {
    [COANEWZIPCODECHAR :: Address.Zip +AddressChange]
    [COANEWADDONCODECHAR :: Address.Zip4 +AddressChange]
    [COANEWHOUSENUMBER :: Address.PrimaryNumber +AddressChange]
    [COANEWSTREETPREDIR :: Address.PreDirectional +AddressChange]
    [COANEWSTREETNAME :: Address.PrimaryName +AddressChange]
    [COANEWSTREETSUFFIX :: Address.Suffix +AddressChange]
    [COANEWSTREETPOSTDIR :: Address.PostDirectional +AddressChange]
    [COANEWUNITDESIGNATOR :: Address.SecondaryUnitDesignator +AddressChange]
    [COANEWSECONDARYNUM :: Address.SecondaryNumber +AddressChange]
    [COANEWCITY :: Address.City +AddressChange]
    [COANEWSTATE :: Address.State +AddressChange]
    [COANEWURBANIZATIONNAME :: Address.UrbanizationName +AddressChange]
    [AAPRINUMBERAFTERCOA :: Address.PrimaryNumber +AddressChange]
    [AAPREDIR1AFTERCOA :: Address.PreDirectional +AddressChange]
    [AAPRINAMEAFTERCOA :: Address.PrimaryName +AddressChange]
    [AASUFFIX1AFTERCOA :: Address.Suffix +AddressChange]
    [AAPOSTDIR1AFTERCOA :: Address.PostDirectional +AddressChange]
    [AASECUNITDESAFTERCOA :: Address.SecondaryUnitDesignator +AddressChange]
    [AASECNUMBERAFTERCOA :: Address.SecondaryNumber +AddressChange]
    [AAOUTPRURBAFTERCOA :: Address.UrbanizationName +AddressChange]
    [AAOUTBUSADDRESSAFTERCOA :: Address.Business +AddressChange]
    [AACITYNAMEAFTERCOA :: Address.City +AddressChange]
    [AASTATEABREVAFTERCOA :: Address.State +AddressChange]
    [AAZIPCODEAFTERCOA :: Address.Zip +AddressChange]
    [AAADDONCODEAFTERCOA :: Address.Zip4 +AddressChange]
    [AAOUTPRIMADDRESSAFTERCOA :: Address.Primary +AddressChange]
    [AAOUTSECADDRESSAFTERCOA :: Address.Secondary +AddressChange]
    [AAADDRESSFLAGSAFTERCOA :: Address.Flags +AddressChange]
  }
}

PRECONDITION
{
  [BUSINESSNAME].REQUIRED
  & [BUSINESSNAME :: Name.Business ]

  POSTCONDITION
  {
    [COANEWZIPCODECHAR :: Address.Zip +AddressChange]
    [COANEWADDONCODECHAR :: Address.Zip4 +AddressChange]
    [COANEWHOUSENUMBER :: Address.PrimaryNumber +AddressChange]
    [COANEWSTREETPREDIR :: Address.PreDirectional +AddressChange]
    [COANEWSTREETNAME :: Address.PrimaryName +AddressChange]
    [COANEWSTREETSUFFIX :: Address.Suffix +AddressChange]
    [COANEWSTREETPOSTDIR :: Address.PostDirectional +AddressChange]
    [COANEWUNITDESIGNATOR :: Address.SecondaryUnitDesignator +AddressChange]
    [COANEWSECONDARYNUM :: Address.SecondaryNumber +AddressChange]
    [COANEWCITY :: Address.City +AddressChange]
    [COANEWSTATE :: Address.State +AddressChange]
    [COANEWURBANIZATIONNAME :: Address.UrbanizationName +AddressChange]
    [AAPRINUMBERAFTERCOA :: Address.PrimaryNumber +AddressChange]
    [AAPREDIR1AFTERCOA :: Address.PreDirectional +AddressChange]
    [AAPRINAMEAFTERCOA :: Address.PrimaryName +AddressChange]
    [AASUFFIX1AFTERCOA :: Address.Suffix +AddressChange]
    [AAPOSTDIR1AFTERCOA :: Address.PostDirectional +AddressChange]
```

```
            [AASECUNITDESAFTERCOA :: Address.SecondaryUnitDesignator +AddressChange]
            [AASECNUMBERAFTERCOA :: Address.SecondaryNumber +AddressChange]
            [AAOUTPRURBAFTERCOA :: Address.UrbanizationName +AddressChange]
            [AAOUTBUSADDRESSAFTERCOA :: Address.Business +AddressChange]
            [AACITYNAMEAFTERCOA :: Address.City +AddressChange]
            [AASTATEABREVAFTERCOA :: Address.State +AddressChange]
            [AAZIPCODEAFTERCOA :: Address.Zip +AddressChange]
            [AAADDONCODEAFTERCOA :: Address.Zip4 +AddressChange]
            [AAOUTPRIMADDRESSAFTERCOA :: Address.Primary +AddressChange]
            [AAOUTSECADDRESSAFTERCOA :: Address.Secondary +AddressChange]
            [AAADDRESSFLAGSAFTERCOA :: Address.Flags +AddressChange]
        }
      }
    }
  }
}

PerformGeocoding
{

  OPTION SETTINGS
  {

    None
    {

      PRECONDITIONS
      {
      }
    }

    ZipCode
    {

      PRECONDITIONS
      {

        PRECONDITION
        {
          NONE

          POSTCONDITION
          {
            [AAPRINUMBERBEFORECOA :: Address.PrimaryNumber +Geocoded]
            [AAPREDIR1BEFORECOA :: Address.PreDirectional +Geocoded]
            [AAPRINAMEBEFORECOA :: Address.PrimaryName +Geocoded]
            [AASUFFIX1BEFORECOA :: Address.Suffix +Geocoded]
            [AAPOSTDIR1BEFORECOA :: Address.PostDirectional +Geocoded]
            [AASECUNITDESBEFORECOA :: Address.SecondaryUnitDesignator +Geocoded]
            [AASECNUMBERBEFORECOA :: Address.SecondaryNumber +Geocoded]
            [AAOUTPRURBBEFORECOA :: Address.UrbanizationName +Geocoded]
            [AAOUTBUSADDRESSBEFORECOA :: Address.Business +Geocoded]
            [AACITYNAMEBEFORECOA :: Address.City +Geocoded]
            [AASTATEABREVBEFORECOA :: Address.State +Geocoded]
            [AAZIPCODEBEFORECOA :: Address.Zip +Geocoded]
```

```
                [AAADDONCODEBEFORECOA :: Address.Zip4 +Geocoded]
                [AAOUTPRIMADDRESSBEFORECOA :: Address.Primary +Geocoded]
                [AAOUTSECADDRESSBEFORECOA :: Address.Secondary +Geocoded]
                [AAPRINUMBERAFTERCOA :: Address.PrimaryNumber +Geocoded]
                [AAPREDIR1AFTERCOA :: Address.PreDirectional +Geocoded]
                [AAPRINAMEAFTERCOA :: Address.PrimaryName +Geocoded]
                [AASUFFIX1AFTERCOA :: Address.Suffix +Geocoded]
                [AAPOSTDIR1AFTERCOA :: Address.PostDirectional +Geocoded]
                [AASECUNITDESAFTERCOA :: Address.SecondaryUnitDesignator +Geocoded]
                [AASECNUMBERAFTERCOA :: Address.SecondaryNumber +Geocoded]
                [AAOUTPRURBAFTERCOA :: Address.UrbanizationName +Geocoded]
                [AAOUTBUSADDRESSAFTERCOA :: Address.Business +Geocoded]
                [AACITYNAMEAFTERCOA :: Address.City +Geocoded]
                [AASTATEABREVAFTERCOA :: Address.State +Geocoded]
                [AAZIPCODEAFTERCOA :: Address.Zip +Geocoded]
                [AAADDONCODEAFTERCOA :: Address.Zip4 +Geocoded]
                [AAOUTPRIMADDRESSAFTERCOA :: Address.Primary +Geocoded]
                [AAOUTSECADDRESSAFTERCOA :: Address.Secondary +Geocoded]
            }
        }
    }
  }
 }
}
```

A.3 AddressSelect

```
AddressSelect
{

   INPUT FIELDS
   {
      oprBUSADDRESS
      oprPRINUMBER
      oprPREDIR
      oprPRINAME
      oprSUFFIX
      oprPOSTDIR
      oprSECUNITDES
      oprSECNUMBER
      oprCITY
      oprSTATE
      oprZIP
      oprZIP4
      oprPRIADDRESS
      oprSECADDRESS
      oprURBNAME
      oprADDRESSFLAGS
      NCOABUSADDRESS
      NCOAPRINUMBER
      NCOAPREDIR
      NCOAPRINAME
      NCOASUFFIX
      NCOAPOSTDIR
      NCOASECUNITDES
      NCOASECNUMBER
      NCOACITY
      NCOASTATE
      NCOAZIP
      NCOAZIP4
      NCOAPRIADDRESS
      NCOASECADDRESS
      NCOAURBNAME
      NCOAADDRESSFLAGS
      CHANGEPLUSBUSADDRESS
      CHANGEPLUSPRINUMBER
      CHANGEPLUSPREDIR
      CHANGEPLUSPRINAME
      CHANGEPLUSSUFFIX
      CHANGEPLUSPOSTDIR
      CHANGEPLUSSECUNITDES
      CHANGEPLUSSECNUMBER
      CHANGEPLUSCITY
      CHANGEPLUSSTATE
      CHANGEPLUSZIP
      CHANGEPLUSZIP4
      CHANGEPLUSPRIADDRESS
      CHANGEPLUSSECADDRESS
      CHANGEPLUSURBNAME
   }
```

```
OUTPUT FIELDS
{
   PASBUSADDRESS
   PASPRINUMBER
   PASPREDIR
   PASPRINAME
   PASSUFFIX
   PASPOSTDIR
   PASSECUNITDES
   PASSECNUMBER
   PASCITYNAME
   PASSTATEABREV
   PASZIPCODE
   PASADDONCODE
   PASPRIADDRESS
   PASSECADDRESS
   PASPRURBANIZATION
   PASADDRESSFLAGS
}

PRECONDITIONS
{

   PRECONDITION
   {
      [oprBUSADDRESS].REQUIRED
      & [oprBUSADDRESS :: Address.Business +UspsStandardized]
      & [oprBUSADDRESS :: Address.Business -AddressChange-PremiumAddressChange]
      & [oprPRINUMBER].REQUIRED
      & [oprPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
      & [oprPRINUMBER :: Address.PrimaryNumber -AddressChange-PremiumAddressChange]
      & [oprPREDIR].REQUIRED
      & [oprPREDIR :: Address.PreDirectional +UspsStandardized]
      & [oprPREDIR :: Address.PreDirectional -AddressChange-PremiumAddressChange]
      & [oprPRINAME].REQUIRED
      & [oprPRINAME :: Address.PrimaryName +UspsStandardized]
      & [oprPRINAME :: Address.PrimaryName -AddressChange-PremiumAddressChange]
      & [oprSUFFIX].REQUIRED
      & [oprSUFFIX :: Address.Suffix +UspsStandardized]
      & [oprSUFFIX :: Address.Suffix -AddressChange-PremiumAddressChange]
      & [oprPOSTDIR].REQUIRED
      & [oprPOSTDIR :: Address.PostDirectional +UspsStandardized]
      & [oprPOSTDIR :: Address.PostDirectional -AddressChange-PremiumAddressChange]
      & [oprSECUNITDES].REQUIRED
      & [oprSECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
      & [oprSECUNITDES :: Address.SecondaryUnitDesignator -AddressChange-PremiumAddressChange]
      & [oprSECNUMBER].REQUIRED
      & [oprSECNUMBER :: Address.SecondaryNumber +UspsStandardized]
      & [oprSECNUMBER :: Address.SecondaryNumber -AddressChange-PremiumAddressChange]
      & [oprCITY].REQUIRED
      & [oprCITY :: Address.City +UspsStandardized]
      & [oprCITY :: Address.City -AddressChange-PremiumAddressChange]
      & [oprSTATE].REQUIRED
      & [oprSTATE :: Address.State +UspsStandardized]
      & [oprSTATE :: Address.State -AddressChange-PremiumAddressChange]
```

& [oprZIP].REQUIRED
& [oprZIP :: Address.Zip +UspsStandardized]
& [oprZIP :: Address.Zip -AddressChange-PremiumAddressChange]
& [oprZIP4].REQUIRED
& [oprZIP4 :: Address.Zip4 +UspsStandardized]
& [oprZIP4 :: Address.Zip4 -AddressChange-PremiumAddressChange]
& [oprPRIADDRESS].REQUIRED
& [oprPRIADDRESS :: Address.Primary +UspsStandardized]
& [oprPRIADDRESS :: Address.Primary -AddressChange-PremiumAddressChange]
& [oprSECADDRESS].REQUIRED
& [oprSECADDRESS :: Address.Secondary +UspsStandardized]
& [oprSECADDRESS :: Address.Secondary -AddressChange-PremiumAddressChange]
& [oprURBNAME].REQUIRED
& [oprURBNAME :: Address.UrbanizationName +UspsStandardized]
& [oprURBNAME :: Address.UrbanizationName -AddressChange-PremiumAddressChange]
& [oprADDRESSFLAGS].REQUIRED
& [oprADDRESSFLAGS :: Address.Flags +UspsStandardized]
& [oprADDRESSFLAGS :: Address.Flags -AddressChange-PremiumAddressChange]
& [NCOABUSADDRESS].NA
& [NCOAPRINUMBER].NA
& [NCOAPREDIR].NA
& [NCOAPRINAME].NA
& [NCOASUFFIX].NA
& [NCOAPOSTDIR].NA
& [NCOASECUNITDES].NA
& [NCOASECNUMBER].NA
& [NCOACITY].NA
& [NCOASTATE].NA
& [NCOAZIP].NA
& [NCOAZIP4].NA
& [NCOAPRIADDRESS].NA
& [NCOASECADDRESS].NA
& [NCOAURBNAME].NA
& [NCOAADDRESSFLAGS].NA
& [CHANGEPLUSBUSADDRESS].NA
& [CHANGEPLUSPRINUMBER].NA
& [CHANGEPLUSPREDIR].NA
& [CHANGEPLUSPRINAME].NA
& [CHANGEPLUSSUFFIX].NA
& [CHANGEPLUSPOSTDIR].NA
& [CHANGEPLUSSECUNITDES].NA
& [CHANGEPLUSSECNUMBER].NA
& [CHANGEPLUSCITY].NA
& [CHANGEPLUSSTATE].NA
& [CHANGEPLUSZIP].NA
& [CHANGEPLUSZIP4].NA
& [CHANGEPLUSPRIADDRESS].NA
& [CHANGEPLUSSECADDRESS].NA
& [CHANGEPLUSURBNAME].NA

POSTCONDITION
{
  [PASBUSADDRESS :: Address.Business +CurrentAddressSelected]
  [PASPRINUMBER :: Address.PrimaryNumber +CurrentAddressSelected]
  [PASPREDIR :: Address.PreDirectional +CurrentAddressSelected]
  [PASPRINAME :: Address.PrimaryName +CurrentAddressSelected]

```
      [PASSUFFIX :: Address.Suffix +CurrentAddressSelected]
      [PASPOSTDIR :: Address.PostDirectional +CurrentAddressSelected]
      [PASSECUNITDES :: Address.SecondaryUnitDesignator +CurrentAddressSelected]
      [PASSECNUMBER :: Address.SecondaryNumber +CurrentAddressSelected]
      [PASCITYNAME :: Address.City +CurrentAddressSelected]
      [PASSTATEABREV :: Address.State +CurrentAddressSelected]
      [PASZIPCODE :: Address.Zip +CurrentAddressSelected]
      [PASADDONCODE :: Address.Zip4 +CurrentAddressSelected]
      [PASPRIADDRESS :: Address.Primary +CurrentAddressSelected]
      [PASSECADDRESS :: Address.Secondary +CurrentAddressSelected]
      [PASPRURBANIZATION :: Address.UrbanizationName +CurrentAddressSelected]
      [PASADDRESSFLAGS :: Address.Flags +CurrentAddressSelected]
   }
}

PRECONDITION
{
   [oprBUSADDRESS].REQUIRED
   & [oprBUSADDRESS :: Address.Business +UspsStandardized]
   & [oprBUSADDRESS :: Address.Business -AddressChange-PremiumAddressChange]
   & [oprPRINUMBER].REQUIRED
   & [oprPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
   & [oprPRINUMBER :: Address.PrimaryNumber -AddressChange-PremiumAddressChange]
   & [oprPREDIR].REQUIRED
   & [oprPREDIR :: Address.PreDirectional +UspsStandardized]
   & [oprPREDIR :: Address.PreDirectional -AddressChange-PremiumAddressChange]
   & [oprPRINAME].REQUIRED
   & [oprPRINAME :: Address.PrimaryName +UspsStandardized]
   & [oprPRINAME :: Address.PrimaryName -AddressChange-PremiumAddressChange]
   & [oprSUFFIX].REQUIRED
   & [oprSUFFIX :: Address.Suffix +UspsStandardized]
   & [oprSUFFIX :: Address.Suffix -AddressChange-PremiumAddressChange]
   & [oprPOSTDIR].REQUIRED
   & [oprPOSTDIR :: Address.PostDirectional +UspsStandardized]
   & [oprPOSTDIR :: Address.PostDirectional -AddressChange-PremiumAddressChange]
   & [oprSECUNITDES].REQUIRED
   & [oprSECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
   & [oprSECUNITDES :: Address.SecondaryUnitDesignator -AddressChange-PremiumAddressChange]
   & [oprSECNUMBER].REQUIRED
   & [oprSECNUMBER :: Address.SecondaryNumber +UspsStandardized]
   & [oprSECNUMBER :: Address.SecondaryNumber -AddressChange-PremiumAddressChange]
   & [oprCITY].REQUIRED
   & [oprCITY :: Address.City +UspsStandardized]
   & [oprCITY :: Address.City -AddressChange-PremiumAddressChange]
   & [oprSTATE].REQUIRED
   & [oprSTATE :: Address.State +UspsStandardized]
   & [oprSTATE :: Address.State -AddressChange-PremiumAddressChange]
   & [oprZIP].REQUIRED
   & [oprZIP :: Address.Zip +UspsStandardized]
   & [oprZIP :: Address.Zip -AddressChange-PremiumAddressChange]
   & [oprZIP4].REQUIRED
   & [oprZIP4 :: Address.Zip4 +UspsStandardized]
   & [oprZIP4 :: Address.Zip4 -AddressChange-PremiumAddressChange]
   & [oprPRIADDRESS].REQUIRED
   & [oprPRIADDRESS :: Address.Primary +UspsStandardized]
   & [oprPRIADDRESS :: Address.Primary -AddressChange-PremiumAddressChange]
```

& [oprSECADDRESS].REQUIRED
& [oprSECADDRESS :: Address.Secondary +UspsStandardized]
& [oprSECADDRESS :: Address.Secondary -AddressChange-PremiumAddressChange]
& [oprURBNAME].REQUIRED
& [oprURBNAME :: Address.UrbanizationName +UspsStandardized]
& [oprURBNAME :: Address.UrbanizationName -AddressChange-PremiumAddressChange]
& [oprADDRESSFLAGS].REQUIRED
& [oprADDRESSFLAGS :: Address.Flags +UspsStandardized]
& [oprADDRESSFLAGS :: Address.Flags -AddressChange-PremiumAddressChange]
& [NCOABUSADDRESS].REQUIRED
& [NCOABUSADDRESS :: Address.Business +AddressChange]
& [NCOAPRINUMBER].REQUIRED
& [NCOAPRINUMBER :: Address.PrimaryNumber +AddressChange]
& [NCOAPREDIR].REQUIRED
& [NCOAPREDIR :: Address.PreDirectional +AddressChange]
& [NCOAPRINAME].REQUIRED
& [NCOAPRINAME :: Address.PrimaryName +AddressChange]
& [NCOASUFFIX].REQUIRED
& [NCOASUFFIX :: Address.Suffix +AddressChange]
& [NCOAPOSTDIR].REQUIRED
& [NCOAPOSTDIR :: Address.PostDirectional +AddressChange]
& [NCOASECUNITDES].REQUIRED
& [NCOASECUNITDES :: Address.SecondaryUnitDesignator +AddressChange]
& [NCOASECNUMBER].REQUIRED
& [NCOASECNUMBER :: Address.SecondaryNumber +AddressChange]
& [NCOACITY].REQUIRED
& [NCOACITY :: Address.City +AddressChange]
& [NCOASTATE].REQUIRED
& [NCOASTATE :: Address.State +AddressChange]
& [NCOAZIP].REQUIRED
& [NCOAZIP :: Address.Zip +AddressChange]
& [NCOAZIP4].REQUIRED
& [NCOAZIP4 :: Address.Zip4 +AddressChange]
& [NCOAPRIADDRESS].REQUIRED
& [NCOAPRIADDRESS :: Address.Primary +AddressChange]
& [NCOASECADDRESS].REQUIRED
& [NCOASECADDRESS :: Address.Secondary +AddressChange]
& [NCOAURBNAME].REQUIRED
& [NCOAURBNAME :: Address.UrbanizationName +AddressChange]
& [NCOAADDRESSFLAGS].REQUIRED
& [NCOAADDRESSFLAGS :: Address.Flags +AddressChange]
& [CHANGEPLUSBUSADDRESS].NA
& [CHANGEPLUSPRINUMBER].NA
& [CHANGEPLUSPREDIR].NA
& [CHANGEPLUSPRINAME].NA
& [CHANGEPLUSSUFFIX].NA
& [CHANGEPLUSPOSTDIR].NA
& [CHANGEPLUSSECUNITDES].NA
& [CHANGEPLUSSECNUMBER].NA
& [CHANGEPLUSCITY].NA
& [CHANGEPLUSSTATE].NA
& [CHANGEPLUSZIP].NA
& [CHANGEPLUSZIP4].NA
& [CHANGEPLUSPRIADDRESS].NA
& [CHANGEPLUSSECADDRESS].NA
& [CHANGEPLUSURBNAME].NA

```
  POSTCONDITION
  {
    [PASBUSADDRESS :: Address.Business +CurrentAddressSelected]
    [PASPRINUMBER :: Address.PrimaryNumber +CurrentAddressSelected]
    [PASPREDIR :: Address.PreDirectional +CurrentAddressSelected]
    [PASPRINAME :: Address.PrimaryName +CurrentAddressSelected]
    [PASSUFFIX :: Address.Suffix +CurrentAddressSelected]
    [PASPOSTDIR :: Address.PostDirectional +CurrentAddressSelected]
    [PASSECUNITDES :: Address.SecondaryUnitDesignator +CurrentAddressSelected]
    [PASSECNUMBER :: Address.SecondaryNumber +CurrentAddressSelected]
    [PASCITYNAME :: Address.City +CurrentAddressSelected]
    [PASSTATEABREV :: Address.State +CurrentAddressSelected]
    [PASZIPCODE :: Address.Zip +CurrentAddressSelected]
    [PASADDONCODE :: Address.Zip4 +CurrentAddressSelected]
    [PASPRIADDRESS :: Address.Primary +CurrentAddressSelected]
    [PASSECADDRESS :: Address.Secondary +CurrentAddressSelected]
    [PASPRURBANIZATION :: Address.UrbanizationName +CurrentAddressSelected]
    [PASADDRESSFLAGS :: Address.Flags +CurrentAddressSelected]
  }
}

PRECONDITION
{
  [oprBUSADDRESS].REQUIRED
  & [oprBUSADDRESS :: Address.Business +UspsStandardized]
  & [oprBUSADDRESS :: Address.Business -AddressChange-PremiumAddressChange]
  & [oprPRINUMBER].REQUIRED
  & [oprPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
  & [oprPRINUMBER :: Address.PrimaryNumber -AddressChange-PremiumAddressChange]
  & [oprPREDIR].REQUIRED
  & [oprPREDIR :: Address.PreDirectional +UspsStandardized]
  & [oprPREDIR :: Address.PreDirectional -AddressChange-PremiumAddressChange]
  & [oprPRINAME].REQUIRED
  & [oprPRINAME :: Address.PrimaryName +UspsStandardized]
  & [oprPRINAME :: Address.PrimaryName -AddressChange-PremiumAddressChange]
  & [oprSUFFIX].REQUIRED
  & [oprSUFFIX :: Address.Suffix +UspsStandardized]
  & [oprSUFFIX :: Address.Suffix -AddressChange-PremiumAddressChange]
  & [oprPOSTDIR].REQUIRED
  & [oprPOSTDIR :: Address.PostDirectional +UspsStandardized]
  & [oprPOSTDIR :: Address.PostDirectional -AddressChange-PremiumAddressChange]
  & [oprSECUNITDES].REQUIRED
  & [oprSECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
  & [oprSECUNITDES :: Address.SecondaryUnitDesignator -AddressChange-PremiumAddressChange]
  & [oprSECNUMBER].REQUIRED
  & [oprSECNUMBER :: Address.SecondaryNumber +UspsStandardized]
  & [oprSECNUMBER :: Address.SecondaryNumber -AddressChange-PremiumAddressChange]
  & [oprCITY].REQUIRED
  & [oprCITY :: Address.City +UspsStandardized]
  & [oprCITY :: Address.City -AddressChange-PremiumAddressChange]
  & [oprSTATE].REQUIRED
  & [oprSTATE :: Address.State +UspsStandardized]
  & [oprSTATE :: Address.State -AddressChange-PremiumAddressChange]
  & [oprZIP].REQUIRED
  & [oprZIP :: Address.Zip +UspsStandardized]
```

& [oprZIP :: Address.Zip -AddressChange-PremiumAddressChange]
& [oprZIP4].REQUIRED
& [oprZIP4 :: Address.Zip4 +UspsStandardized]
& [oprZIP4 :: Address.Zip4 -AddressChange-PremiumAddressChange]
& [oprPRIADDRESS].REQUIRED
& [oprPRIADDRESS :: Address.Primary +UspsStandardized]
& [oprPRIADDRESS :: Address.Primary -AddressChange-PremiumAddressChange]
& [oprSECADDRESS].REQUIRED
& [oprSECADDRESS :: Address.Secondary +UspsStandardized]
& [oprSECADDRESS :: Address.Secondary -AddressChange-PremiumAddressChange]
& [oprURBNAME].REQUIRED
& [oprURBNAME :: Address.UrbanizationName +UspsStandardized]
& [oprURBNAME :: Address.UrbanizationName -AddressChange-PremiumAddressChange]
& [oprADDRESSFLAGS].REQUIRED
& [oprADDRESSFLAGS :: Address.Flags +UspsStandardized]
& [oprADDRESSFLAGS :: Address.Flags -AddressChange-PremiumAddressChange]
& [NCOABUSADDRESS].REQUIRED
& [NCOABUSADDRESS :: Address.Business +AddressChange]
& [NCOAPRINUMBER].REQUIRED
& [NCOAPRINUMBER :: Address.PrimaryNumber +AddressChange]
& [NCOAPREDIR].REQUIRED
& [NCOAPREDIR :: Address.PreDirectional +AddressChange]
& [NCOAPRINAME].REQUIRED
& [NCOAPRINAME :: Address.PrimaryName +AddressChange]
& [NCOASUFFIX].REQUIRED
& [NCOASUFFIX :: Address.Suffix +AddressChange]
& [NCOAPOSTDIR].REQUIRED
& [NCOAPOSTDIR :: Address.PostDirectional +AddressChange]
& [NCOASECUNITDES].REQUIRED
& [NCOASECUNITDES :: Address.SecondaryUnitDesignator +AddressChange]
& [NCOASECNUMBER].REQUIRED
& [NCOASECNUMBER :: Address.SecondaryNumber +AddressChange]
& [NCOACITY].REQUIRED
& [NCOACITY :: Address.City +AddressChange]
& [NCOASTATE].REQUIRED
& [NCOASTATE :: Address.State +AddressChange]
& [NCOAZIP].REQUIRED
& [NCOAZIP :: Address.Zip +AddressChange]
& [NCOAZIP4].REQUIRED
& [NCOAZIP4 :: Address.Zip4 +AddressChange]
& [NCOAPRIADDRESS].REQUIRED
& [NCOAPRIADDRESS :: Address.Primary +AddressChange]
& [NCOASECADDRESS].REQUIRED
& [NCOASECADDRESS :: Address.Secondary +AddressChange]
& [NCOAURBNAME].REQUIRED
& [NCOAURBNAME :: Address.UrbanizationName +AddressChange]
& [NCOAADDRESSFLAGS].REQUIRED
& [NCOAADDRESSFLAGS :: Address.Flags +AddressChange]
& [CHANGEPLUSBUSADDRESS].REQUIRED
& [CHANGEPLUSBUSADDRESS :: Address.Business +PremiumAddressChange]
& [CHANGEPLUSPRINUMBER].REQUIRED
& [CHANGEPLUSPRINUMBER :: Address.PrimaryNumber +PremiumAddressChange]
& [CHANGEPLUSPREDIR].REQUIRED
& [CHANGEPLUSPREDIR :: Address.PreDirectional +PremiumAddressChange]
& [CHANGEPLUSPRINAME].REQUIRED
& [CHANGEPLUSPRINAME :: Address.PrimaryName +PremiumAddressChange]

```
        & [CHANGEPLUSSUFFIX].REQUIRED
        & [CHANGEPLUSSUFFIX :: Address.Suffix +PremiumAddressChange]
        & [CHANGEPLUSPOSTDIR].REQUIRED
        & [CHANGEPLUSPOSTDIR :: Address.PostDirectional +PremiumAddressChange]
        & [CHANGEPLUSSECUNITDES].REQUIRED
        & [CHANGEPLUSSECUNITDES :: Address.SecondaryUnitDesignator +PremiumAddressChange]
        & [CHANGEPLUSSECNUMBER].REQUIRED
        & [CHANGEPLUSSECNUMBER :: Address.SecondaryNumber +PremiumAddressChange]
        & [CHANGEPLUSCITY].REQUIRED
        & [CHANGEPLUSCITY :: Address.City +PremiumAddressChange]
        & [CHANGEPLUSSTATE].REQUIRED
        & [CHANGEPLUSSTATE :: Address.State +PremiumAddressChange]
        & [CHANGEPLUSZIP].REQUIRED
        & [CHANGEPLUSZIP :: Address.Zip +PremiumAddressChange]
        & [CHANGEPLUSZIP4].REQUIRED
        & [CHANGEPLUSZIP4 :: Address.Zip4 +PremiumAddressChange]
        & [CHANGEPLUSPRIADDRESS].REQUIRED
        & [CHANGEPLUSPRIADDRESS :: Address.Primary +PremiumAddressChange]
        & [CHANGEPLUSSECADDRESS].REQUIRED
        & [CHANGEPLUSSECADDRESS :: Address.Secondary +PremiumAddressChange]
        & [CHANGEPLUSURBNAME].REQUIRED
        & [CHANGEPLUSURBNAME :: Address.UrbanizationName +PremiumAddressChange]

        POSTCONDITION
        {
          [PASBUSADDRESS :: Address.Business +CurrentAddressSelected]
          [PASPRINUMBER :: Address.PrimaryNumber +CurrentAddressSelected]
          [PASPREDIR :: Address.PreDirectional +CurrentAddressSelected]
          [PASPRINAME :: Address.PrimaryName +CurrentAddressSelected]
          [PASSUFFIX :: Address.Suffix +CurrentAddressSelected]
          [PASPOSTDIR :: Address.PostDirectional +CurrentAddressSelected]
          [PASSECUNITDES :: Address.SecondaryUnitDesignator +CurrentAddressSelected]
          [PASSECNUMBER :: Address.SecondaryNumber +CurrentAddressSelected]
          [PASCITYNAME :: Address.City +CurrentAddressSelected]
          [PASSTATEABREV :: Address.State +CurrentAddressSelected]
          [PASZIPCODE :: Address.Zip +CurrentAddressSelected]
          [PASADDONCODE :: Address.Zip4 +CurrentAddressSelected]
          [PASPRIADDRESS :: Address.Primary +CurrentAddressSelected]
          [PASSECADDRESS :: Address.Secondary +CurrentAddressSelected]
          [PASPRURBANIZATION :: Address.UrbanizationName +CurrentAddressSelected]
          [PASADDRESSFLAGS :: Address.Flags +CurrentAddressSelected]
        }
      }
    }

    OPTIONS
    {
    }
}
```

A.4 ContactLink

```
ContactLink
{

   INPUT FIELDS
   {
      FIRSTNAME
      MIDDLEINITIAL
      LASTNAME
      CONSBUSINDICATOR
      YEAROFBIRTH
      SUMOFSSN4
      BUSADDRESS
      PRINUMBER
      PREDIR
      PRINAME
      SUFFIX
      POSTDIR
      SECUNITDES
      SECNUMBER
      CITY
      STATE
      ZIP
      ZIP4
      PRIADDRESS
      SECADDRESS
      URBNAME
   }

   OUTPUT FIELDS
   {
      ADDRESSLINK
   }

   PRECONDITIONS
   {

      PRECONDITION
      {
         [BUSADDRESS].REQUIRED
         & [BUSADDRESS :: Address.Business +CurrentAddressSelected]
         & [PRINUMBER].REQUIRED
         & [PRINUMBER :: Address.PrimaryNumber +CurrentAddressSelected]
         & [PREDIR].REQUIRED
         & [PREDIR :: Address.PreDirectional +CurrentAddressSelected]
         & [PRINAME].REQUIRED
         & [PRINAME :: Address.PrimaryName +CurrentAddressSelected]
         & [SUFFIX].REQUIRED
         & [SUFFIX :: Address.Suffix +CurrentAddressSelected]
         & [POSTDIR].REQUIRED
         & [POSTDIR :: Address.PostDirectional +CurrentAddressSelected]
         & [SECUNITDES].REQUIRED
         & [SECUNITDES :: Address.SecondaryUnitDesignator +CurrentAddressSelected]
         & [SECNUMBER].REQUIRED
```

```
      & [SECNUMBER :: Address.SecondaryNumber +CurrentAddressSelected]
      & [CITY].REQUIRED
      & [CITY :: Address.City +CurrentAddressSelected]
      & [STATE].REQUIRED
      & [STATE :: Address.State +CurrentAddressSelected]
      & [ZIP].REQUIRED
      & [ZIP :: Address.Zip +CurrentAddressSelected]
      & [ZIP4].REQUIRED
      & [ZIP4 :: Address.Zip4 +CurrentAddressSelected]
      & [PRIADDRESS].REQUIRED
      & [PRIADDRESS :: Address.Primary +CurrentAddressSelected]
      & [SECADDRESS].REQUIRED
      & [SECADDRESS :: Address.Secondary +CurrentAddressSelected]
      & [URBNAME].REQUIRED
      & [URBNAME :: Address.UrbanizationName +CurrentAddressSelected]

      POSTCONDITION
      {
        [ADDRESSLINK :: Link.Address +AddressContactLinked]
      }
    }
}

OPTIONS
{

   AppendConsumerContactLinks
   {

      OPTION SETTINGS
      {

        N
        {

          PRECONDITIONS
          {
          }
        }

        Y
        {

          PRECONDITIONS
          {

            PRECONDITION
            {
              NONE

              POSTCONDITION
              {
                [BUSADDRESS :: Address.Business +ConsumerContactLinked+AddressContactLinked]
                [PRINUMBER :: Address.PrimaryNumber +ConsumerContactLinked+AddressContactLinked]
                [PREDIR :: Address.PreDirectional +ConsumerContactLinked+AddressContactLinked]
                [PRINAME :: Address.PrimaryName +ConsumerContactLinked+AddressContactLinked]
```

```
                    [SUFFIX :: Address.Suffix +ConsumerContactLinked+AddressContactLinked]
                    [POSTDIR :: Address.PostDirectional +ConsumerContactLinked+AddressContactLinked]
                    [SECUNITDES :: Address.SecondaryUnitDesignator
+ConsumerContactLinked+AddressContactLinked]
                    [SECNUMBER :: Address.SecondaryNumber
+ConsumerContactLinked+AddressContactLinked]
                    [CITY :: Address.City +ConsumerContactLinked+AddressContactLinked]
                    [STATE :: Address.State +ConsumerContactLinked+AddressContactLinked]
                    [ZIP :: Address.Zip +ConsumerContactLinked+AddressContactLinked]
                    [ZIP4 :: Address.Zip4 +ConsumerContactLinked+AddressContactLinked]
                    [PRIADDRESS :: Address.Primary +ConsumerContactLinked+AddressContactLinked]
                    [SECADDRESS :: Address.Secondary +ConsumerContactLinked+AddressContactLinked]
                    [URBNAME :: Address.UrbanizationName +ConsumerContactLinked+AddressContactLinked]
                }
              }
            }
          }
        }
      }

      AppendConsumerContactLinks
      {

        OPTION SETTINGS
        {

          N
          {

            PRECONDITIONS
            {
            }
          }

          Y
          {

            PRECONDITIONS
            {

              PRECONDITION
              {
                NONE

                POSTCONDITION
                {
                  [BUSADDRESS :: Address.Business +BusinessContactLinked+AddressContactLinked]
                  [PRINUMBER :: Address.PrimaryNumber +BusinessContactLinked+AddressContactLinked]
                  [PREDIR :: Address.PreDirectional +BusinessContactLinked+AddressContactLinked]
                  [PRINAME :: Address.PrimaryName +BusinessContactLinked+AddressContactLinked]
                  [SUFFIX :: Address.Suffix +BusinessContactLinked+AddressContactLinked]
                  [POSTDIR :: Address.PostDirectional +BusinessContactLinked+AddressContactLinked]
                  [SECUNITDES :: Address.SecondaryUnitDesignator
+BusinessContactLinked+AddressContactLinked]
                  [SECNUMBER :: Address.SecondaryNumber
+BusinessContactLinked+AddressContactLinked]
```

```
                [CITY :: Address.City +BusinessContactLinked+AddressContactLinked]
                [STATE :: Address.State +BusinessContactLinked+AddressContactLinked]
                [ZIP :: Address.Zip +BusinessContactLinked+AddressContactLinked]
                [ZIP4 :: Address.Zip4 +BusinessContactLinked+AddressContactLinked]
                [PRIADDRESS :: Address.Primary +BusinessContactLinked+AddressContactLinked]
                [SECADDRESS :: Address.Secondary +BusinessContactLinked+AddressContactLinked]
                [URBNAME :: Address.UrbanizationName +BusinessContactLinked+AddressContactLinked]
            }
        }
    }
  }
 }
}
```

A.5 IndustryCode

```
IndustryCode
{

   INPUT FIELDS
   {
      PROFESSIONAL_TITLE
      LAST_NAME_SUFFIX
      BUSINESS_NAME
      EXISTING_SIM_TITLE_CODE
      EXISTING_SIM_FUNCTION1_CODE
   }

   OUTPUT FIELDS
   {
      SIMNEWTITLE
      SIMTITLECODE
      SIMFUNCTION1CODE
   }

   PRECONDITIONS
   {

      PRECONDITION
      {
         [PROFESSIONAL_TITLE].REQUIRED
         & [PROFESSIONAL_TITLE :: Name.ProfessionalTitle ]

         POSTCONDITION
         {
            [SIMNEWTITLE :: Name.IndustryTitle ]
            [SIMTITLECODE :: Name.IndustryTitleCode ]
            [SIMFUNCTION1CODE :: Name.IndustryFunctionCode ]
            [PROFESSIONAL_TITLE :: Name.ProfessionalTitle +IndustryCoded]
         }
      }
   }

   OPTIONS
   {

      SUFFIX TITLE
      {

         OPTION SETTINGS
         {

            N
            {

               PRECONDITIONS
               {
               }
            }
```

```
        Y
        {

          PRECONDITIONS
          {

            PRECONDITION
            {
              [LAST_NAME_SUFFIX].REQUIRED
              & [LAST_NAME_SUFFIX :: Name.Suffix ]

              POSTCONDITION
              {
              }
            }
          }
        }
      }

      BUSINESS NAME
      {

        OPTION SETTINGS
        {

          N
          {

            PRECONDITIONS
            {
            }
          }

          Y
          {

            PRECONDITIONS
            {

              PRECONDITION
              {
                [BUSINESS_NAME].REQUIRED
                & [BUSINESS_NAME :: Name.Business ]

                POSTCONDITION
                {
                }
              }
            }
          }
        }
      }
    }
  }
```

A.6 NameEditCheck

```
NameEditCheck
{

   INPUT FIELDS
   {
      GX_FNAME
      GX_MNAME
      GX_LNAME
      GX_FULLNAME
   }

   OUTPUT FIELDS
   {
      GX_SUGFNAME
      GX_SUGMNAME
      GX_SUGLNAME
      GX_SUGFULLNAME
   }

   PRECONDITIONS
   {

      PRECONDITION
      {
         [GX_FNAME].REQUIRED
         & [GX_FNAME :: Name.First ]
         & [GX_LNAME].REQUIRED
         & [GX_LNAME :: Name.Last ]
         & [GX_MNAME].NA
         & [GX_FULLNAME].NA

         POSTCONDITION
         {
            [GX_SUGFNAME :: Name.First +NameEditChecked]
            [GX_SUGLNAME :: Name.Last +NameEditChecked]
         }
      }

      PRECONDITION
      {
         [GX_FNAME].REQUIRED
         & [GX_FNAME :: Name.First ]
         & [GX_LNAME].REQUIRED
         & [GX_LNAME :: Name.Last ]
         & [GX_MNAME].REQUIRED
         & [GX_MNAME :: Name.Middle ]
         & [GX_FULLNAME].NA

         POSTCONDITION
         {
            [GX_SUGFNAME :: Name.First +NameEditChecked]
            [GX_SUGMNAME :: Name.Middle +NameEditChecked]
            [GX_SUGLNAME :: Name.Last +NameEditChecked]
```

```
        }
      }

      PRECONDITION
      {
        [GX_FNAME].NA
        & [GX_LNAME].NA
        & [GX_MNAME].NA
        & [GX_FULLNAME].REQUIRED
        & [GX_FULLNAME :: Name.Full ]

        POSTCONDITION
        {
          [GX_SUGFULLNAME :: Name.Full +NameEditChecked]
        }
      }
    }

    OPTIONS
    {
    }
}
```

A.7 Parser

```
Parser
{

   INPUT FIELDS
   {
      UNPARSEDNAME
      PARSEDFIRSTNAME
      PARSEDMIDDLENAME
      PARSEDLASTNAME
      PARSEDLASTNAMESUFFIX
      PROFESSIONALTITLE
      PRIMARYBUSINESSNAME
      SECONDARYBUSINESSNAME
      ADDRESSLINE1
      ADDRESSLINE2
      UNPARSEDCITYSTATEZIP
      PARSEDCITY
      PARSEDSTATE
      PARSEDZIP
      PARSEDZIP4
      PARSEDURBNAME
   }

   OUTPUT FIELDS
   {
      CVFIRSTNAME1
      CVMIDDLENAME1
      CVLASTNAME1
      CVPROFTITLE1
      CVGENDERCODE1
      CVPRIMARYBUSINESS
      CVSECONDARYBUSINESS
      CVADDRESSTYPE
      CVPRIMARYADDRESS
      CVSECONDARYADDRESS
      CVAAPRINUMBER
      CVAAPREDIR1
      CVAAPRINAME
      CVAASUFFIX1
      CVAAPOSTDIR
      CVAASECUNITDES
      CVAASECNUMBER
      CVAABUSNAME
      CVAAOUTPRURB
      CVAACITYNAME
      CVAASTATEABBREV
      CVAAZIPCODE
      CVAAADDONCODE
   }

   PRECONDITIONS
   {
```

```
PRECONDITION
{
   [UNPARSEDNAME].REQUIRED
   & [UNPARSEDNAME :: Name.Full ]
   & [PARSEDFIRSTNAME].NA
   & [PARSEDMIDDLENAME].NA
   & [PARSEDLASTNAME].NA
   & [PARSEDLASTNAMESUFFIX].NA
   & [ADDRESSLINE1].REQUIRED
   & [ADDRESSLINE1 :: Address.Primary ]
   & [UNPARSEDCITYSTATEZIP].REQUIRED
   & [UNPARSEDCITYSTATEZIP :: Address.CityStateZip ]
   & [PARSEDCITY].NA
   & [PARSEDSTATE].NA
   & [PARSEDZIP].NA
   & [PARSEDZIP4].NA
   & [PARSEDURBNAME].NA

   POSTCONDITION
   {
      [CVFIRSTNAME1 :: Name.First ]
      [CVMIDDLENAME1 :: Name.Middle ]
      [CVLASTNAME1 :: Name.Last ]
      [CVGENDERCODE1 :: Name.Gender ]
      [CVADDRESSTYPE :: Address.Type ]
      [CVPRIMARYADDRESS :: Address.Primary ]
      [CVSECONDARYADDRESS :: Address.Secondary ]
      [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
      [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
      [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
      [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
      [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
      [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
      [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
      [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
      [CVAACITYNAME :: Address.City +UspsStandardized]
      [CVAASTATEABBREV :: Address.State +UspsStandardized]
      [CVAAZIPCODE :: Address.Zip +UspsStandardized]
      [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
      [CVPROFTITLE1 :: Name.ProfessionalTitle ]
      [CVPRIMARYBUSINESS :: Name.Business ]
      [CVSECONDARYBUSINESS :: Name.Business ]
      [CVAABUSNAME :: Name.Business +UspsStandardized]
   }
}

PRECONDITION
{
   [UNPARSEDNAME].REQUIRED
   & [UNPARSEDNAME :: Name.Full ]
   & [PARSEDFIRSTNAME].NA
   & [PARSEDMIDDLENAME].NA
   & [PARSEDLASTNAME].NA
   & [PARSEDLASTNAMESUFFIX].NA
   & [ADDRESSLINE1].REQUIRED
   & [ADDRESSLINE1 :: Address.Primary ]
```

```
   & [UNPARSEDCITYSTATEZIP].NA
   & [PARSEDCITY].REQUIRED
   & [PARSEDCITY :: Address.City ]
   & [PARSEDSTATE].REQUIRED
   & [PARSEDSTATE :: Address.State ]
   & [PARSEDZIP].REQUIRED
   & [PARSEDZIP :: Address.Zip ]

   POSTCONDITION
   {
      [CVFIRSTNAME1 :: Name.First ]
      [CVMIDDLENAME1 :: Name.Middle ]
      [CVLASTNAME1 :: Name.Last ]
      [CVGENDERCODE1 :: Name.Gender ]
      [CVADDRESSTYPE :: Address.Type ]
      [CVPRIMARYADDRESS :: Address.Primary ]
      [CVSECONDARYADDRESS :: Address.Secondary ]
      [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
      [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
      [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
      [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
      [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
      [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
      [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
      [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
      [CVAACITYNAME :: Address.City +UspsStandardized]
      [CVAASTATEABBREV :: Address.State +UspsStandardized]
      [CVAAZIPCODE :: Address.Zip +UspsStandardized]
      [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
      [CVPROFTITLE1 :: Name.ProfessionalTitle ]
      [CVPRIMARYBUSINESS :: Name.Business ]
      [CVSECONDARYBUSINESS :: Name.Business ]
      [CVAABUSNAME :: Name.Business +UspsStandardized]
   }
}

PRECONDITION
{
   [UNPARSEDNAME].NA
   & [PARSEDFIRSTNAME].REQUIRED
   & [PARSEDFIRSTNAME :: Name.First ]
   & [PARSEDLASTNAME].REQUIRED
   & [PARSEDLASTNAME :: Name.Last ]
   & [ADDRESSLINE1].REQUIRED
   & [ADDRESSLINE1 :: Address.Primary ]
   & [PROFESSIONALTITLE].NA
   & [PRIMARYBUSINESSNAME].NA
   & [SECONDARYBUSINESSNAME].NA
   & [UNPARSEDCITYSTATEZIP].REQUIRED
   & [UNPARSEDCITYSTATEZIP :: Address.CityStateZip ]
   & [PARSEDCITY].NA
   & [PARSEDSTATE].NA
   & [PARSEDZIP].NA
   & [PARSEDZIP4].NA
   & [PARSEDURBNAME].NA
```

```
  POSTCONDITION
  {
    [CVFIRSTNAME1 :: Name.First ]
    [CVMIDDLENAME1 :: Name.Middle ]
    [CVLASTNAME1 :: Name.Last ]
    [CVGENDERCODE1 :: Name.Gender ]
    [CVADDRESSTYPE :: Address.Type ]
    [CVPRIMARYADDRESS :: Address.Primary ]
    [CVSECONDARYADDRESS :: Address.Secondary ]
    [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
    [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
    [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
    [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
    [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
    [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
    [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
    [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
    [CVAACITYNAME :: Address.City +UspsStandardized]
    [CVAASTATEABBREV :: Address.State +UspsStandardized]
    [CVAAZIPCODE :: Address.Zip +UspsStandardized]
    [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
  }
}

PRECONDITION
{
  [UNPARSEDNAME].NA
  & [PARSEDFIRSTNAME].REQUIRED
  & [PARSEDFIRSTNAME :: Name.First ]
  & [PARSEDLASTNAME].REQUIRED
  & [PARSEDLASTNAME :: Name.Last ]
  & [ADDRESSLINE1].REQUIRED
  & [ADDRESSLINE1 :: Address.Primary ]
  & [PROFESSIONALTITLE].REQUIRED
  & [PROFESSIONALTITLE :: Name.ProfessionalTitle ]
  & [PRIMARYBUSINESSNAME].NA
  & [SECONDARYBUSINESSNAME].NA
  & [UNPARSEDCITYSTATEZIP].REQUIRED
  & [UNPARSEDCITYSTATEZIP :: Address.CityStateZip ]
  & [PARSEDCITY].NA
  & [PARSEDSTATE].NA
  & [PARSEDZIP].NA
  & [PARSEDZIP4].NA
  & [PARSEDURBNAME].NA

  POSTCONDITION
  {
    [CVFIRSTNAME1 :: Name.First ]
    [CVMIDDLENAME1 :: Name.Middle ]
    [CVLASTNAME1 :: Name.Last ]
    [CVGENDERCODE1 :: Name.Gender ]
    [CVADDRESSTYPE :: Address.Type ]
    [CVPRIMARYADDRESS :: Address.Primary ]
    [CVSECONDARYADDRESS :: Address.Secondary ]
    [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
    [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
```

```
      [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
      [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
      [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
      [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
      [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
      [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
      [CVAACITYNAME :: Address.City +UspsStandardized]
      [CVAASTATEABBREV :: Address.State +UspsStandardized]
      [CVAAZIPCODE :: Address.Zip +UspsStandardized]
      [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
      [CVPROFTITLE1 :: Name.ProfessionalTitle ]
   }
}

PRECONDITION
{
   [UNPARSEDNAME].NA
   & [PARSEDFIRSTNAME].REQUIRED
   & [PARSEDFIRSTNAME :: Name.First ]
   & [PARSEDLASTNAME].REQUIRED
   & [PARSEDLASTNAME :: Name.Last ]
   & [ADDRESSLINE1].REQUIRED
   & [ADDRESSLINE1 :: Address.Primary ]
   & [PROFESSIONALTITLE].NA
   & [PRIMARYBUSINESSNAME].REQUIRED
   & [PRIMARYBUSINESSNAME :: Name.Business ]
   & [SECONDARYBUSINESSNAME].NA
   & [UNPARSEDCITYSTATEZIP].REQUIRED
   & [UNPARSEDCITYSTATEZIP :: Address.CityStateZip ]
   & [PARSEDCITY].NA
   & [PARSEDSTATE].NA
   & [PARSEDZIP].NA
   & [PARSEDZIP4].NA
   & [PARSEDURBNAME].NA

   POSTCONDITION
   {
      [CVFIRSTNAME1 :: Name.First ]
      [CVMIDDLENAME1 :: Name.Middle ]
      [CVLASTNAME1 :: Name.Last ]
      [CVGENDERCODE1 :: Name.Gender ]
      [CVADDRESSTYPE :: Address.Type ]
      [CVPRIMARYADDRESS :: Address.Primary ]
      [CVSECONDARYADDRESS :: Address.Secondary ]
      [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
      [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
      [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
      [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
      [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
      [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
      [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
      [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
      [CVAACITYNAME :: Address.City +UspsStandardized]
      [CVAASTATEABBREV :: Address.State +UspsStandardized]
      [CVAAZIPCODE :: Address.Zip +UspsStandardized]
      [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
```

```
      [CVPRIMARYBUSINESS :: Name.Business ]
      [CVAABUSNAME :: Name.Business +UspsStandardized]
   }
}

PRECONDITION
{
   [UNPARSEDNAME].NA
   & [PARSEDFIRSTNAME].REQUIRED
   & [PARSEDFIRSTNAME :: Name.First ]
   & [PARSEDLASTNAME].REQUIRED
   & [PARSEDLASTNAME :: Name.Last ]
   & [ADDRESSLINE1].REQUIRED
   & [ADDRESSLINE1 :: Address.Primary ]
   & [PROFESSIONALTITLE].NA
   & [PRIMARYBUSINESSNAME].NA
   & [SECONDARYBUSINESSNAME].REQUIRED
   & [SECONDARYBUSINESSNAME :: Name.Business ]
   & [UNPARSEDCITYSTATEZIP].REQUIRED
   & [UNPARSEDCITYSTATEZIP :: Address.CityStateZip ]
   & [PARSEDCITY].NA
   & [PARSEDSTATE].NA
   & [PARSEDZIP].NA
   & [PARSEDZIP4].NA
   & [PARSEDURBNAME].NA

   POSTCONDITION
   {
      [CVFIRSTNAME1 :: Name.First ]
      [CVMIDDLENAME1 :: Name.Middle ]
      [CVLASTNAME1 :: Name.Last ]
      [CVGENDERCODE1 :: Name.Gender ]
      [CVADDRESSTYPE :: Address.Type ]
      [CVPRIMARYADDRESS :: Address.Primary ]
      [CVSECONDARYADDRESS :: Address.Secondary ]
      [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
      [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
      [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
      [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
      [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
      [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
      [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
      [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
      [CVAACITYNAME :: Address.City +UspsStandardized]
      [CVAASTATEABBREV :: Address.State +UspsStandardized]
      [CVAAZIPCODE :: Address.Zip +UspsStandardized]
      [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
      [CVSECONDARYBUSINESS :: Name.Business ]
   }
}

PRECONDITION
{
   [UNPARSEDNAME].NA
   & [PARSEDFIRSTNAME].REQUIRED
   & [PARSEDFIRSTNAME :: Name.First ]
```

```
   & [PARSEDLASTNAME].REQUIRED
   & [PARSEDLASTNAME :: Name.Last ]
   & [ADDRESSLINE1].REQUIRED
   & [ADDRESSLINE1 :: Address.Primary ]
   & [PROFESSIONALTITLE].REQUIRED
   & [PROFESSIONALTITLE :: Name.ProfessionalTitle ]
   & [PRIMARYBUSINESSNAME].REQUIRED
   & [PRIMARYBUSINESSNAME :: Name.Business ]
   & [SECONDARYBUSINESSNAME].NA
   & [UNPARSEDCITYSTATEZIP].REQUIRED
   & [UNPARSEDCITYSTATEZIP :: Address.CityStateZip ]
   & [PARSEDCITY].NA
   & [PARSEDSTATE].NA
   & [PARSEDZIP].NA
   & [PARSEDZIP4].NA
   & [PARSEDURBNAME].NA

   POSTCONDITION
   {
      [CVFIRSTNAME1 :: Name.First ]
      [CVMIDDLENAME1 :: Name.Middle ]
      [CVLASTNAME1 :: Name.Last ]
      [CVGENDERCODE1 :: Name.Gender ]
      [CVADDRESSTYPE :: Address.Type ]
      [CVPRIMARYADDRESS :: Address.Primary ]
      [CVSECONDARYADDRESS :: Address.Secondary ]
      [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
      [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
      [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
      [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
      [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
      [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
      [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
      [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
      [CVAACITYNAME :: Address.City +UspsStandardized]
      [CVAASTATEABBREV :: Address.State +UspsStandardized]
      [CVAAZIPCODE :: Address.Zip +UspsStandardized]
      [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
      [CVPROFTITLE1 :: Name.ProfessionalTitle ]
      [CVPRIMARYBUSINESS :: Name.Business ]
      [CVAABUSNAME :: Name.Business +UspsStandardized]
   }
}

PRECONDITION
{
   [UNPARSEDNAME].NA
   & [PARSEDFIRSTNAME].REQUIRED
   & [PARSEDFIRSTNAME :: Name.First ]
   & [PARSEDLASTNAME].REQUIRED
   & [PARSEDLASTNAME :: Name.Last ]
   & [ADDRESSLINE1].REQUIRED
   & [ADDRESSLINE1 :: Address.Primary ]
   & [PROFESSIONALTITLE].REQUIRED
   & [PROFESSIONALTITLE :: Name.ProfessionalTitle ]
   & [PRIMARYBUSINESSNAME].NA
```

```
   & [SECONDARYBUSINESSNAME].REQUIRED
   & [SECONDARYBUSINESSNAME :: Name.Business ]
   & [UNPARSEDCITYSTATEZIP].REQUIRED
   & [UNPARSEDCITYSTATEZIP :: Address.CityStateZip ]
   & [PARSEDCITY].NA
   & [PARSEDSTATE].NA
   & [PARSEDZIP].NA
   & [PARSEDZIP4].NA
   & [PARSEDURBNAME].NA

   POSTCONDITION
   {
      [CVFIRSTNAME1 :: Name.First ]
      [CVMIDDLENAME1 :: Name.Middle ]
      [CVLASTNAME1 :: Name.Last ]
      [CVGENDERCODE1 :: Name.Gender ]
      [CVADDRESSTYPE :: Address.Type ]
      [CVPRIMARYADDRESS :: Address.Primary ]
      [CVSECONDARYADDRESS :: Address.Secondary ]
      [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
      [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
      [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
      [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
      [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
      [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
      [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
      [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
      [CVAACITYNAME :: Address.City +UspsStandardized]
      [CVAASTATEABBREV :: Address.State +UspsStandardized]
      [CVAAZIPCODE :: Address.Zip +UspsStandardized]
      [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
      [CVPROFTITLE1 :: Name.ProfessionalTitle ]
      [CVSECONDARYBUSINESS :: Name.Business ]
   }
}

PRECONDITION
{
   [UNPARSEDNAME].NA
   & [PARSEDFIRSTNAME].REQUIRED
   & [PARSEDFIRSTNAME :: Name.First ]
   & [PARSEDLASTNAME].REQUIRED
   & [PARSEDLASTNAME :: Name.Last ]
   & [ADDRESSLINE1].REQUIRED
   & [ADDRESSLINE1 :: Address.Primary ]
   & [PROFESSIONALTITLE].NA
   & [PRIMARYBUSINESSNAME].REQUIRED
   & [PRIMARYBUSINESSNAME :: Name.Business ]
   & [SECONDARYBUSINESSNAME].REQUIRED
   & [SECONDARYBUSINESSNAME :: Name.Business ]
   & [UNPARSEDCITYSTATEZIP].REQUIRED
   & [UNPARSEDCITYSTATEZIP :: Address.CityStateZip ]
   & [PARSEDCITY].NA
   & [PARSEDSTATE].NA
   & [PARSEDZIP].NA
   & [PARSEDZIP4].NA
```

```
   & [PARSEDURBNAME].NA

   POSTCONDITION
   {
      [CVFIRSTNAME1 :: Name.First ]
      [CVMIDDLENAME1 :: Name.Middle ]
      [CVLASTNAME1 :: Name.Last ]
      [CVGENDERCODE1 :: Name.Gender ]
      [CVADDRESSTYPE :: Address.Type ]
      [CVPRIMARYADDRESS :: Address.Primary ]
      [CVSECONDARYADDRESS :: Address.Secondary ]
      [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
      [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
      [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
      [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
      [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
      [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
      [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
      [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
      [CVAACITYNAME :: Address.City +UspsStandardized]
      [CVAASTATEABBREV :: Address.State +UspsStandardized]
      [CVAAZIPCODE :: Address.Zip +UspsStandardized]
      [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
      [CVPRIMARYBUSINESS :: Name.Business ]
      [CVSECONDARYBUSINESS :: Name.Business ]
      [CVAABUSNAME :: Name.Business +UspsStandardized]
   }
}

PRECONDITION
{
   [UNPARSEDNAME].NA
   & [PARSEDFIRSTNAME].REQUIRED
   & [PARSEDFIRSTNAME :: Name.First ]
   & [PARSEDLASTNAME].REQUIRED
   & [PARSEDLASTNAME :: Name.Last ]
   & [ADDRESSLINE1].REQUIRED
   & [ADDRESSLINE1 :: Address.Primary ]
   & [PROFESSIONALTITLE].REQUIRED
   & [PROFESSIONALTITLE :: Name.ProfessionalTitle ]
   & [PRIMARYBUSINESSNAME].REQUIRED
   & [PRIMARYBUSINESSNAME :: Name.Business ]
   & [SECONDARYBUSINESSNAME].REQUIRED
   & [SECONDARYBUSINESSNAME :: Name.Business ]
   & [UNPARSEDCITYSTATEZIP].REQUIRED
   & [UNPARSEDCITYSTATEZIP :: Address.CityStateZip ]
   & [PARSEDCITY].NA
   & [PARSEDSTATE].NA
   & [PARSEDZIP].NA
   & [PARSEDZIP4].NA
   & [PARSEDURBNAME].NA

   POSTCONDITION
   {
      [CVFIRSTNAME1 :: Name.First ]
      [CVMIDDLENAME1 :: Name.Middle ]
```

```
    [CVLASTNAME1 :: Name.Last ]
    [CVGENDERCODE1 :: Name.Gender ]
    [CVADDRESSTYPE :: Address.Type ]
    [CVPRIMARYADDRESS :: Address.Primary ]
    [CVSECONDARYADDRESS :: Address.Secondary ]
    [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
    [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
    [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
    [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
    [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
    [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
    [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
    [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
    [CVAACITYNAME :: Address.City +UspsStandardized]
    [CVAASTATEABBREV :: Address.State +UspsStandardized]
    [CVAAZIPCODE :: Address.Zip +UspsStandardized]
    [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
    [CVPROFTITLE1 :: Name.ProfessionalTitle ]
    [CVPRIMARYBUSINESS :: Name.Business ]
    [CVSECONDARYBUSINESS :: Name.Business ]
    [CVAABUSNAME :: Name.Business +UspsStandardized]
  }
}

PRECONDITION
{
  [UNPARSEDNAME].NA
  & [PARSEDFIRSTNAME].REQUIRED
  & [PARSEDFIRSTNAME :: Name.First ]
  & [PARSEDLASTNAME].REQUIRED
  & [PARSEDLASTNAME :: Name.Last ]
  & [ADDRESSLINE1].REQUIRED
  & [ADDRESSLINE1 :: Address.Primary ]
  & [PROFESSIONALTITLE].NA
  & [PRIMARYBUSINESSNAME].NA
  & [SECONDARYBUSINESSNAME].NA
  & [UNPARSEDCITYSTATEZIP].NA
  & [PARSEDCITY].REQUIRED
  & [PARSEDCITY :: Address.City ]
  & [PARSEDSTATE].REQUIRED
  & [PARSEDSTATE :: Address.State ]
  & [PARSEDZIP].REQUIRED
  & [PARSEDZIP :: Address.Zip ]

  POSTCONDITION
  {
    [CVFIRSTNAME1 :: Name.First ]
    [CVMIDDLENAME1 :: Name.Middle ]
    [CVLASTNAME1 :: Name.Last ]
    [CVGENDERCODE1 :: Name.Gender ]
    [CVADDRESSTYPE :: Address.Type ]
    [CVPRIMARYADDRESS :: Address.Primary ]
    [CVSECONDARYADDRESS :: Address.Secondary ]
    [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
    [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
    [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
```

```
    [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
    [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
    [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
    [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
    [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
    [CVAACITYNAME :: Address.City +UspsStandardized]
    [CVAASTATEABBREV :: Address.State +UspsStandardized]
    [CVAAZIPCODE :: Address.Zip +UspsStandardized]
    [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
  }
}

PRECONDITION
{
  [UNPARSEDNAME].NA
  & [PARSEDFIRSTNAME].REQUIRED
  & [PARSEDFIRSTNAME :: Name.First ]
  & [PARSEDLASTNAME].REQUIRED
  & [PARSEDLASTNAME :: Name.Last ]
  & [ADDRESSLINE1].REQUIRED
  & [ADDRESSLINE1 :: Address.Primary ]
  & [PROFESSIONALTITLE].REQUIRED
  & [PROFESSIONALTITLE :: Name.ProfessionalTitle ]
  & [PRIMARYBUSINESSNAME].NA
  & [SECONDARYBUSINESSNAME].NA
  & [UNPARSEDCITYSTATEZIP].NA
  & [PARSEDCITY].REQUIRED
  & [PARSEDCITY :: Address.City ]
  & [PARSEDSTATE].REQUIRED
  & [PARSEDSTATE :: Address.State ]
  & [PARSEDZIP].REQUIRED
  & [PARSEDZIP :: Address.Zip ]

  POSTCONDITION
  {
    [CVFIRSTNAME1 :: Name.First ]
    [CVMIDDLENAME1 :: Name.Middle ]
    [CVLASTNAME1 :: Name.Last ]
    [CVGENDERCODE1 :: Name.Gender ]
    [CVADDRESSTYPE :: Address.Type ]
    [CVPRIMARYADDRESS :: Address.Primary ]
    [CVSECONDARYADDRESS :: Address.Secondary ]
    [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
    [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
    [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
    [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
    [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
    [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
    [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
    [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
    [CVAACITYNAME :: Address.City +UspsStandardized]
    [CVAASTATEABBREV :: Address.State +UspsStandardized]
    [CVAAZIPCODE :: Address.Zip +UspsStandardized]
    [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
    [CVPROFTITLE1 :: Name.ProfessionalTitle ]
  }
```

```
}

PRECONDITION
{
   [UNPARSEDNAME].NA
   & [PARSEDFIRSTNAME].REQUIRED
   & [PARSEDFIRSTNAME :: Name.First ]
   & [PARSEDLASTNAME].REQUIRED
   & [PARSEDLASTNAME :: Name.Last ]
   & [ADDRESSLINE1].REQUIRED
   & [ADDRESSLINE1 :: Address.Primary ]
   & [PROFESSIONALTITLE].NA
   & [PRIMARYBUSINESSNAME].REQUIRED
   & [PRIMARYBUSINESSNAME :: Name.Business ]
   & [SECONDARYBUSINESSNAME].NA
   & [UNPARSEDCITYSTATEZIP].NA
   & [PARSEDCITY].REQUIRED
   & [PARSEDCITY :: Address.City ]
   & [PARSEDSTATE].REQUIRED
   & [PARSEDSTATE :: Address.State ]
   & [PARSEDZIP].REQUIRED
   & [PARSEDZIP :: Address.Zip ]

   POSTCONDITION
   {
      [CVFIRSTNAME1 :: Name.First ]
      [CVMIDDLENAME1 :: Name.Middle ]
      [CVLASTNAME1 :: Name.Last ]
      [CVGENDERCODE1 :: Name.Gender ]
      [CVADDRESSTYPE :: Address.Type ]
      [CVPRIMARYADDRESS :: Address.Primary ]
      [CVSECONDARYADDRESS :: Address.Secondary ]
      [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
      [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
      [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
      [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
      [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
      [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
      [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
      [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
      [CVAACITYNAME :: Address.City +UspsStandardized]
      [CVAASTATEABBREV :: Address.State +UspsStandardized]
      [CVAAZIPCODE :: Address.Zip +UspsStandardized]
      [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
      [CVPRIMARYBUSINESS :: Name.Business ]
      [CVAABUSNAME :: Name.Business +UspsStandardized]
   }
}

PRECONDITION
{
   [UNPARSEDNAME].NA
   & [PARSEDFIRSTNAME].REQUIRED
   & [PARSEDFIRSTNAME :: Name.First ]
   & [PARSEDLASTNAME].REQUIRED
   & [PARSEDLASTNAME :: Name.Last ]
```

```
   & [ADDRESSLINE1].REQUIRED
   & [ADDRESSLINE1 :: Address.Primary ]
   & [PROFESSIONALTITLE].NA
   & [PRIMARYBUSINESSNAME].NA
   & [SECONDARYBUSINESSNAME].REQUIRED
   & [SECONDARYBUSINESSNAME :: Name.Business ]
   & [UNPARSEDCITYSTATEZIP].NA
   & [PARSEDCITY].REQUIRED
   & [PARSEDCITY :: Address.City ]
   & [PARSEDSTATE].REQUIRED
   & [PARSEDSTATE :: Address.State ]
   & [PARSEDZIP].REQUIRED
   & [PARSEDZIP :: Address.Zip ]

   POSTCONDITION
   {
      [CVFIRSTNAME1 :: Name.First ]
      [CVMIDDLENAME1 :: Name.Middle ]
      [CVLASTNAME1 :: Name.Last ]
      [CVGENDERCODE1 :: Name.Gender ]
      [CVADDRESSTYPE :: Address.Type ]
      [CVPRIMARYADDRESS :: Address.Primary ]
      [CVSECONDARYADDRESS :: Address.Secondary ]
      [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
      [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
      [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
      [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
      [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
      [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
      [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
      [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
      [CVAACITYNAME :: Address.City +UspsStandardized]
      [CVAASTATEABBREV :: Address.State +UspsStandardized]
      [CVAAZIPCODE :: Address.Zip +UspsStandardized]
      [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
      [CVSECONDARYBUSINESS :: Name.Business ]
   }
}

PRECONDITION
{
   [UNPARSEDNAME].NA
   & [PARSEDFIRSTNAME].REQUIRED
   & [PARSEDFIRSTNAME :: Name.First ]
   & [PARSEDLASTNAME].REQUIRED
   & [PARSEDLASTNAME :: Name.Last ]
   & [ADDRESSLINE1].REQUIRED
   & [ADDRESSLINE1 :: Address.Primary ]
   & [PROFESSIONALTITLE].REQUIRED
   & [PROFESSIONALTITLE :: Name.ProfessionalTitle ]
   & [PRIMARYBUSINESSNAME].REQUIRED
   & [PRIMARYBUSINESSNAME :: Name.Business ]
   & [SECONDARYBUSINESSNAME].NA
   & [UNPARSEDCITYSTATEZIP].NA
   & [PARSEDCITY].REQUIRED
   & [PARSEDCITY :: Address.City ]
```

```
   & [PARSEDSTATE].REQUIRED
   & [PARSEDSTATE :: Address.State ]
   & [PARSEDZIP].REQUIRED
   & [PARSEDZIP :: Address.Zip ]

   POSTCONDITION
   {
      [CVFIRSTNAME1 :: Name.First ]
      [CVMIDDLENAME1 :: Name.Middle ]
      [CVLASTNAME1 :: Name.Last ]
      [CVGENDERCODE1 :: Name.Gender ]
      [CVADDRESSTYPE :: Address.Type ]
      [CVPRIMARYADDRESS :: Address.Primary ]
      [CVSECONDARYADDRESS :: Address.Secondary ]
      [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
      [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
      [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
      [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
      [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
      [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
      [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
      [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
      [CVAACITYNAME :: Address.City +UspsStandardized]
      [CVAASTATEABBREV :: Address.State +UspsStandardized]
      [CVAAZIPCODE :: Address.Zip +UspsStandardized]
      [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
      [CVPROFTITLE1 :: Name.ProfessionalTitle ]
      [CVPRIMARYBUSINESS :: Name.Business ]
      [CVAABUSNAME :: Name.Business +UspsStandardized]
   }
}

PRECONDITION
{
   [UNPARSEDNAME].NA
   & [PARSEDFIRSTNAME].REQUIRED
   & [PARSEDFIRSTNAME :: Name.First ]
   & [PARSEDLASTNAME].REQUIRED
   & [PARSEDLASTNAME :: Name.Last ]
   & [ADDRESSLINE1].REQUIRED
   & [ADDRESSLINE1 :: Address.Primary ]
   & [PROFESSIONALTITLE].REQUIRED
   & [PROFESSIONALTITLE :: Name.ProfessionalTitle ]
   & [PRIMARYBUSINESSNAME].NA
   & [SECONDARYBUSINESSNAME].REQUIRED
   & [SECONDARYBUSINESSNAME :: Name.Business ]
   & [UNPARSEDCITYSTATEZIP].NA
   & [PARSEDCITY].REQUIRED
   & [PARSEDCITY :: Address.City ]
   & [PARSEDSTATE].REQUIRED
   & [PARSEDSTATE :: Address.State ]
   & [PARSEDZIP].REQUIRED
   & [PARSEDZIP :: Address.Zip ]

   POSTCONDITION
   {
```

```
    [CVFIRSTNAME1 :: Name.First ]
    [CVMIDDLENAME1 :: Name.Middle ]
    [CVLASTNAME1 :: Name.Last ]
    [CVGENDERCODE1 :: Name.Gender ]
    [CVADDRESSTYPE :: Address.Type ]
    [CVPRIMARYADDRESS :: Address.Primary ]
    [CVSECONDARYADDRESS :: Address.Secondary ]
    [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
    [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
    [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
    [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
    [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
    [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
    [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
    [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
    [CVAACITYNAME :: Address.City +UspsStandardized]
    [CVAASTATEABBREV :: Address.State +UspsStandardized]
    [CVAAZIPCODE :: Address.Zip +UspsStandardized]
    [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
    [CVPROFTITLE1 :: Name.ProfessionalTitle ]
    [CVSECONDARYBUSINESS :: Name.Business ]
  }
}

PRECONDITION
{
  [UNPARSEDNAME].NA
  & [PARSEDFIRSTNAME].REQUIRED
  & [PARSEDFIRSTNAME :: Name.First ]
  & [PARSEDLASTNAME].REQUIRED
  & [PARSEDLASTNAME :: Name.Last ]
  & [ADDRESSLINE1].REQUIRED
  & [ADDRESSLINE1 :: Address.Primary ]
  & [PROFESSIONALTITLE].NA
  & [PRIMARYBUSINESSNAME].REQUIRED
  & [PRIMARYBUSINESSNAME :: Name.Business ]
  & [SECONDARYBUSINESSNAME].REQUIRED
  & [SECONDARYBUSINESSNAME :: Name.Business ]
  & [UNPARSEDCITYSTATEZIP].NA
  & [PARSEDCITY].REQUIRED
  & [PARSEDCITY :: Address.City ]
  & [PARSEDSTATE].REQUIRED
  & [PARSEDSTATE :: Address.State ]
  & [PARSEDZIP].REQUIRED
  & [PARSEDZIP :: Address.Zip ]

  POSTCONDITION
  {
    [CVFIRSTNAME1 :: Name.First ]
    [CVMIDDLENAME1 :: Name.Middle ]
    [CVLASTNAME1 :: Name.Last ]
    [CVGENDERCODE1 :: Name.Gender ]
    [CVADDRESSTYPE :: Address.Type ]
    [CVPRIMARYADDRESS :: Address.Primary ]
    [CVSECONDARYADDRESS :: Address.Secondary ]
    [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
```

```
    [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
    [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
    [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
    [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
    [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
    [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
    [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
    [CVAACITYNAME :: Address.City +UspsStandardized]
    [CVAASTATEABBREV :: Address.State +UspsStandardized]
    [CVAAZIPCODE :: Address.Zip +UspsStandardized]
    [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
    [CVPRIMARYBUSINESS :: Name.Business ]
    [CVSECONDARYBUSINESS :: Name.Business ]
    [CVAABUSNAME :: Name.Business +UspsStandardized]
  }
}

PRECONDITION
{
  [UNPARSEDNAME].NA
  & [PARSEDFIRSTNAME].REQUIRED
  & [PARSEDFIRSTNAME :: Name.First ]
  & [PARSEDLASTNAME].REQUIRED
  & [PARSEDLASTNAME :: Name.Last ]
  & [ADDRESSLINE1].REQUIRED
  & [ADDRESSLINE1 :: Address.Primary ]
  & [PROFESSIONALTITLE].REQUIRED
  & [PROFESSIONALTITLE :: Name.ProfessionalTitle ]
  & [PRIMARYBUSINESSNAME].REQUIRED
  & [PRIMARYBUSINESSNAME :: Name.Business ]
  & [SECONDARYBUSINESSNAME].REQUIRED
  & [SECONDARYBUSINESSNAME :: Name.Business ]
  & [UNPARSEDCITYSTATEZIP].NA
  & [PARSEDCITY].REQUIRED
  & [PARSEDCITY :: Address.City ]
  & [PARSEDSTATE].REQUIRED
  & [PARSEDSTATE :: Address.State ]
  & [PARSEDZIP].REQUIRED
  & [PARSEDZIP :: Address.Zip ]

  POSTCONDITION
  {
    [CVFIRSTNAME1 :: Name.First ]
    [CVMIDDLENAME1 :: Name.Middle ]
    [CVLASTNAME1 :: Name.Last ]
    [CVGENDERCODE1 :: Name.Gender ]
    [CVADDRESSTYPE :: Address.Type ]
    [CVPRIMARYADDRESS :: Address.Primary ]
    [CVSECONDARYADDRESS :: Address.Secondary ]
    [CVAAPRINUMBER :: Address.PrimaryNumber +UspsStandardized]
    [CVAAPREDIR1 :: Address.PreDirectional +UspsStandardized]
    [CVAAPRINAME :: Address.PrimaryName +UspsStandardized]
    [CVAASUFFIX1 :: Address.Suffix +UspsStandardized]
    [CVAAPOSTDIR :: Address.PostDirectional +UspsStandardized]
    [CVAASECUNITDES :: Address.SecondaryUnitDesignator +UspsStandardized]
    [CVAASECNUMBER :: Address.SecondaryNumber +UspsStandardized]
```

```
        [CVAAOUTPRURB :: Address.UrbanizationName +UspsStandardized]
        [CVAACITYNAME :: Address.City +UspsStandardized]
        [CVAASTATEABBREV :: Address.State +UspsStandardized]
        [CVAAZIPCODE :: Address.Zip +UspsStandardized]
        [CVAAADDONCODE :: Address.Zip4 +UspsStandardized]
        [CVPROFTITLE1 :: Name.ProfessionalTitle ]
        [CVPRIMARYBUSINESS :: Name.Business ]
        [CVSECONDARYBUSINESS :: Name.Business ]
        [CVAABUSNAME :: Name.Business +UspsStandardized]
      }
    }
}

OPTIONS
{

   FilterProfanity
   {

      OPTION SETTINGS
      {

         FilterWords
         {

            PRECONDITIONS
            {

               PRECONDITION
               {
                  NONE

                  POSTCONDITION
                  {
                     [CVFIRSTNAME1 :: Name.First +ProfanityFiltered]
                     [CVMIDDLENAME1 :: Name.Middle +ProfanityFiltered]
                     [CVLASTNAME1 :: Name.Last +ProfanityFiltered]
                     [CVPROFTITLE1 :: Name.ProfessionalTitle +ProfanityFiltered]
                     [CVPRIMARYBUSINESS :: Name.Business +ProfanityFiltered]
                     [CVSECONDARYBUSINESS :: Name.Business +ProfanityFiltered]
                     [CVPRIMARYADDRESS :: Address.Primary +ProfanityFiltered]
                     [CVSECONDARYADDRESS :: Address.Secondary +ProfanityFiltered]
                     [CVAAPRINUMBER :: Address.PrimaryNumber +ProfanityFiltered]
                     [CVAAPREDIR1 :: Address.PreDirectional +ProfanityFiltered]
                     [CVAAPRINAME :: Address.PrimaryName +ProfanityFiltered]
                     [CVAASUFFIX1 :: Address.Suffix +ProfanityFiltered]
                     [CVAAPOSTDIR :: Address.PostDirectional +ProfanityFiltered]
                     [CVAASECUNITDES :: Address.SecondaryUnitDesignator +ProfanityFiltered]
                     [CVAASECNUMBER :: Address.SecondaryNumber +ProfanityFiltered]
                     [CVAABUSNAME :: Name.Business +ProfanityFiltered]
                     [CVAAOUTPRURB :: Address.UrbanizationName +ProfanityFiltered]
                     [CVAACITYNAME :: Address.City +ProfanityFiltered]
                     [CVAASTATEABBREV :: Address.State +ProfanityFiltered]
                     [CVAAZIPCODE :: Address.Zip +ProfanityFiltered]
                     [CVAAADDONCODE :: Address.Zip4 +ProfanityFiltered]
                  }
```

```
            }
         }
      }

      DoNotFilterWords
      {

         PRECONDITIONS
         {

            PRECONDITION
            {
               NONE

               POSTCONDITION
               {
               }
            }
         }
      }
   }
}
```

A.8 PremiumAddress

```
PremiumAddress
{

   INPUT FIELDS
   {
      ADDRESSLINK
      CONSUMERLINK
      COARECORDMATCHCODE
      PRIMARYNUMBER
      PREDIRECTIONAL
      STREETNAME
      STREETSUFFIX
      POSTDIRECTIONAL
      UNITDESIGNATOR
      SECONDARYNUMBER
      CITYNAME
      STATEABBREV
      ZIP
      ZIP4
   }

   OUTPUT FIELDS
   {
      IBBA_PRIMARY_NUMBER
      IBBA_PRE_DIRECTIONAL
      IBBA_STREET_NAME
      IBBA_STREET_SUFFIX
      IBBA_POST_DIRECTIONAL
      IBBA_UNIT_DESIGNATOR
      IBBA_SECONDARY_NUMBER
      IBBA_CITY
      IBBA_STATE
      IBBA_ZIP_CODE
      IBBA_ZIP_4_CODE
      IBBA_PRIMARY_ADDRESS
      IBBA_ECP_PRIMARY_NUMBER
      IBBA_ECP_PRE_DIRECTIONAL
      IBBA_ECP_STREET_NAME
      IBBA_ECP_STREET_SUFFIX
      IBBA_ECP_POST_DIRECTIONAL
      IBBA_ECP_UNIT_DESIGNATOR
      IBBA_ECP_SECONDARY_NUMBER
      IBBA_ECP_CITY
      IBBA_ECP_STATE
      IBBA_ECP_ZIP_CODE
      IBBA_ECP_ZIP_4_CODE
      IBBA_ECP_PRIMARY_ADDRESS
      IBBA_DPD_PRIMARY_ADDRESS
      IBBA_DPD_CITY
      IBBA_DPD_STATE
      IBBA_DPD_ZIP_CODE
   }
```

```
PRECONDITIONS
{

   PRECONDITION
   {
      NONE

      POSTCONDITION
      {
      }
   }
}

OPTIONS
{

   PerformHygiene
   {

      OPTION SETTINGS
      {

         N
         {

            PRECONDITIONS
            {
            }
         }

         Y
         {

            PRECONDITIONS
            {

               PRECONDITION
               {
                  [ADDRESSLINK].REQUIRED
                  & [ADDRESSLINK :: Link.Address ]

                  POSTCONDITION
                  {
                     [IBBA_PRIMARY_NUMBER :: Address.PrimaryNumber +PremiumStandardized]
                     [IBBA_PRE_DIRECTIONAL :: Address.PreDirectional +PremiumStandardized]
                     [IBBA_STREET_NAME :: Address.PrimaryName +PremiumStandardized]
                     [IBBA_STREET_SUFFIX :: Address.Suffix +PremiumStandardized]
                     [IBBA_POST_DIRECTIONAL :: Address.PostDirectional +PremiumStandardized]
                     [IBBA_UNIT_DESIGNATOR :: Address.SecondaryUnitDesignator +PremiumStandardized]
                     [IBBA_SECONDARY_NUMBER :: Address.SecondaryNumber +PremiumStandardized]
                     [IBBA_CITY :: Address.City +PremiumStandardized]
                     [IBBA_STATE :: Address.State +PremiumStandardized]
                     [IBBA_ZIP_CODE :: Address.Zip +PremiumStandardized]
                     [IBBA_ZIP_4_CODE :: Address.Zip4 +PremiumStandardized]
                     [IBBA_PRIMARY_ADDRESS :: Address.Primary +PremiumStandardized]
                  }
```

```
            }
          }
        }
      }
    }

    PerformEnhancedChangeOfAddress
    {

      OPTION SETTINGS
      {

        N
        {

          PRECONDITIONS
          {
          }
        }

        Y
        {

          PRECONDITIONS
          {

            PRECONDITION
            {
              [ADDRESSLINK].REQUIRED
              & [ADDRESSLINK :: Link.Address ]
              & [CONSUMERLINK].REQUIRED
              & [CONSUMERLINK :: Link.Consumer ]

              POSTCONDITION
              {
                [IBBA_ECP_PRIMARY_NUMBER :: Address.PrimaryNumber +PremiumAddressChange]
                [IBBA_ECP_PRE_DIRECTIONAL :: Address.PreDirectional +PremiumAddressChange]
                [IBBA_ECP_STREET_NAME :: Address.PrimaryName +PremiumAddressChange]
                [IBBA_ECP_STREET_SUFFIX :: Address.Suffix +PremiumAddressChange]
                [IBBA_ECP_POST_DIRECTIONAL :: Address.PostDirectional +PremiumAddressChange]
                [IBBA_ECP_UNIT_DESIGNATOR :: Address.SecondaryUnitDesignator
+PremiumAddressChange]
                [IBBA_ECP_SECONDARY_NUMBER :: Address.SecondaryNumber
+PremiumAddressChange]
                [IBBA_ECP_CITY :: Address.City +PremiumAddressChange]
                [IBBA_ECP_STATE :: Address.State +PremiumAddressChange]
                [IBBA_ECP_ZIP_CODE :: Address.Zip +PremiumAddressChange]
                [IBBA_ECP_ZIP_4_CODE :: Address.Zip4 +PremiumAddressChange]
                [IBBA_ECP_PRIMARY_ADDRESS :: Address.Primary +PremiumAddressChange]
              }
            }
          }
        }
      }
    }
  }
```

```
}
```

APPENDIX B: INTENTS

B.1 Address hygiene

```
Address Hygiene
{

   POSTCONDITIONS
   {
          [PRIMARYADDRESS :: Address.Primary
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
          [SECONDARYADDRESS :: Address.Secondary
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
          [PRIMARYNUMBER :: Address.PrimaryNumber
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
          [PREDIRECTIONAL :: Address.PreDirectional
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
          [PRINAME :: Address.PrimaryName
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
          [SUFFIX :: Address.Suffix +UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
          [POSTDIRECTIONAL :: Address.PostDirectional
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
          [SECUNITDES :: Address.SecondaryUnitDesignator
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
          [SECNUMBER :: Address.SecondaryNumber
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
          [BUSADDRESS :: Address.Business
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
          [CITYNAME :: Address.City +UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
          [STATE :: Address.State +UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
          [ZIPCODE :: Address.Zip +UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
          [ZIP4 :: Address.Zip4 +UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
          [URBNAME :: Address.UrbanizationName
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
   }
}
```

B.2 Premium address hygiene

```
Premium Address Hygiene
{

   POSTCONDITIONS
   {
         [PRIMARYADDRESS :: Address.Primary +PremiumStandardized+CurrentAddressSelected]
         [SECONDARYADDRESS :: Address.Secondary +PremiumStandardized+CurrentAddressSelected]
         [PRIMARYNUMBER :: Address.PrimaryNumber +PremiumStandardized+CurrentAddressSelected]
         [PREDIRECTIONAL :: Address.PreDirectional +PremiumStandardized+CurrentAddressSelected]
         [PRINAME :: Address.PrimaryName +PremiumStandardized+CurrentAddressSelected]
         [SUFFIX :: Address.Suffix +PremiumStandardized+CurrentAddressSelected]
         [POSTDIRECTIONAL :: Address.PostDirectional +PremiumStandardized+CurrentAddressSelected]
         [SECUNITDES :: Address.SecondaryUnitDesignator +PremiumStandardized+CurrentAddressSelected]
         [SECNUMBER :: Address.SecondaryNumber +PremiumStandardized+CurrentAddressSelected]
         [BUSADDRESS :: Address.Business +PremiumStandardized+CurrentAddressSelected]
         [CITYNAME :: Address.City +PremiumStandardized+CurrentAddressSelected]
         [STATE :: Address.State +PremiumStandardized+CurrentAddressSelected]
         [ZIPCODE :: Address.Zip +PremiumStandardized+CurrentAddressSelected]
         [ZIP4 :: Address.Zip4 +PremiumStandardized+CurrentAddressSelected]
         [URBNAME :: Address.UrbanizationName +PremiumStandardized+CurrentAddressSelected]
   }
}
```

B.3 Change of address

```
Change Of Address
{

   POSTCONDITIONS
   {
          [PRIMARYADDRESS :: Address.Primary +AddressChange+CurrentAddressSelected]
          [SECONDARYADDRESS :: Address.Secondary +AddressChange+CurrentAddressSelected]
          [PRIMARYNUMBER :: Address.PrimaryNumber +AddressChange+CurrentAddressSelected]
          [PREDIRECTIONAL :: Address.PreDirectional +AddressChange+CurrentAddressSelected]
          [PRINAME :: Address.PrimaryName +AddressChange+CurrentAddressSelected]
          [SUFFIX :: Address.Suffix +AddressChange+CurrentAddressSelected]
          [POSTDIRECTIONAL :: Address.PostDirectional +AddressChange+CurrentAddressSelected]
          [SECUNITDES :: Address.SecondaryUnitDesignator +AddressChange+CurrentAddressSelected]
          [SECNUMBER :: Address.SecondaryNumber +AddressChange+CurrentAddressSelected]
          [BUSADDRESS :: Address.Business +AddressChange+CurrentAddressSelected]
          [CITYNAME :: Address.City +AddressChange+CurrentAddressSelected]
          [STATE :: Address.State +AddressChange+CurrentAddressSelected]
          [ZIPCODE :: Address.Zip +AddressChange+CurrentAddressSelected]
          [ZIP4 :: Address.Zip4 +AddressChange+CurrentAddressSelected]
          [URBNAME :: Address.UrbanizationName +AddressChange+CurrentAddressSelected]
          [ADDRESSFLAGS :: Address.Flags +AddressChange+CurrentAddressSelected]
   }
}
```

B.4 Premium change of address

```
Premium Change Of Address
{

   POSTCONDITIONS
   {
          [PRIMARYADDRESS :: Address.Primary +PremiumAddressChange+CurrentAddressSelected]
          [SECONDARYADDRESS :: Address.Secondary +PremiumAddressChange+CurrentAddressSelected]
          [PRIMARYNUMBER :: Address.PrimaryNumber +PremiumAddressChange+CurrentAddressSelected]
          [PREDIRECTIONAL :: Address.PreDirectional +PremiumAddressChange+CurrentAddressSelected]
          [PRINAME :: Address.PrimaryName +PremiumAddressChange+CurrentAddressSelected]
          [SUFFIX :: Address.Suffix +PremiumAddressChange+CurrentAddressSelected]
          [POSTDIRECTIONAL :: Address.PostDirectional +PremiumAddressChange+CurrentAddressSelected]
          [SECUNITDES :: Address.SecondaryUnitDesignator
+PremiumAddressChange+CurrentAddressSelected]
          [SECNUMBER :: Address.SecondaryNumber +PremiumAddressChange+CurrentAddressSelected]
          [BUSADDRESS :: Address.Business +PremiumAddressChange+CurrentAddressSelected]
          [CITYNAME :: Address.City +PremiumAddressChange+CurrentAddressSelected]
          [STATE :: Address.State +PremiumAddressChange+CurrentAddressSelected]
          [ZIPCODE :: Address.Zip +PremiumAddressChange+CurrentAddressSelected]
          [ZIP4 :: Address.Zip4 +PremiumAddressChange+CurrentAddressSelected]
          [URBNAME :: Address.UrbanizationName +PremiumAddressChange+CurrentAddressSelected]
   }
}
```

B.5 Filter profanity

```
Filter Profanity
{

   POSTCONDITIONS
   {
          [FIRSTNAME :: Name.First +ProfanityFiltered]
          [LASTNAME :: Name.Last +ProfanityFiltered]
          [PRIMARYADDRESS :: Address.Primary +ProfanityFiltered]
          [SECONDARYADDRESS :: Address.Secondary +ProfanityFiltered]
          [PRIMARYNUMBER :: Address.PrimaryNumber +ProfanityFiltered]
          [PREDIRECTIONAL :: Address.PreDirectional +ProfanityFiltered]
          [PRINAME :: Address.PrimaryName +ProfanityFiltered]
          [SUFFIX :: Address.Suffix +ProfanityFiltered]
          [POSTDIRECTIONAL :: Address.PostDirectional +ProfanityFiltered]
          [SECUNITDES :: Address.SecondaryUnitDesignator +ProfanityFiltered]
          [SECNUMBER :: Address.SecondaryNumber +ProfanityFiltered]
          [CITYNAME :: Address.City +ProfanityFiltered]
          [STATE :: Address.State +ProfanityFiltered]
          [ZIPCODE :: Address.Zip +ProfanityFiltered]
          [ZIP4 :: Address.Zip4 +ProfanityFiltered]
          [URBNAME :: Address.UrbanizationName +ProfanityFiltered]
   }
}
```

B.6 Validate names

```
Validate Names
{

   POSTCONDITIONS
   {
          [FIRSTNAME :: Name.First +NameEditChecked]
          [LASTNAME :: Name.Last +NameEditChecked]
   }
}
```

B.7 Determine industry demographic

```
Determine Industry Demographic
{

   POSTCONDITIONS
   {
            [PROFESSIONALTITLE :: Name.ProfessionalTitle +IndustryCoded]
   }
}
```

B.8 Validate addresses

```
Validate Addresses
{

   POSTCONDITIONS
   {
            [PRIMARYADDRESS :: Address.Primary +AddressEditChecked]
   }
}
```

B.9 Delivery sequencing

```
Delivery Sequencing
{

   POSTCONDITIONS
   {
          [PRIMARYADDRESS :: Address.Primary +DeliverySequenced]
          [SECONDARYADDRESS :: Address.Secondary +DeliverySequenced]
          [PRIMARYNUMBER :: Address.PrimaryNumber +DeliverySequenced]
          [PREDIRECTIONAL :: Address.PreDirectional +DeliverySequenced]
          [PRINAME :: Address.PrimaryName +DeliverySequenced]
          [SUFFIX :: Address.Suffix +DeliverySequenced]
          [POSTDIRECTIONAL :: Address.PostDirectional +DeliverySequenced]
          [SECUNITDES :: Address.SecondaryUnitDesignator +DeliverySequenced]
          [SECNUMBER :: Address.SecondaryNumber +DeliverySequenced]
          [BUSADDRESS :: Address.Business +DeliverySequenced]
          [CITYNAME :: Address.City +DeliverySequenced]
          [STATE :: Address.State +DeliverySequenced]
          [ZIPCODE :: Address.Zip +DeliverySequenced]
          [ZIP4 :: Address.Zip4 +DeliverySequenced]
          [URBNAME :: Address.UrbanizationName +DeliverySequenced]
   }
}
```

B.10 Geocode addresses

```
Geocode Addresses
{

   POSTCONDITIONS
   {
          [PRIMARYADDRESS :: Address.Primary +Geocoded]
          [SECONDARYADDRESS :: Address.Secondary +Geocoded]
          [PRIMARYNUMBER :: Address.PrimaryNumber +Geocoded]
          [PREDIRECTIONAL :: Address.PreDirectional +Geocoded]
          [PRINAME :: Address.PrimaryName +Geocoded]
          [SUFFIX :: Address.Suffix +Geocoded]
          [POSTDIRECTIONAL :: Address.PostDirectional +Geocoded]
          [SECUNITDES :: Address.SecondaryUnitDesignator +Geocoded]
          [SECNUMBER :: Address.SecondaryNumber +Geocoded]
          [BUSADDRESS :: Address.Business +Geocoded]
          [CITYNAME :: Address.City +Geocoded]
          [STATE :: Address.State +Geocoded]
          [ZIPCODE :: Address.Zip +Geocoded]
          [ZIP4 :: Address.Zip4 +Geocoded]
          [URBNAME :: Address.UrbanizationName +Geocoded]
   }
}
```

B.11 Link contacts

```
Link Contacts
{

   POSTCONDITIONS
   {
          [ADDRESSLINK :: Link.Address +AddressContactLinked]
          [PRIMARYADDRESS :: Address.Primary +CurrentAddressSelected]
          [SECONDARYADDRESS :: Address.Secondary +CurrentAddressSelected]
          [PRIMARYNUMBER :: Address.PrimaryNumber +CurrentAddressSelected]
          [PREDIRECTIONAL :: Address.PreDirectional +CurrentAddressSelected]
          [PRINAME :: Address.PrimaryName +CurrentAddressSelected]
          [SUFFIX :: Address.Suffix +CurrentAddressSelected]
          [POSTDIRECTIONAL :: Address.PostDirectional +CurrentAddressSelected]
          [SECUNITDES :: Address.SecondaryUnitDesignator +CurrentAddressSelected]
          [SECNUMBER :: Address.SecondaryNumber +CurrentAddressSelected]
          [BUSADDRESS :: Address.Business +CurrentAddressSelected]
          [CITYNAME :: Address.City +CurrentAddressSelected]
          [STATE :: Address.State +CurrentAddressSelected]
          [ZIPCODE :: Address.Zip +CurrentAddressSelected]
          [ZIP4 :: Address.Zip4 +CurrentAddressSelected]
          [URBNAME :: Address.UrbanizationName +CurrentAddressSelected]
          [ADDRESSFLAGS :: Address.Flags +CurrentAddressSelected]
   }
}
```

APPENDIX C: GENERATED WORKFLOWS

C.1 Testing Scenario 1

C.1.1 Distinct Sequences

1. Parser => AddressEnhance => NameEditCheck => AddressSelect => IndustryCode
2. Parser => AddressEnhance => NameEditCheck => IndustryCode => AddressSelect
3. Parser => AddressEnhance => AddressSelect => NameEditCheck => IndustryCode
4. Parser => AddressEnhance => AddressSelect => IndustryCode => NameEditCheck
5. Parser => AddressEnhance => IndustryCode => NameEditCheck => AddressSelect
6. Parser => AddressEnhance => IndustryCode => AddressSelect => NameEditCheck
7. Parser => NameEditCheck => AddressEnhance => AddressSelect => IndustryCode
8. Parser => NameEditCheck => AddressEnhance => IndustryCode => AddressSelect
9. Parser => NameEditCheck => IndustryCode => AddressEnhance => AddressSelect
10. Parser => IndustryCode => AddressEnhance => NameEditCheck => AddressSelect
11. Parser => IndustryCode => AddressEnhance => AddressSelect => NameEditCheck
12. Parser => IndustryCode => NameEditCheck => AddressEnhance => AddressSelect

C.1.2 Detailed Result

```
WORKFLOW
{

   INITIAL STATE
   {
      Address.Primary::Address.Primary
      Name.Full::Name.Full
      Address.CityStateZip::Address.CityStateZip
   }

   GOAL STATE
   {
      [PRIMARYADDRESS :: Address.Primary
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
      [SECONDARYADDRESS :: Address.Secondary
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
      [PRIMARYNUMBER :: Address.PrimaryNumber
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
      [PREDIRECTIONAL :: Address.PreDirectional
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
      [PRINAME :: Address.PrimaryName +UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
      [SUFFIX :: Address.Suffix +UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
      [POSTDIRECTIONAL :: Address.PostDirectional
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
      [SECUNITDES :: Address.SecondaryUnitDesignator
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
      [SECNUMBER :: Address.SecondaryNumber
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
      [BUSADDRESS :: Address.Business +UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
```

```
    [CITYNAME :: Address.City +UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
    [STATE :: Address.State +UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
    [ZIPCODE :: Address.Zip +UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
    [ZIP4 :: Address.Zip4 +UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
    [URBNAME :: Address.UrbanizationName
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
    [FIRSTNAME :: Name.First +NameEditChecked]
    [LASTNAME :: Name.Last +NameEditChecked]
    [PROFESSIONALTITLE :: Name.ProfessionalTitle +IndustryCoded]
  }

  WORKFLOW STEPS
  {

    Parser
    {

      INPUT MAPPINGS
      {
        UNPARSEDNAME <= [Name.Full::Name.Full]
        ADDRESSLINE1 <= [Address.Primary::Address.Primary]
        UNPARSEDCITYSTATEZIP <= [Address.CityStateZip::Address.CityStateZip]
      }

      OPTION MAPPINGS
      {
        FilterProfanity <= (DoNotFilterWords)
      }

      END STATE
      {
        Address.Primary::Address.Primary
        Name.Full::Name.Full
        Address.CityStateZip::Address.CityStateZip
        CVFIRSTNAME1::Name.First
        CVMIDDLENAME1::Name.Middle
        CVLASTNAME1::Name.Last
        CVGENDERCODE1::Name.Gender
        CVADDRESSTYPE::Address.Type
        CVPRIMARYADDRESS::Address.Primary
        CVSECONDARYADDRESS::Address.Secondary
        CVAAPRINUMBER::Address.PrimaryNumber +UspsStandardized
        CVAAPREDIR1::Address.PreDirectional +UspsStandardized
        CVAAPRINAME::Address.PrimaryName +UspsStandardized
        CVAASUFFIX1::Address.Suffix +UspsStandardized
        CVAAPOSTDIR::Address.PostDirectional +UspsStandardized
        CVAASECUNITDES::Address.SecondaryUnitDesignator +UspsStandardized
        CVAASECNUMBER::Address.SecondaryNumber +UspsStandardized
        CVAAOUTPRURB::Address.UrbanizationName +UspsStandardized
        CVAACITYNAME::Address.City +UspsStandardized
        CVAASTATEABBREV::Address.State +UspsStandardized
        CVAAZIPCODE::Address.Zip +UspsStandardized
        CVAAADDONCODE::Address.Zip4 +UspsStandardized
        CVPROFTITLE1::Name.ProfessionalTitle
        CVPRIMARYBUSINESS::Name.Business
        CVSECONDARYBUSINESS::Name.Business
```

```
        CVAABUSNAME::Name.Business +UspsStandardized
      }

      REMAINING GOALS
      {
        [CVPRIMARYADDRESS :: Address.Primary
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
        [CVSECONDARYADDRESS :: Address.Secondary
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
        [CVAAPRINUMBER :: Address.PrimaryNumber +DeliverabilityValidated+CurrentAddressSelected]
        [CVAAPREDIR1 :: Address.PreDirectional +DeliverabilityValidated+CurrentAddressSelected]
        [CVAAPRINAME :: Address.PrimaryName +DeliverabilityValidated+CurrentAddressSelected]
        [CVAASUFFIX1 :: Address.Suffix +DeliverabilityValidated+CurrentAddressSelected]
        [CVAAPOSTDIR :: Address.PostDirectional +DeliverabilityValidated+CurrentAddressSelected]
        [CVAASECUNITDES :: Address.SecondaryUnitDesignator
+DeliverabilityValidated+CurrentAddressSelected]
        [CVAASECNUMBER :: Address.SecondaryNumber +DeliverabilityValidated+CurrentAddressSelected]
        [BUSADDRESS :: Address.Business
+UspsStandardized+DeliverabilityValidated+CurrentAddressSelected]
        [CVAACITYNAME :: Address.City +DeliverabilityValidated+CurrentAddressSelected]
        [CVAASTATEABBREV :: Address.State +DeliverabilityValidated+CurrentAddressSelected]
        [CVAAZIPCODE :: Address.Zip +DeliverabilityValidated+CurrentAddressSelected]
        [CVAAADDONCODE :: Address.Zip4 +DeliverabilityValidated+CurrentAddressSelected]
        [CVAAOUTPRURB :: Address.UrbanizationName +DeliverabilityValidated+CurrentAddressSelected]
        [CVFIRSTNAME1 :: Name.First +NameEditChecked]
        [CVLASTNAME1 :: Name.Last +NameEditChecked]
        [CVPROFTITLE1 :: Name.ProfessionalTitle +IndustryCoded]
      }
    }

    AddressEnhance
    {

      INPUT MAPPINGS
      {
        PRIMARYADDRESSLINE <= [Address.Primary::Address.Primary]
        CITY <= [CVAACITYNAME::Address.City +UspsStandardized]
        STATE <= [CVAASTATEABBREV::Address.State +UspsStandardized]
      }

      OPTION MAPPINGS
      {
        PerformDeliverySequencing <= (N)
        PerformChangeOfAddress <= (N)
        PerformGeocoding <= (None)
      }

      END STATE
      {
        Address.Primary::Address.Primary
        Name.Full::Name.Full
        Address.CityStateZip::Address.CityStateZip
        CVFIRSTNAME1::Name.First
        CVMIDDLENAME1::Name.Middle
        CVLASTNAME1::Name.Last
        CVGENDERCODE1::Name.Gender
```

CVADDRESSTYPE::Address.Type
CVPRIMARYADDRESS::Address.Primary
CVSECONDARYADDRESS::Address.Secondary
CVAAPRINUMBER::Address.PrimaryNumber +UspsStandardized
CVAAPREDIR1::Address.PreDirectional +UspsStandardized
CVAAPRINAME::Address.PrimaryName +UspsStandardized
CVAASUFFIX1::Address.Suffix +UspsStandardized
CVAAPOSTDIR::Address.PostDirectional +UspsStandardized
CVAASECUNITDES::Address.SecondaryUnitDesignator +UspsStandardized
CVAASECNUMBER::Address.SecondaryNumber +UspsStandardized
CVAAOUTPRURB::Address.UrbanizationName +UspsStandardized
CVAACITYNAME::Address.City +UspsStandardized
CVAASTATEABBREV::Address.State +UspsStandardized
CVAAZIPCODE::Address.Zip +UspsStandardized
CVAAADDONCODE::Address.Zip4 +UspsStandardized
CVPROFTITLE1::Name.ProfessionalTitle
CVPRIMARYBUSINESS::Name.Business
CVSECONDARYBUSINESS::Name.Business
CVAABUSNAME::Name.Business +UspsStandardized
AAPRINUMBERBEFORECOA::Address.PrimaryNumber +UspsStandardized +DeliverabilityValidated
AAPREDIR1BEFORECOA::Address.PreDirectional +UspsStandardized +DeliverabilityValidated
AAPRINAMEBEFORECOA::Address.PrimaryName +UspsStandardized +DeliverabilityValidated
AASUFFIX1BEFORECOA::Address.Suffix +UspsStandardized +DeliverabilityValidated
AAPOSTDIR1BEFORECOA::Address.PostDirectional +UspsStandardized +DeliverabilityValidated
AASECUNITDESBEFORECOA::Address.SecondaryUnitDesignator +UspsStandardized +DeliverabilityValidated
AASECNUMBERBEFORECOA::Address.SecondaryNumber +UspsStandardized +DeliverabilityValidated
AAOUTPRURBBEFORECOA::Address.UrbanizationName +UspsStandardized +DeliverabilityValidated
AAOUTBUSADDRESSBEFORECOA::Address.Business +UspsStandardized +DeliverabilityValidated
AACITYNAMEBEFORECOA::Address.City +UspsStandardized +DeliverabilityValidated
AASTATEABREVBEFORECOA::Address.State +UspsStandardized +DeliverabilityValidated
AAZIPCODEBEFORECOA::Address.Zip +UspsStandardized +DeliverabilityValidated
AAADDONCODEBEFORECOA::Address.Zip4 +UspsStandardized +DeliverabilityValidated
AAOUTPRIMADDRESSBEFORECOA::Address.Primary +UspsStandardized +DeliverabilityValidated
AAOUTSECADDRESSBEFORECOA::Address.Secondary +UspsStandardized +DeliverabilityValidated
AAADDRESSFLAGSBEFORECOA::Address.Flags +UspsStandardized
AAPRINUMBERAFTERCOA::Address.PrimaryNumber +UspsStandardized +DeliverabilityValidated
AAPREDIR1AFTERCOA::Address.PreDirectional +UspsStandardized +DeliverabilityValidated
AAPRINAMEAFTERCOA::Address.PrimaryName +UspsStandardized +DeliverabilityValidated
AASUFFIX1AFTERCOA::Address.Suffix +UspsStandardized +DeliverabilityValidated
AAPOSTDIR1AFTERCOA::Address.PostDirectional +UspsStandardized +DeliverabilityValidated
AASECUNITDESAFTERCOA::Address.SecondaryUnitDesignator +UspsStandardized +DeliverabilityValidated
AASECNUMBERAFTERCOA::Address.SecondaryNumber +UspsStandardized +DeliverabilityValidated
AAOUTPRURBAFTERCOA::Address.UrbanizationName +UspsStandardized +DeliverabilityValidated
AAOUTBUSADDRESSAFTERCOA::Address.Business +UspsStandardized +DeliverabilityValidated
AACITYNAMEAFTERCOA::Address.City +UspsStandardized +DeliverabilityValidated
AASTATEABREVAFTERCOA::Address.State +UspsStandardized +DeliverabilityValidated
AAZIPCODEAFTERCOA::Address.Zip +UspsStandardized +DeliverabilityValidated
AAADDONCODEAFTERCOA::Address.Zip4 +UspsStandardized +DeliverabilityValidated
AAOUTPRIMADDRESSAFTERCOA::Address.Primary +UspsStandardized +DeliverabilityValidated
AAOUTSECADDRESSAFTERCOA::Address.Secondary +UspsStandardized +DeliverabilityValidated
AAADDRESSFLAGSAFTERCOA::Address.Flags +UspsStandardized
}

```
  REMAINING GOALS
  {
    [AAOUTPRIMADDRESSAFTERCOA :: Address.Primary +CurrentAddressSelected]
    [AAOUTSECADDRESSAFTERCOA :: Address.Secondary +CurrentAddressSelected]
    [AAPRINUMBERAFTERCOA :: Address.PrimaryNumber +CurrentAddressSelected]
    [AAPREDIR1AFTERCOA :: Address.PreDirectional +CurrentAddressSelected]
    [AAPRINAMEAFTERCOA :: Address.PrimaryName +CurrentAddressSelected]
    [AASUFFIX1AFTERCOA :: Address.Suffix +CurrentAddressSelected]
    [AAPOSTDIR1AFTERCOA :: Address.PostDirectional +CurrentAddressSelected]
    [AASECUNITDESAFTERCOA :: Address.SecondaryUnitDesignator +CurrentAddressSelected]
    [AASECNUMBERAFTERCOA :: Address.SecondaryNumber +CurrentAddressSelected]
    [AAOUTBUSADDRESSAFTERCOA :: Address.Business +CurrentAddressSelected]
    [AACITYNAMEAFTERCOA :: Address.City +CurrentAddressSelected]
    [AASTATEABREVAFTERCOA :: Address.State +CurrentAddressSelected]
    [AAZIPCODEAFTERCOA :: Address.Zip +CurrentAddressSelected]
    [AAADDONCODEAFTERCOA :: Address.Zip4 +CurrentAddressSelected]
    [AAOUTPRURBAFTERCOA :: Address.UrbanizationName +CurrentAddressSelected]
    [CVFIRSTNAME1 :: Name.First +NameEditChecked]
    [CVLASTNAME1 :: Name.Last +NameEditChecked]
    [CVPROFTITLE1 :: Name.ProfessionalTitle +IndustryCoded]
  }
}

NameEditCheck
{

  INPUT MAPPINGS
  {
    GX_FNAME <= [CVFIRSTNAME1::Name.First]
    GX_LNAME <= [CVLASTNAME1::Name.Last]
  }

  OPTION MAPPINGS
  {
  }

  END STATE
  {
    Address.Primary::Address.Primary
    Name.Full::Name.Full
    Address.CityStateZip::Address.CityStateZip
    CVFIRSTNAME1::Name.First
    CVMIDDLENAME1::Name.Middle
    CVLASTNAME1::Name.Last
    CVGENDERCODE1::Name.Gender
    CVADDRESSTYPE::Address.Type
    CVPRIMARYADDRESS::Address.Primary
    CVSECONDARYADDRESS::Address.Secondary
    CVAAPRINUMBER::Address.PrimaryNumber +UspsStandardized
    CVAAPREDIR1::Address.PreDirectional +UspsStandardized
    CVAAPRINAME::Address.PrimaryName +UspsStandardized
    CVAASUFFIX1::Address.Suffix +UspsStandardized
    CVAAPOSTDIR::Address.PostDirectional +UspsStandardized
    CVAASECUNITDES::Address.SecondaryUnitDesignator +UspsStandardized
    CVAASECNUMBER::Address.SecondaryNumber +UspsStandardized
    CVAAOUTPRURB::Address.UrbanizationName +UspsStandardized
```

CVAACITYNAME::Address.City +UspsStandardized
CVAASTATEABBREV::Address.State +UspsStandardized
CVAAZIPCODE::Address.Zip +UspsStandardized
CVAAADDONCODE::Address.Zip4 +UspsStandardized
CVPROFTITLE1::Name.ProfessionalTitle
CVPRIMARYBUSINESS::Name.Business
CVSECONDARYBUSINESS::Name.Business
CVAABUSNAME::Name.Business +UspsStandardized
AAPRINUMBERBEFORECOA::Address.PrimaryNumber +UspsStandardized +DeliverabilityValidated
AAPREDIR1BEFORECOA::Address.PreDirectional +UspsStandardized +DeliverabilityValidated
AAPRINAMEBEFORECOA::Address.PrimaryName +UspsStandardized +DeliverabilityValidated
AASUFFIX1BEFORECOA::Address.Suffix +UspsStandardized +DeliverabilityValidated
AAPOSTDIR1BEFORECOA::Address.PostDirectional +UspsStandardized +DeliverabilityValidated
AASECUNITDESBEFORECOA::Address.SecondaryUnitDesignator +UspsStandardized +DeliverabilityValidated
AASECNUMBERBEFORECOA::Address.SecondaryNumber +UspsStandardized +DeliverabilityValidated
AAOUTPRURBBEFORECOA::Address.UrbanizationName +UspsStandardized +DeliverabilityValidated
AAOUTBUSADDRESSBEFORECOA::Address.Business +UspsStandardized +DeliverabilityValidated
AACITYNAMEBEFORECOA::Address.City +UspsStandardized +DeliverabilityValidated
AASTATEABREVBEFORECOA::Address.State +UspsStandardized +DeliverabilityValidated
AAZIPCODEBEFORECOA::Address.Zip +UspsStandardized +DeliverabilityValidated
AAADDONCODEBEFORECOA::Address.Zip4 +UspsStandardized +DeliverabilityValidated
AAOUTPRIMADDRESSBEFORECOA::Address.Primary +UspsStandardized +DeliverabilityValidated
AAOUTSECADDRESSBEFORECOA::Address.Secondary +UspsStandardized +DeliverabilityValidated
AAADDRESSFLAGSBEFORECOA::Address.Flags +UspsStandardized
AAPRINUMBERAFTERCOA::Address.PrimaryNumber +UspsStandardized +DeliverabilityValidated
AAPREDIR1AFTERCOA::Address.PreDirectional +UspsStandardized +DeliverabilityValidated
AAPRINAMEAFTERCOA::Address.PrimaryName +UspsStandardized +DeliverabilityValidated
AASUFFIX1AFTERCOA::Address.Suffix +UspsStandardized +DeliverabilityValidated
AAPOSTDIR1AFTERCOA::Address.PostDirectional +UspsStandardized +DeliverabilityValidated
AASECUNITDESAFTERCOA::Address.SecondaryUnitDesignator +UspsStandardized +DeliverabilityValidated
AASECNUMBERAFTERCOA::Address.SecondaryNumber +UspsStandardized +DeliverabilityValidated
AAOUTPRURBAFTERCOA::Address.UrbanizationName +UspsStandardized +DeliverabilityValidated
AAOUTBUSADDRESSAFTERCOA::Address.Business +UspsStandardized +DeliverabilityValidated
AACITYNAMEAFTERCOA::Address.City +UspsStandardized +DeliverabilityValidated
AASTATEABREVAFTERCOA::Address.State +UspsStandardized +DeliverabilityValidated
AAZIPCODEAFTERCOA::Address.Zip +UspsStandardized +DeliverabilityValidated
AAADDONCODEAFTERCOA::Address.Zip4 +UspsStandardized +DeliverabilityValidated
AAOUTPRIMADDRESSAFTERCOA::Address.Primary +UspsStandardized +DeliverabilityValidated
AAOUTSECADDRESSAFTERCOA::Address.Secondary +UspsStandardized +DeliverabilityValidated
AAADDRESSFLAGSAFTERCOA::Address.Flags +UspsStandardized
GX_SUGFNAME::Name.First +NameEditChecked
GX_SUGLNAME::Name.Last +NameEditChecked
}

REMAINING GOALS
{
[AAOUTPRIMADDRESSAFTERCOA :: Address.Primary +CurrentAddressSelected]
[AAOUTSECADDRESSAFTERCOA :: Address.Secondary +CurrentAddressSelected]
[AAPRINUMBERAFTERCOA :: Address.PrimaryNumber +CurrentAddressSelected]
[AAPREDIR1AFTERCOA :: Address.PreDirectional +CurrentAddressSelected]
[AAPRINAMEAFTERCOA :: Address.PrimaryName +CurrentAddressSelected]
[AASUFFIX1AFTERCOA :: Address.Suffix +CurrentAddressSelected]
[AAPOSTDIR1AFTERCOA :: Address.PostDirectional +CurrentAddressSelected]

```
        [AASECUNITDESAFTERCOA :: Address.SecondaryUnitDesignator +CurrentAddressSelected]
        [AASECNUMBERAFTERCOA :: Address.SecondaryNumber +CurrentAddressSelected]
        [AAOUTBUSADDRESSAFTERCOA :: Address.Business +CurrentAddressSelected]
        [AACITYNAMEAFTERCOA :: Address.City +CurrentAddressSelected]
        [AASTATEABREVAFTERCOA :: Address.State +CurrentAddressSelected]
        [AAZIPCODEAFTERCOA :: Address.Zip +CurrentAddressSelected]
        [AAADDONCODEAFTERCOA :: Address.Zip4 +CurrentAddressSelected]
        [AAOUTPRURBAFTERCOA :: Address.UrbanizationName +CurrentAddressSelected]
        [CVPROFTITLE1 :: Name.ProfessionalTitle +IndustryCoded]
      }
    }

    AddressSelect
    {

      INPUT MAPPINGS
      {
        oprBUSADDRESS <= [AAOUTBUSADDRESSBEFORECOA::Address.Business +UspsStandardized
+DeliverabilityValidated]
        oprPRINUMBER <= [CVAAPRINUMBER::Address.PrimaryNumber +UspsStandardized]
        oprPREDIR <= [CVAAPREDIR1::Address.PreDirectional +UspsStandardized]
        oprPRINAME <= [CVAAPRINAME::Address.PrimaryName +UspsStandardized]
        oprSUFFIX <= [CVAASUFFIX1::Address.Suffix +UspsStandardized]
        oprPOSTDIR <= [CVAAPOSTDIR::Address.PostDirectional +UspsStandardized]
        oprSECUNITDES <= [CVAASECUNITDES::Address.SecondaryUnitDesignator +UspsStandardized]
        oprSECNUMBER <= [CVAASECNUMBER::Address.SecondaryNumber +UspsStandardized]
        oprCITY <= [CVAACITYNAME::Address.City +UspsStandardized]
        oprSTATE <= [CVAASTATEABBREV::Address.State +UspsStandardized]
        oprZIP <= [CVAAZIPCODE::Address.Zip +UspsStandardized]
        oprZIP4 <= [CVAAADDONCODE::Address.Zip4 +UspsStandardized]
        oprPRIADDRESS <= [AAOUTPRIMADDRESSBEFORECOA::Address.Primary +UspsStandardized
+DeliverabilityValidated]
        oprSECADDRESS <= [AAOUTSECADDRESSBEFORECOA::Address.Secondary +UspsStandardized
+DeliverabilityValidated]
        oprURBNAME <= [CVAAOUTPRURB::Address.UrbanizationName +UspsStandardized]
        oprADDRESSFLAGS <= [AAADDRESSFLAGSBEFORECOA::Address.Flags +UspsStandardized]
      }

      OPTION MAPPINGS
      {
      }

      END STATE
      {
        Address.Primary::Address.Primary
        Name.Full::Name.Full
        Address.CityStateZip::Address.CityStateZip
        CVFIRSTNAME1::Name.First
        CVMIDDLENAME1::Name.Middle
        CVLASTNAME1::Name.Last
        CVGENDERCODE1::Name.Gender
        CVADDRESSTYPE::Address.Type
        CVPRIMARYADDRESS::Address.Primary
        CVSECONDARYADDRESS::Address.Secondary
        CVAAPRINUMBER::Address.PrimaryNumber +UspsStandardized
        CVAAPREDIR1::Address.PreDirectional +UspsStandardized
```

CVAAPRINAME::Address.PrimaryName +UspsStandardized
CVAASUFFIX1::Address.Suffix +UspsStandardized
CVAAPOSTDIR::Address.PostDirectional +UspsStandardized
CVAASECUNITDES::Address.SecondaryUnitDesignator +UspsStandardized
CVAASECNUMBER::Address.SecondaryNumber +UspsStandardized
CVAAOUTPRURB::Address.UrbanizationName +UspsStandardized
CVAACITYNAME::Address.City +UspsStandardized
CVAASTATEABBREV::Address.State +UspsStandardized
CVAAZIPCODE::Address.Zip +UspsStandardized
CVAAADDONCODE::Address.Zip4 +UspsStandardized
CVPROFTITLE1::Name.ProfessionalTitle
CVPRIMARYBUSINESS::Name.Business
CVSECONDARYBUSINESS::Name.Business
CVAABUSNAME::Name.Business +UspsStandardized
AAPRINUMBERBEFORECOA::Address.PrimaryNumber +UspsStandardized +DeliverabilityValidated
AAPREDIR1BEFORECOA::Address.PreDirectional +UspsStandardized +DeliverabilityValidated
AAPRINAMEBEFORECOA::Address.PrimaryName +UspsStandardized +DeliverabilityValidated
AASUFFIX1BEFORECOA::Address.Suffix +UspsStandardized +DeliverabilityValidated
AAPOSTDIR1BEFORECOA::Address.PostDirectional +UspsStandardized +DeliverabilityValidated
AASECUNITDESBEFORECOA::Address.SecondaryUnitDesignator +UspsStandardized +DeliverabilityValidated
AASECNUMBERBEFORECOA::Address.SecondaryNumber +UspsStandardized +DeliverabilityValidated
AAOUTPRURBBEFORECOA::Address.UrbanizationName +UspsStandardized +DeliverabilityValidated
AAOUTBUSADDRESSBEFORECOA::Address.Business +UspsStandardized +DeliverabilityValidated
AACITYNAMEBEFORECOA::Address.City +UspsStandardized +DeliverabilityValidated
AASTATEABREVBEFORECOA::Address.State +UspsStandardized +DeliverabilityValidated
AAZIPCODEBEFORECOA::Address.Zip +UspsStandardized +DeliverabilityValidated
AAADDONCODEBEFORECOA::Address.Zip4 +UspsStandardized +DeliverabilityValidated
AAOUTPRIMADDRESSBEFORECOA::Address.Primary +UspsStandardized +DeliverabilityValidated
AAOUTSECADDRESSBEFORECOA::Address.Secondary +UspsStandardized +DeliverabilityValidated
AAADDRESSFLAGSBEFORECOA::Address.Flags +UspsStandardized
AAPRINUMBERAFTERCOA::Address.PrimaryNumber +UspsStandardized +DeliverabilityValidated
AAPREDIR1AFTERCOA::Address.PreDirectional +UspsStandardized +DeliverabilityValidated
AAPRINAMEAFTERCOA::Address.PrimaryName +UspsStandardized +DeliverabilityValidated
AASUFFIX1AFTERCOA::Address.Suffix +UspsStandardized +DeliverabilityValidated
AAPOSTDIR1AFTERCOA::Address.PostDirectional +UspsStandardized +DeliverabilityValidated
AASECUNITDESAFTERCOA::Address.SecondaryUnitDesignator +UspsStandardized +DeliverabilityValidated
AASECNUMBERAFTERCOA::Address.SecondaryNumber +UspsStandardized +DeliverabilityValidated
AAOUTPRURBAFTERCOA::Address.UrbanizationName +UspsStandardized +DeliverabilityValidated
AAOUTBUSADDRESSAFTERCOA::Address.Business +UspsStandardized +DeliverabilityValidated
AACITYNAMEAFTERCOA::Address.City +UspsStandardized +DeliverabilityValidated
AASTATEABREVAFTERCOA::Address.State +UspsStandardized +DeliverabilityValidated
AAZIPCODEAFTERCOA::Address.Zip +UspsStandardized +DeliverabilityValidated
AAADDONCODEAFTERCOA::Address.Zip4 +UspsStandardized +DeliverabilityValidated
AAOUTPRIMADDRESSAFTERCOA::Address.Primary +UspsStandardized +DeliverabilityValidated
AAOUTSECADDRESSAFTERCOA::Address.Secondary +UspsStandardized +DeliverabilityValidated
AAADDRESSFLAGSAFTERCOA::Address.Flags +UspsStandardized
GX_SUGFNAME::Name.First +NameEditChecked
GX_SUGLNAME::Name.Last +NameEditChecked
PASBUSADDRESS::Address.Business +CurrentAddressSelected
PASPRINUMBER::Address.PrimaryNumber +CurrentAddressSelected
PASPREDIR::Address.PreDirectional +CurrentAddressSelected
PASPRINAME::Address.PrimaryName +CurrentAddressSelected
PASSUFFIX::Address.Suffix +CurrentAddressSelected

```
      PASPOSTDIR::Address.PostDirectional +CurrentAddressSelected
      PASSECUNITDES::Address.SecondaryUnitDesignator +CurrentAddressSelected
      PASSECNUMBER::Address.SecondaryNumber +CurrentAddressSelected
      PASCITYNAME::Address.City +CurrentAddressSelected
      PASSTATEABREV::Address.State +CurrentAddressSelected
      PASZIPCODE::Address.Zip +CurrentAddressSelected
      PASADDONCODE::Address.Zip4 +CurrentAddressSelected
      PASPRIADDRESS::Address.Primary +CurrentAddressSelected
      PASSECADDRESS::Address.Secondary +CurrentAddressSelected
      PASPRURBANIZATION::Address.UrbanizationName +CurrentAddressSelected
      PASADDRESSFLAGS::Address.Flags +CurrentAddressSelected
   }

   REMAINING GOALS
   {
      [CVPROFTITLE1 :: Name.ProfessionalTitle +IndustryCoded]
   }
}

IndustryCode
{

   INPUT MAPPINGS
   {
      PROFESSIONAL_TITLE <= [CVPROFTITLE1::Name.ProfessionalTitle]
   }

   OPTION MAPPINGS
   {
      SUFFIX TITLE <= (N)
      BUSINESS NAME <= (N)
   }

   END STATE
   {
      Address.Primary::Address.Primary
      Name.Full::Name.Full
      Address.CityStateZip::Address.CityStateZip
      CVFIRSTNAME1::Name.First
      CVMIDDLENAME1::Name.Middle
      CVLASTNAME1::Name.Last
      CVGENDERCODE1::Name.Gender
      CVADDRESSTYPE::Address.Type
      CVPRIMARYADDRESS::Address.Primary
      CVSECONDARYADDRESS::Address.Secondary
      CVAAPRINUMBER::Address.PrimaryNumber +UspsStandardized
      CVAAPREDIR1::Address.PreDirectional +UspsStandardized
      CVAAPRINAME::Address.PrimaryName +UspsStandardized
      CVAASUFFIX1::Address.Suffix +UspsStandardized
      CVAAPOSTDIR::Address.PostDirectional +UspsStandardized
      CVAASECUNITDES::Address.SecondaryUnitDesignator +UspsStandardized
      CVAASECNUMBER::Address.SecondaryNumber +UspsStandardized
      CVAAOUTPRURB::Address.UrbanizationName +UspsStandardized
      CVAACITYNAME::Address.City +UspsStandardized
      CVAASTATEABBREV::Address.State +UspsStandardized
      CVAAZIPCODE::Address.Zip +UspsStandardized
```

CVAAADDONCODE::Address.Zip4 +UspsStandardized
CVPROFTITLE1::Name.ProfessionalTitle
CVPRIMARYBUSINESS::Name.Business
CVSECONDARYBUSINESS::Name.Business
CVAABUSNAME::Name.Business +UspsStandardized
AAPRINUMBERBEFORECOA::Address.PrimaryNumber +UspsStandardized +DeliverabilityValidated
AAPREDIR1BEFORECOA::Address.PreDirectional +UspsStandardized +DeliverabilityValidated
AAPRINAMEBEFORECOA::Address.PrimaryName +UspsStandardized +DeliverabilityValidated
AASUFFIX1BEFORECOA::Address.Suffix +UspsStandardized +DeliverabilityValidated
AAPOSTDIR1BEFORECOA::Address.PostDirectional +UspsStandardized +DeliverabilityValidated
AASECUNITDESBEFORECOA::Address.SecondaryUnitDesignator +UspsStandardized +DeliverabilityValidated
AASECNUMBERBEFORECOA::Address.SecondaryNumber +UspsStandardized +DeliverabilityValidated
AAOUTPRURBBEFORECOA::Address.UrbanizationName +UspsStandardized +DeliverabilityValidated
AAOUTBUSADDRESSBEFORECOA::Address.Business +UspsStandardized +DeliverabilityValidated
AACITYNAMEBEFORECOA::Address.City +UspsStandardized +DeliverabilityValidated
AASTATEABREVBEFORECOA::Address.State +UspsStandardized +DeliverabilityValidated
AAZIPCODEBEFORECOA::Address.Zip +UspsStandardized +DeliverabilityValidated
AAADDONCODEBEFORECOA::Address.Zip4 +UspsStandardized +DeliverabilityValidated
AAOUTPRIMADDRESSBEFORECOA::Address.Primary +UspsStandardized +DeliverabilityValidated
AAOUTSECADDRESSBEFORECOA::Address.Secondary +UspsStandardized +DeliverabilityValidated
AAADDRESSFLAGSBEFORECOA::Address.Flags +UspsStandardized
AAPRINUMBERAFTERCOA::Address.PrimaryNumber +UspsStandardized +DeliverabilityValidated
AAPREDIR1AFTERCOA::Address.PreDirectional +UspsStandardized +DeliverabilityValidated
AAPRINAMEAFTERCOA::Address.PrimaryName +UspsStandardized +DeliverabilityValidated
AASUFFIX1AFTERCOA::Address.Suffix +UspsStandardized +DeliverabilityValidated
AAPOSTDIR1AFTERCOA::Address.PostDirectional +UspsStandardized +DeliverabilityValidated
AASECUNITDESAFTERCOA::Address.SecondaryUnitDesignator +UspsStandardized +DeliverabilityValidated
AASECNUMBERAFTERCOA::Address.SecondaryNumber +UspsStandardized +DeliverabilityValidated
AAOUTPRURBAFTERCOA::Address.UrbanizationName +UspsStandardized +DeliverabilityValidated
AAOUTBUSADDRESSAFTERCOA::Address.Business +UspsStandardized +DeliverabilityValidated
AACITYNAMEAFTERCOA::Address.City +UspsStandardized +DeliverabilityValidated
AASTATEABREVAFTERCOA::Address.State +UspsStandardized +DeliverabilityValidated
AAZIPCODEAFTERCOA::Address.Zip +UspsStandardized +DeliverabilityValidated
AAADDONCODEAFTERCOA::Address.Zip4 +UspsStandardized +DeliverabilityValidated
AAOUTPRIMADDRESSAFTERCOA::Address.Primary +UspsStandardized +DeliverabilityValidated
AAOUTSECADDRESSAFTERCOA::Address.Secondary +UspsStandardized +DeliverabilityValidated
AAADDRESSFLAGSAFTERCOA::Address.Flags +UspsStandardized
GX_SUGFNAME::Name.First +NameEditChecked
GX_SUGLNAME::Name.Last +NameEditChecked
PASBUSADDRESS::Address.Business +CurrentAddressSelected
PASPRINUMBER::Address.PrimaryNumber +CurrentAddressSelected
PASPREDIR::Address.PreDirectional +CurrentAddressSelected
PASPRINAME::Address.PrimaryName +CurrentAddressSelected
PASSUFFIX::Address.Suffix +CurrentAddressSelected
PASPOSTDIR::Address.PostDirectional +CurrentAddressSelected
PASSECUNITDES::Address.SecondaryUnitDesignator +CurrentAddressSelected
PASSECNUMBER::Address.SecondaryNumber +CurrentAddressSelected
PASCITYNAME::Address.City +CurrentAddressSelected
PASSTATEABREV::Address.State +CurrentAddressSelected
PASZIPCODE::Address.Zip +CurrentAddressSelected
PASADDONCODE::Address.Zip4 +CurrentAddressSelected
PASPRIADDRESS::Address.Primary +CurrentAddressSelected
PASSECADDRESS::Address.Secondary +CurrentAddressSelected

```
            PASPRURBANIZATION::Address.UrbanizationName +CurrentAddressSelected
            PASADDRESSFLAGS::Address.Flags +CurrentAddressSelected
            SIMNEWTITLE::Name.IndustryTitle
            SIMTITLECODE::Name.IndustryTitleCode
            SIMFUNCTION1CODE::Name.IndustryFunctionCode
            PROFESSIONAL_TITLE::Name.ProfessionalTitle +IndustryCoded
        }

        REMAINING GOALS
        {
        }
    }
  }
}
```

C.2 Testing Scenario 2

C.2.1 Distinct Sequences

1. Parser => AddressEnhance => AddressEditCheck => NameEditCheck => AddressSelect => ContactLink => IndustryCode
2. Parser => AddressEnhance => AddressEditCheck => NameEditCheck => AddressSelect => IndustryCode => ContactLink
3. Parser => AddressEnhance => AddressEditCheck => NameEditCheck => IndustryCode => AddressSelect => ContactLink
4. Parser => AddressEnhance => AddressEditCheck => AddressSelect => ContactLink => NameEditCheck => IndustryCode
5. Parser => AddressEnhance => AddressEditCheck => AddressSelect => ContactLink => IndustryCode => NameEditCheck
6. Parser => AddressEnhance => AddressEditCheck => AddressSelect => NameEditCheck => ContactLink => IndustryCode
7. Parser => AddressEnhance => AddressEditCheck => AddressSelect => NameEditCheck => IndustryCode => ContactLink
8. Parser => AddressEnhance => AddressEditCheck => AddressSelect => IndustryCode => ContactLink => NameEditCheck
9. Parser => AddressEnhance => AddressEditCheck => AddressSelect => IndustryCode => NameEditCheck => ContactLink
10. Parser => AddressEnhance => AddressEditCheck => IndustryCode => NameEditCheck => AddressSelect => ContactLink
11. Parser => AddressEnhance => AddressEditCheck => IndustryCode => AddressSelect => ContactLink => NameEditCheck
12. Parser => AddressEnhance => AddressEditCheck => IndustryCode => AddressSelect => NameEditCheck => ContactLink
13. Parser => AddressEnhance => NameEditCheck => AddressEditCheck => AddressSelect => ContactLink => IndustryCode
14. Parser => AddressEnhance => NameEditCheck => AddressEditCheck => AddressSelect => IndustryCode => ContactLink
15. Parser => AddressEnhance => NameEditCheck => AddressEditCheck => IndustryCode => AddressSelect => ContactLink
16. Parser => AddressEnhance => NameEditCheck => AddressSelect => ContactLink => AddressEditCheck => IndustryCode
17. Parser => AddressEnhance => NameEditCheck => AddressSelect => ContactLink => IndustryCode => AddressEditCheck
18. Parser => AddressEnhance => NameEditCheck => AddressSelect => AddressEditCheck => ContactLink => IndustryCode
19. Parser => AddressEnhance => NameEditCheck => AddressSelect => AddressEditCheck => IndustryCode => ContactLink
20. Parser => AddressEnhance => NameEditCheck => AddressSelect => IndustryCode => ContactLink => AddressEditCheck
21. Parser => AddressEnhance => NameEditCheck => AddressSelect => IndustryCode => AddressEditCheck => ContactLink
22. Parser => AddressEnhance => NameEditCheck => IndustryCode => AddressEditCheck => AddressSelect => ContactLink
23. Parser => AddressEnhance => NameEditCheck => IndustryCode => AddressSelect => ContactLink => AddressEditCheck
24. Parser => AddressEnhance => NameEditCheck => IndustryCode => AddressSelect => AddressEditCheck => ContactLink

25. Parser => AddressEnhance => AddressSelect => ContactLink => AddressEditCheck => NameEditCheck => IndustryCode
26. Parser => AddressEnhance => AddressSelect => ContactLink => AddressEditCheck => IndustryCode => NameEditCheck
27. Parser => AddressEnhance => AddressSelect => ContactLink => NameEditCheck => AddressEditCheck => IndustryCode
28. Parser => AddressEnhance => AddressSelect => ContactLink => NameEditCheck => IndustryCode => AddressEditCheck
29. Parser => AddressEnhance => AddressSelect => ContactLink => IndustryCode => AddressEditCheck => NameEditCheck
30. Parser => AddressEnhance => AddressSelect => ContactLink => IndustryCode => NameEditCheck => AddressEditCheck
31. Parser => AddressEnhance => AddressSelect => AddressEditCheck => ContactLink => NameEditCheck => IndustryCode
32. Parser => AddressEnhance => AddressSelect => AddressEditCheck => ContactLink => IndustryCode => NameEditCheck
33. Parser => AddressEnhance => AddressSelect => AddressEditCheck => NameEditCheck => ContactLink => IndustryCode
34. Parser => AddressEnhance => AddressSelect => AddressEditCheck => NameEditCheck => IndustryCode => ContactLink
35. Parser => AddressEnhance => AddressSelect => AddressEditCheck => IndustryCode => ContactLink => NameEditCheck
36. Parser => AddressEnhance => AddressSelect => AddressEditCheck => IndustryCode => NameEditCheck => ContactLink
37. Parser => AddressEnhance => AddressSelect => NameEditCheck => ContactLink => AddressEditCheck => IndustryCode
38. Parser => AddressEnhance => AddressSelect => NameEditCheck => ContactLink => IndustryCode => AddressEditCheck
39. Parser => AddressEnhance => AddressSelect => NameEditCheck => AddressEditCheck => ContactLink => IndustryCode
40. Parser => AddressEnhance => AddressSelect => NameEditCheck => AddressEditCheck => IndustryCode => ContactLink
41. Parser => AddressEnhance => AddressSelect => NameEditCheck => IndustryCode => ContactLink => AddressEditCheck
42. Parser => AddressEnhance => AddressSelect => NameEditCheck => IndustryCode => AddressEditCheck => ContactLink
43. Parser => AddressEnhance => AddressSelect => IndustryCode => ContactLink => AddressEditCheck => NameEditCheck
44. Parser => AddressEnhance => AddressSelect => IndustryCode => ContactLink => NameEditCheck => AddressEditCheck
45. Parser => AddressEnhance => AddressSelect => IndustryCode => AddressEditCheck => ContactLink => NameEditCheck
46. Parser => AddressEnhance => AddressSelect => IndustryCode => AddressEditCheck => NameEditCheck => ContactLink
47. Parser => AddressEnhance => AddressSelect => IndustryCode => NameEditCheck => ContactLink => AddressEditCheck
48. Parser => AddressEnhance => AddressSelect => IndustryCode => NameEditCheck => AddressEditCheck => ContactLink
49. Parser => AddressEnhance => IndustryCode => AddressEditCheck => NameEditCheck => AddressSelect => ContactLink
50. Parser => AddressEnhance => IndustryCode => AddressEditCheck => AddressSelect => ContactLink => NameEditCheck
51. Parser => AddressEnhance => IndustryCode => AddressEditCheck => AddressSelect => NameEditCheck => ContactLink
52. Parser => AddressEnhance => IndustryCode => NameEditCheck => AddressEditCheck => AddressSelect => ContactLink

53. Parser => AddressEnhance => IndustryCode => NameEditCheck => AddressSelect => ContactLink => AddressEditCheck
54. Parser => AddressEnhance => IndustryCode => NameEditCheck => AddressSelect => AddressEditCheck => ContactLink
55. Parser => AddressEnhance => IndustryCode => AddressSelect => ContactLink => AddressEditCheck => NameEditCheck
56. Parser => AddressEnhance => IndustryCode => AddressSelect => ContactLink => NameEditCheck => AddressEditCheck
57. Parser => AddressEnhance => IndustryCode => AddressSelect => AddressEditCheck => ContactLink => NameEditCheck
58. Parser => AddressEnhance => IndustryCode => AddressSelect => AddressEditCheck => NameEditCheck => ContactLink
59. Parser => AddressEnhance => IndustryCode => AddressSelect => NameEditCheck => ContactLink => AddressEditCheck
60. Parser => AddressEnhance => IndustryCode => AddressSelect => NameEditCheck => AddressEditCheck => ContactLink
61. Parser => AddressEditCheck => AddressEnhance => NameEditCheck => AddressSelect => ContactLink => IndustryCode
62. Parser => AddressEditCheck => AddressEnhance => NameEditCheck => AddressSelect => IndustryCode => ContactLink
63. Parser => AddressEditCheck => AddressEnhance => NameEditCheck => IndustryCode => AddressSelect => ContactLink
64. Parser => AddressEditCheck => AddressEnhance => AddressSelect => ContactLink => NameEditCheck => IndustryCode
65. Parser => AddressEditCheck => AddressEnhance => AddressSelect => ContactLink => IndustryCode => NameEditCheck
66. Parser => AddressEditCheck => AddressEnhance => AddressSelect => NameEditCheck => ContactLink => IndustryCode
67. Parser => AddressEditCheck => AddressEnhance => AddressSelect => NameEditCheck => IndustryCode => ContactLink
68. Parser => AddressEditCheck => AddressEnhance => AddressSelect => IndustryCode => ContactLink => NameEditCheck
69. Parser => AddressEditCheck => AddressEnhance => AddressSelect => IndustryCode => NameEditCheck => ContactLink
70. Parser => AddressEditCheck => AddressEnhance => IndustryCode => NameEditCheck => AddressSelect => ContactLink
71. Parser => AddressEditCheck => AddressEnhance => IndustryCode => AddressSelect => ContactLink => NameEditCheck
72. Parser => AddressEditCheck => AddressEnhance => IndustryCode => AddressSelect => NameEditCheck => ContactLink
73. Parser => AddressEditCheck => NameEditCheck => AddressEnhance => AddressSelect => ContactLink => IndustryCode
74. Parser => AddressEditCheck => NameEditCheck => AddressEnhance => AddressSelect => IndustryCode => ContactLink
75. Parser => AddressEditCheck => NameEditCheck => AddressEnhance => IndustryCode => AddressSelect => ContactLink
76. Parser => AddressEditCheck => NameEditCheck => IndustryCode => AddressEnhance => AddressSelect => ContactLink
77. Parser => AddressEditCheck => IndustryCode => AddressEnhance => NameEditCheck => AddressSelect => ContactLink
78. Parser => AddressEditCheck => IndustryCode => AddressEnhance => AddressSelect => ContactLink => NameEditCheck
79. Parser => AddressEditCheck => IndustryCode => AddressEnhance => AddressSelect => NameEditCheck => ContactLink
80. Parser => AddressEditCheck => IndustryCode => NameEditCheck => AddressEnhance => AddressSelect => ContactLink

81. Parser => NameEditCheck => AddressEnhance => AddressEditCheck => AddressSelect => ContactLink => IndustryCode
82. Parser => NameEditCheck => AddressEnhance => AddressEditCheck => AddressSelect => IndustryCode => ContactLink
83. Parser => NameEditCheck => AddressEnhance => AddressEditCheck => IndustryCode => AddressSelect => ContactLink
84. Parser => NameEditCheck => AddressEnhance => AddressSelect => ContactLink => AddressEditCheck => IndustryCode
85. Parser => NameEditCheck => AddressEnhance => AddressSelect => ContactLink => IndustryCode => AddressEditCheck
86. Parser => NameEditCheck => AddressEnhance => AddressSelect => AddressEditCheck => ContactLink => IndustryCode
87. Parser => NameEditCheck => AddressEnhance => AddressSelect => AddressEditCheck => IndustryCode => ContactLink
88. Parser => NameEditCheck => AddressEnhance => AddressSelect => IndustryCode => ContactLink => AddressEditCheck
89. Parser => NameEditCheck => AddressEnhance => AddressSelect => IndustryCode => AddressEditCheck => ContactLink
90. Parser => NameEditCheck => AddressEnhance => IndustryCode => AddressEditCheck => AddressSelect => ContactLink
91. Parser => NameEditCheck => AddressEnhance => IndustryCode => AddressSelect => ContactLink => AddressEditCheck
92. Parser => NameEditCheck => AddressEnhance => IndustryCode => AddressSelect => AddressEditCheck => ContactLink
93. Parser => NameEditCheck => AddressEditCheck => AddressEnhance => AddressSelect => ContactLink => IndustryCode
94. Parser => NameEditCheck => AddressEditCheck => AddressEnhance => AddressSelect => IndustryCode => ContactLink
95. Parser => NameEditCheck => AddressEditCheck => AddressEnhance => IndustryCode => AddressSelect => ContactLink
96. Parser => NameEditCheck => AddressEditCheck => IndustryCode => AddressEnhance => AddressSelect => ContactLink
97. Parser => NameEditCheck => IndustryCode => AddressEnhance => AddressEditCheck => AddressSelect => ContactLink
98. Parser => NameEditCheck => IndustryCode => AddressEnhance => AddressSelect => ContactLink => AddressEditCheck
99. Parser => NameEditCheck => IndustryCode => AddressEnhance => AddressSelect => AddressEditCheck => ContactLink
100. Parser => NameEditCheck => IndustryCode => AddressEditCheck => AddressEnhance => AddressSelect => ContactLink
101. Parser => IndustryCode => AddressEnhance => AddressEditCheck => NameEditCheck => AddressSelect => ContactLink
102. Parser => IndustryCode => AddressEnhance => AddressEditCheck => AddressSelect => ContactLink => NameEditCheck
103. Parser => IndustryCode => AddressEnhance => AddressEditCheck => AddressSelect => NameEditCheck => ContactLink
104. Parser => IndustryCode => AddressEnhance => NameEditCheck => AddressEditCheck => AddressSelect => ContactLink
105. Parser => IndustryCode => AddressEnhance => NameEditCheck => AddressSelect => ContactLink => AddressEditCheck
106. Parser => IndustryCode => AddressEnhance => NameEditCheck => AddressSelect => AddressEditCheck => ContactLink
107. Parser => IndustryCode => AddressEnhance => AddressSelect => ContactLink => AddressEditCheck => NameEditCheck
108. Parser => IndustryCode => AddressEnhance => AddressSelect => ContactLink => NameEditCheck => AddressEditCheck

109. Parser => IndustryCode => AddressEnhance => AddressSelect => AddressEditCheck => ContactLink => NameEditCheck
110. Parser => IndustryCode => AddressEnhance => AddressSelect => AddressEditCheck => NameEditCheck => ContactLink
111. Parser => IndustryCode => AddressEnhance => AddressSelect => NameEditCheck => ContactLink => AddressEditCheck
112. Parser => IndustryCode => AddressEnhance => AddressSelect => NameEditCheck => AddressEditCheck => ContactLink
113. Parser => IndustryCode => AddressEditCheck => AddressEnhance => NameEditCheck => AddressSelect => ContactLink
114. Parser => IndustryCode => AddressEditCheck => AddressEnhance => AddressSelect => ContactLink => NameEditCheck
115. Parser => IndustryCode => AddressEditCheck => AddressEnhance => AddressSelect => NameEditCheck => ContactLink
116. Parser => IndustryCode => AddressEditCheck => NameEditCheck => AddressEnhance => AddressSelect => ContactLink
117. Parser => IndustryCode => NameEditCheck => AddressEnhance => AddressEditCheck => AddressSelect => ContactLink
118. Parser => IndustryCode => NameEditCheck => AddressEnhance => AddressSelect => ContactLink => AddressEditCheck
119. Parser => IndustryCode => NameEditCheck => AddressEnhance => AddressSelect => AddressEditCheck => ContactLink
120. Parser => IndustryCode => NameEditCheck => AddressEditCheck => AddressEnhance => AddressSelect => ContactLink