AN AI PLANNING APPROACH FOR GENERATING BIG DATA WORKFLOWS

Wesley Deneke[1], Wing-Ning Li[2], and Craig Thompson[2]

[1]Computer Science and Industrial Technology Department, Southeastern Louisiana University, Hammond, LA, USA
[2]Computer Science and Computer Engineering Department, University of Arkansas, Fayetteville, AR, USA

ABSTRACT

The scale of big data causes the compositions of extract-transform-load (ETL) workflows to grow increasingly complex. With the turnaround time for delivering solutions becoming a greater emphasis, stakeholders cannot continue to afford to wait the hundreds of hours it takes for domain experts to manually compose a workflow solution. This paper describes a novel AI planning approach that facilitates rapid composition and maintenance of ETL workflows. The workflow engine is evaluated on real-world scenarios from an industrial partner and results gathered from a prototype are reported to demonstrate the validity of the approach.

KEYWORDS

Workflows, Big Data, Extract-Transform-Load (ETL), AI Planning

1. INTRODUCTION

Workflows are increasingly important in the "big data" area of many scientific and industrial problem domains. With the advent of the big data phenomenon, greater quantities of data are being collected and shared [1][2][3]. Modern organizations have data continuously flowing into their systems and seek solutions to continually re-evaluate this "big data" for real time decisions. Through correctly applied data science techniques that correlate, filter, and transform collections of data, these organizations can convert their raw data into knowledge that can be used to mine actionable information.

As described by Deneke et al [4], traditional interfaces for workflow specification are manual, time intensive and cumbersome (e.g. XML), requiring a whole new team of human users with considerable domain knowledge to build, verify, and maintain solutions. The data is typically fragmented, distributed among many different systems with differing schemas, quality, and stored using different technologies. Further, the data often arrives raw; a mix of unstructured and multi-structured data of poor quality that must be standardized and cleansed before it can be integrated and utilized. As a consequence, manually building these workflows becomes a heavy investment of time and resources.

The objective of our research is to address this rapidly rising cost by showing that ETL workflow specification can be automated in an extensible manner through a domain-specific modeling language. In order to effectively supplant the manual approach, an automated solution must provide: better solution accuracy, lower required level of expertise, faster turnaround and fewer

DOI : 10.5121/ijaia.2015.6501

errors. Ultimately, our aim is to abstract the specification process, instead providing an intuitive interface that flexibly integrates appropriate domain knowledge to enable even casual users to generate the correct ETL workflow for their goals by simply stating their "intents".

Our earlier work [5] presented preliminary results, but only provided an overview of two of the key components of our approach: the knowledge representation and intent language. This paper presents the details of the final component of our work: the workflow engine. The workflow engine is an implementation of an AI planner that leverages the domain model to automatically generate ETL workflows that satisfy the goals expressed through the intent language. The central contribution of this research is generalizing database query languages to cover domain-specific operators and developing a way to automatically map an intent query language to a workflow plan. This enables users to focus on the goal statement (what they want to achieve) instead of how to achieve the result.

The organization of the rest of this paper is as follows. Section 2 provides background and key concepts associated with workflow, ETL, and how they relate to artificial intelligence planning and database query languages. Section 3 presents a summary of our previous work and introduces the AI planning approach employed by our workflow engine. Section 4 describes the testing methodology, experimental results obtained, and evaluation of these results. Section 5 contrasts our approach with existing work in related disciplines. Finally, Section 6 summarizes the conclusions and outlines areas of future research.

2. BACKGROUND

2.1. Workflow

Workflows play a central role in delivering ETL solutions, but are also used in a variety of other areas, from AI planning and query optimization to web service composition and scientific data processing. As defined by Aalst and Hee [6], a workflow models a process as a series of tasks that must be performed to achieve a particular result. This model is commonly represented as a directed graph (digraph). A directed graph G is defined as 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. In this representation, vertices denote the tasks and edges specify the sequence in which these tasks are performed.

A workflow can model a process at different levels of abstraction. A workflow's tasks can describe higher-level behavior or very basic actions. This is analogous to programming languages, where high-level languages (e.g. C#) are easier to understand than low-level languages (e.g. Assembly). While the abstraction of underlying actions makes high-level languages easier to comprehend, this also provides users less control than low-level languages. To illustrate,

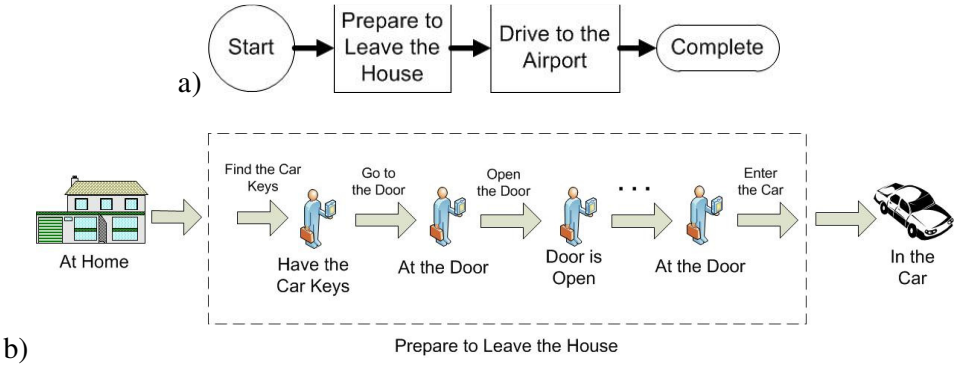

Figure 1. Workflow at: a) higher level of abstraction b) lower level of abstraction.

consider the work performed by a person traveling from their home to the airport. Using high-level descriptions, this process would consist of a task for preparing to leave the house followed by a task for driving to the airport (see Figure 1a). While adding complexity, these tasks could be broken down to describe the process in greater detail, such as in Figure 1b.

While a digraph can represent the composition of a workflow, this basic representation obscures behavioral details, such as why the tasks must occur in this sequence. Instead, the result produced by a data science workflow can be specified as a state: a specific set of conditions. Mapped as a search space problem [7], workflow composition involves a sequence of state transitions (imposed on the data by the operators) to attain the desired set of conditions (goal state). This stateful mapping can be represented with a specific form of a digraph, known as a state diagram. A state diagram S is a directed graph that can be formally defined as: $S = (Q, \sum, \delta, q_0, F)$. In this definition, Q is a collection of states, consisting of the start state q_0 and each state q_i depicted by a vertex of the graph. F is a subset of Q that represents the set of accepting states. $\sum$ is a collection of valid input symbols, also known as an alphabet. Finally, δ is a collection of state transitions, where a state transition δ_i for the input symbol $\sum_t$ from state Q_j to Q_k is expressed as $\delta_i : \sum_t \times Q_j \rightarrow Q_k$, and each state transition δ_i is depicted by an edge of the graph connecting the two states for which there exists a transition. In the context of workflow, the vertices represent a workflow's distinct states and the edges represent the transitions between states dictated by the available tasks.

To demonstrate this concept, consider the workflow depicted in Figure 2. For this example, assume that the person starts at home and the goal is to end up at the airport. To achieve this goal, the sequence of tasks performed in the workflow must transition the person from the initial "At Home" state to the desired "At the Airport" state. The set of distinct states Q for this state diagram representation can be defined as: $Q = \{Q_1, Q_2, Q_3\}$, where Q_1="At Home", Q_2="In the Car", and Q_3="At the Airport". Similarly, the set of available tasks $\sum$ can be defined as: $\sum = \{t_1, t_2\}$, where t_1="Prepare to Leave the House" and t_2="Drive to the Airport". The set of state transitions δ can be defined as: $\delta = \{\delta_1, \delta_2\}$, where $\delta_1 = Q_1 \rightarrow Q_2$ and $\delta_2 = Q_2 \rightarrow Q_3$, the start state $q_0 = Q_1$. Finally, the set accepting states F can be defined as: $F = \{Q_3\}$.

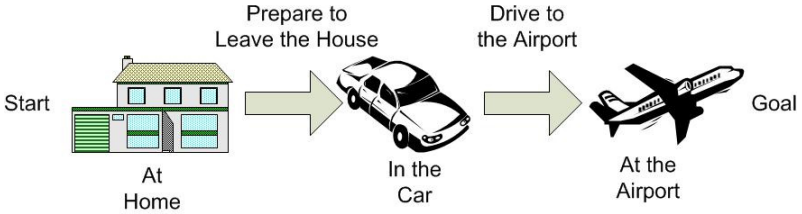

Figure 2. A state diagram for a person traveling to the airport.

2.2. Extract-Transform-Load Workflows

Extract-Transform-Load (ETL) refers to a category of data processing solutions that are designed and executed as workflows to perform a series of extract, transform, and load operations on one or more data sets. A data set is a collection of data records with a uniform schema. The schema defines the data fields – elements that contain the instances of data – that data records are comprised of. Each data field is assigned a data type and a name. The data type is a domain-generic description of the values a data field can contain. The name is a tag – unique within a given schema – that enables each data field to be individually addressed. This structure of data is analogous to that of a relation in a relational database, where a relation contains tuples, which in turn contains attributes.

The data processing operations performed in an ETL workflow can be described as follows:

- Extract – Pull data sets from diverse, possibly numerous sources together into a form of intermediate storage, providing a homogenous format for later operations.
- Transform – Alters data sets to bring data of heterogeneous quality levels to a particular standard and combined, while enabling erroneous, unusable data to be pruned out.
- Load – Persist data that has attained desired characteristics to target repositories, 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.

The procedural logic and environmental details (client, server, grid, etc.) of these operations is encapsulated in reusable modules known as operators. As depicted in Figure 3, each operator contains a set of input fields and options, specified as inputs preceding execution of a workflow, and a set of output fields that are produced after execution. Input fields specify which data fields from a given data set an operator will process. Options are externalized settings that configure the exact behavior of an operator, which can alter the inputs an operator accepts or the outputs an operator produces. Output fields specify data fields that are produced or modified by operator execution.

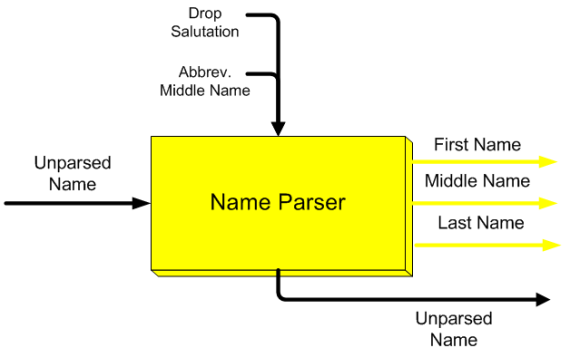

Figure 3. A simple ETL operator.

It is important to note that the graphs of ETL workflows are acyclic, and so are more accurately represented 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.

2.3. Building an ETL Workflow

In practice, ETL workflows are built manually. This process starts by gathering objectives from stakeholders. A stakeholder is a person or organization, such an external client or a business user, that understands and can describe the high-level business goals. Stakeholders often lack the expertise needed to describe the requirements from a technical perspective. To bridge this gap, domain experts must derive the technical requirements from the high-level objectives.

Domain experts are people with extensive domain knowledge, including familiarity with an organization's data and the details of domain-specific operators. To build an appropriate solution, domain experts must determine the following requirements: data sources to include, fields to

extract from these sources, operators to include or exclude, field mappings, option mappings, sequencing of the included operators, what fields to persist, and where to load the data for persistence. These decisions can rely heavily upon domain knowledge. Numerous data fields may appear to be valid arguments for input fields; some from the outputs of previous operators and others from data sources. Further, several operators may perform similar operations; 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. Operator implementation can restrict the quality or format of input data, or conditionally constrain the combination of inputs. Failure to conform to these assumptions can cause the operator to error off or produce invalid results. A workflow's composition may similarly be subject to standards that have been put in place to ensure solution consistency, as well as business rules that consider compliance with regulations or a domain's preferred practices. Once these questions are answered, the workflow graph, such as depicted in Figure 4, can be constructed. Construction of the workflow graph is commonly performed using a visual

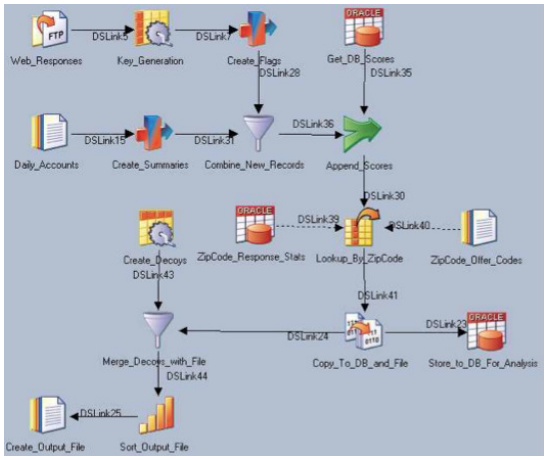

Figure 4. An ETL workflow.

ETL tool. Commercial ETL tools available today include: IBM InfoSphere DataStage [8], Microsoft SQL Server Integration Service [9], and Oracle Data Integrator [10].

Before a workflow is ready for production use it must undergo verification, where it is methodically tested on sample data sets to ensure the workflow behaves according to the specified requirements and verify absence of operator failures at run time from improper mappings. Results produced by execution on sample data sets are evaluated to identify issues such as: ambiguous or missing specifications, contradictory objectives, invalid input mappings, or erroneous operator sequencing. Based on the evaluation, a workflow may need to be reconstructed to resolve compositional errors or even revert back to the analysis phase to clarify requirements.

2.4. SQL and Relational Algebra

Structured Query Language (SQL) is a declarative language, enabling those familiar with the domain of relational databases to make clear, expressive statements in without needing to specify sequences of relational operations or have knowledge of computer systems [11]. A user specifies the properties of the data to be retrieved, not the algorithm required to retrieve the data. Statements in SQL contain constructs known as operators, which perform a task involving one or more relations and produce another relation as a result. Common SQL operators include select, project, union, join, etc.

Before execution, SQL statements are parsed and translated into an intermediate procedural language known as relational algebra, which is utilized by the query optimizer to find the fastest executing expression for a given SQL statement [12]. Of particular interest is that the intermediate form that high-level SQL statements are translated into is a kind of workflow, known as a query plan. As in ETL workflow, operators in a relational query plan can be generic or custom operations, such as relational algebra operators or user-defined functions respectively. Similarly, the operands of these operators can be concrete or virtual relational tables, such as data sources or outputs from earlier operations respectively.

2.5. Search and Planning

Search and planning are general approaches from artificial intelligence for automating problem solving that are also complementary to automatic workflow generation. Similar to the earlier observation that a workflow can be represented as a state diagram, search space concepts can be mapped to workflow. As described by Coppin [7], the search space problem requires three basic pieces of information: an initial state, a goal state, and the collection of available state transitions. For a given problem, each choice can be represented as a state in a state-space diagram and would include an initial state to indicate where the problem starts, and one or more final states to denote the goals. A state transition represents making a choice, which changes the current state in the state-space diagram to another state. Paths through the diagram that transition from the initial state to the final states can then express problem solutions.

Constructing a workflow is a matter of constraint satisfaction (satisfying business rules, pre/post conditions, pre/post requisites, etc.). Given constraint specifications, such as from a knowledge representation, AI planning concepts can be employed to search for a solution to relevant constraints and generate a workflow. The process of planning searches for a solution, determining which actions should be performed to transition the current state closer to the goal. STRIPS [13] is an example of an AI planner that employs an automated planning approach. The approach follows a means-end analysis strategy, determining the differences between the current state and the goal state, then selecting an appropriate action to move closer to the goal state. A set of operators, each with preconditions and postconditions, represent the actions that can be taken. From predicate logic [14], a precondition is a constraint that must be satisfied before a particular action can be performed, while a postcondition is the "effect" of performing the action; what becomes true and no longer true after performing the action.

An instance of STRIPS starts off given the following information: an initial state, described by a set of true conditions; a set of conditions that describe the world; the set of available operators; and a set of conditions that describe the goal state. The STRIPS program then develops a plan by comparing the current world model to the conditions of the goal state. If these conditions match, the goal state is reached and the problem is solved. Otherwise, an operator that lessens the differences between the current state and the goal state is selected and added to the plan. Once an acceptable plan is developed, the plan can be executed by the STRIPS robot and the goal conditions should be satisfied.

3. APPROACH

3.1. Summary of Previous Work

3.1.1. Attributed Field Model

The attributed field model was introduced to express the characteristics of data, augmenting data fields with two semantic annotations: content types and state attributes. These annotations are semantic descriptions of concepts drawn from a domain's ontology. The first annotation, content

type, is a class-like abstraction over the values that data fields contain. Like data type, a given data field can only be assigned a single content type. Unlike data type, a content type is a semantic, user-defined descriptor instead of a technical, system-defined one. Content type describes the data the field contains using an appropriate domain concept, which would be a noun such as "first name", "phone number", "zip code", etc. This key difference enables the specification of relationships between data to be explicit and clear.

The other annotations, state attributes, are used to express data state. A state attribute describes a distinguishable characteristic that instances of data may possess (or lack). Each state attribute assigned to a data field, which can be zero to many, asserts that the values of the given data field possess (or lack) the described characteristic. The composite of these characteristics express the state of the data, which enables variations of data to be differentiated from one another. For example, variations of the same address can be misspelled, missing components, USPS standardized, etc.

3.1.2. Operator Model

The operator model introduced predicate logic to express which input mappings are valid and express the characteristics of the outputs produced for a given input mapping. Operator preconditions assert each requirement an input mapping must satisfy to be considered valid. When the logical disjunction of all of an operator's preconditions evaluates to true, meaning at least one precondition was not violated, an input mapping is valid. Operator postconditions assert which output fields will be written, the characteristics of newly produced data, and the changes to characteristics of derived data. Each postcondition is associated to the precondition that defines the input requirements that guarantee the assertions of the postcondition.

3.1.3. Intent Language

The intent language maps terms to sets of assertions on semantic annotation pairs; a content type and a state attribute per pair. Each assertion specifies a state attribute that must be present (or not present depending on the assertion being inclusive or exclusive) on a data field with the specified content type. This mapping establishes the higher-level terms of the language as abstractions that capture "design intent". Each constituent term of a statement maps to lower-level concepts, providing a partial to full description of the desired data results. Using these mappings, a statement can be translated into technical specifications, which can be merged to produce a single specification of the desired result.

3.2. Infrastructure

Before describing the AI planning approach, it is first necessary to present and explain fundamental details of the infrastructure. First is the concept of state. The state of data fields is represented following the attributed field model, via the state attributes. For workflow, the state of all of the fields in the workflow at a particular instance describes the state, represented by a collection of attributed fields. For application to goals, however, state must be able to be specified in an inclusive manner (data state that should be produced) and an exclusive manner (data state that should not be produced). Accordingly, postconditions represent goal state specifications, which are explained in greater detail later in this section.

Next is the definition of assertion types. Both preconditions and postconditions are composed of assertions, yet the base concept of an assertion is abstract. That is, an assertion's definition can change depending on the criteria it represents. While this abstraction provides extensibility, concrete definitions are necessary for an implementation. For this reason, two assertion types were designed: field mapping assertions and field state assertions.

Field mapping assertions are assertions found in preconditions that represent constraints on the mapping combinations of input fields. Only specific combinations of an operator's multiple input fields may be valid to map conjunctively. An input field can be required, disallowed, or optional for a given precondition. An argument field must be mapped to a required input field for the condition to be satisfied. Conversely, an argument field must not be mapped to a disallowed input field to satisfy the condition. Finally, an optional input field indicates the absence of a constraint, in which case it is acceptable to either map or not map an argument field. The representation of the field mapping assertion contains a reference to the input field the assertion is for, and a Boolean value to express whether the assertion should evaluate to true when the specified field is mapped or when it is not mapped.

To illustrate this assertion, consider the operator in Figure 5. Assume that "Input 1" is required "Input 2" is disallowed, and "Input 3" is optional. An expression of these conditions would include the following field mapping assertions: ("Input 1", true) and ("Input 2", false). Note the lack of an assertion for "Input 3", since it is optional. The precondition of this example can be satisfied by two cases. The first accepted case is 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". The second accepted case is the same for "Input 1" and "Input 2", but an argument field is not mapped to "Input 3". Instead of using the absence of an assertion to express the optional case implicitly, an alternative is to split the precondition into two explicit cases: {("Input 1", true), ("Input 2", false), ("Input 3", true)} and {("Input 1", true), ("Input 2", false), ("Input 3", false)}.

Figure 5. An example operator with multiple inputs.

Next are field state assertions, which can be found in both preconditions and postconditions. Field state assertions in preconditions represent constraints on the state of an argument field's data for mapping to an input field. These assertions specify the required content type, any state attributes that are required, and any that are disallowed for a given input field. The representation of the field state assertion references an attributed field, defining which input field the assertion is for and criteria on content type and state attributes. A Boolean value is also part of the representation to configure the assertion's evaluation. True denotes that both an argument field's content type and all of its state attributes must match those of the attributed field referenced by the assertion. False differs only in that none of an argument field's state attributes must match those of the attributed field referenced by the assertion.

As an example, consider again the operator in Figure 5. Assume "Input 1" requires the content type C1 and the state attribute A1. Also assume that "Input 2" requires the content type C2 and the state attribute A2, but the state attribute A3 is disallowed. An expression of these conditions would consist of the following field state assertions: ("Input 1", C1, {A1}, true), ("Input 2", C2, {A2}, true), and ("Input 2", C2, {A3}, false). The precondition of this example can be satisfied when an argument field with the content type C1 that has the state attribute A1 is mapped to "Input 1", and an argument field with the content type C2, that has the state attribute A2, but not the state attribute A3, is mapped to "Input 2".

Field state assertions for postconditions represent the "effect" produced by an operator's execution. This encompasses altering the state of an existing data field's or appending new data

fields with specific states. For the former case, the assertion specifies which existing field is altered and what changes are made to the content type and state attributes via the referenced attributed field. True for the Boolean value designates merging the specified state attributes with those of the existing field, while false indicates removing the specified state attributes from the existing field. For the latter case, the Boolean value is ignored. Instead, a new field is added to the data set that has all characteristics of the referenced attributed field – the name, data type, content type, and state attributes.

Figure 6. An example operator with a single input and output.

To demonstrate, consider the operator in Figure 6. Assume one result of this operator is to add a new field "Output 1" with the content type C1 and the state attribute A1. Also assume another result is to alter "Input 1" by changing its content type to C2 and adding the state attribute A2. These conditions could by expressed by the following field state assertions: ("Output 1", C1, {A1}, true) and ("Input 1", C2, {A2}, true). Evaluation of the first assertion would add a new field, "Output 1", with a content type C1 and the set of state attributes {A1}. If the argument field mapped to "Input 1" is assumed to have content type C2 and state attributes {A3}, evaluation of the second assertion would alter field "Input 1" to have the state attributes {A2, A3}. As an alternative example, consider a different postcondition which alters "Input 1" by changing its content type to C1, adds the state attribute A1, and removes the state attribute A2. These conditions can be expressed by the following field state assertions: ("Input 1", C1, {A1}, true) and ("Input 1", C1, {A2}, false). If the argument field mapped to "Input 1" is assumed to have content type C2 and state attributes {A2}, evaluation of both assertions would alter field "Input 1" to have content type C1 and state attributes {A1}.

Field state assertions are also utilized to represent the goal state of a workflow, expressing postconditions that are required to be present (inclusive) or not present (exclusive) after workflow execution. This approach does not assume that the name of each field in the final state of a workflow that satisfy a given goal are known. Instead, the goal conditions are asserted on specific content types, not specific data fields. This generalizes the specification to a category of fields rather than a specific field, permitting goal specification to be based on domain concepts and so be more expressive. The referenced attributed field of the assertion specifies which content type and state attributes are required to be present or not present. True for the Boolean value indicates the specified state attributes must be present in a workflow's final state, while false specifies that the state attributes must not be present.

For example, consider the following field state assertions: (C1, {A1}, true), (C1, {A4}, false), and (C2, {A2, A3}, true). The conditions of this example intent can be satisfied by any workflow state that contains a data field with content type C1 and state attributes {A1}, no data field with content type C1 and state attributes {A4}, a data field with content type C2 and state attributes {A2}, and a data field (possibly the same as the previous) with content type C2 and state attributes {A3}.

3.3. Workflow Engine

The workflow engine is an implementation of an AI planner. The purpose of this component is to utilize the knowledge representation to generate the full procedural specifications of ETL workflows that satisfy the requirements users express through the intent language. Workflow composition is modeled as a search space problem by mapping the input data sources as the initial state, user intent specifications as the goal state, and operators as the available state transitions. The workflow engine follows an A* strategy (see Figure 7), using forward chaining to select and arrange operators into workflows that achieve the specified goal state using the given initial state and collection of operators.

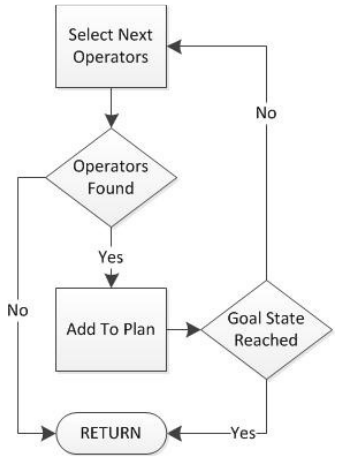

Figure 7. Process flow of the workflow engine.

Execution of the workflow engine generates a collection of steps, referred to as the workflow plan, to reach the goal state from the initial state. Multiple operators and input mappings can transition the workflow from the current step in the workflow plan closer to the goal state. The planning algorithm evaluates as deep as possible upon each available path – a series of sequential steps in the workflow plan – and ends when the goal state is reached or no next steps are available. At this point, the algorithm backtracks to the previous step in the workflow plan and evaluates any other paths that have not been explored. Execution of the workflow engine terminates once all paths have been explored. All valid workflows that reach the goal state from the initial state are then contained in the workflow plan. The remainder of this section expands on this high-level description, describing the algorithm in greater detail.

3.4. Workflow Planning Tree

An integral part of the algorithm is tracking the workflow plan. The workflow engine must be able to backtrack to a previous step in the plan when the current step reaches the goal state or has no next steps still unexplored. Similarly, the workflow engine must be able to extract each distinct paths that reaches the goal state from the plan once all paths in have been explored, as each of these paths represents a workflow solution.

The workflow planning tree is a tree-based data structure composed of a collection of connected nodes. Each node in the tree represents a step in the workflow plan. The root of the tree is the starting point of the workflow and the leaf nodes are the possible last steps based on the path taken through the tree. Each node has a reference to a single parent and one or more children, enabling the workflow engine to traverse both up and down the tree. Each node also maintains

details about the step in the workflow plan they represent, including the operator planned, along with its option and input mappings. This enables the exact specifications of a workflow to be extracted from the tree. The end state and remaining goals after the operator is executed are additional details each step records to assist the workflow engine in planning next possible steps in the workflow.

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 when execution of the workflow engine begins. Starting with the root as the current node, the workflow engine determines which operators should be the next steps in the workflow plan. These are added as the children of the current node. Then the workflow engine recursively performs the same logic on the children of the current node. Once the current node reaches the goal state or no next steps can be found, the recursive algorithm backtracks the workflow engine to the parent node and the next child is evaluated.

3.5. Matching Goals

To determine which steps should be the next in the workflow plan, the workflow engine searches for operators that bring the plan closer to the goal state. During this process, referred to as goal matching, the postconditions of all available operators are searched and matched against the remaining goals of the current step in the planning tree. A match is determined by comparing the assertions in each postcondition to each of the assertions in the remaining goals.

To perform this logic, one criteria is that the content types of the field state assertions must match; if the content types differ, then the assertion does not produce the goal under consideration. Another criteria is that assertions cannot conflict. For example, if the assertion produces a state attribute that is disallowed by the goal state, then it is not a goal match. A final criteria is that if the content type already exists in the current state of the workflow, then the assertion must also share at least one state attribute with the goal state's assertion. This allows the workflow engine to handle data fields that do not exist in the workflow until produced by an operator during workflow execution. This also prevents false positives on matching operators that are not actually bringing the workflow closer to the goal state.

3.6. Mapping Inputs

Once a candidate operator is selected through goal matching, the operator's preconditions must then be evaluated against the current workflow state to determine if a valid input mapping exists. If no valid input mapping exists, an operator should not be added to a workflow plan. Finding a valid input mapping is performed using the fields from the end state of the current step in the planning tree. Each of the field mapping assertions of the precondition are checked first. 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, the algorithm moves on to check the field state assertions. 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 candidate 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. Note that the precondition to be evaluated must be that which produces the postcondition that goal matching determined; however, this is handled by a separate step that is discussed later.

3.7. Growing the Workflow Plan

Candidate operators that pass both goal and input matching are added to the planning tree as a child of the current node in the tree. In order to continue the workflow planning algorithm, these

newly added nodes must have their end states and remaining goals evaluated. First, the postcondition of the planned operator is applied to the data fields from the input mapping to determine the end state of the new step. For each assertion in the postcondition, the input mapping is checked for matching fields. If matched, the state attributes of the attributed field from the input mapping are merged with those from the assertion. Otherwise, the attributed field from the assertion is added to the end state.

Next, the remaining goals are determined, which involves taking the difference between the end state of the child node and the remaining goals of the current node. This logic is performed by taking the field state assertions from the postcondition with a Boolean value of true, finding the data fields in the end state that match (based on content type), and taking the difference of the assertion with the matched fields. The resulting assertion has all the state attributes that were in the old assertion excluding those present in the attributed field. Once no state attributes are left in the assertion it is dropped from the result. Otherwise, it is added to the resultant postcondition. Similarly, the existing assertion is added to the resultant postcondition if no matching fields are found, as this indicates that none of its asserted goals have been satisfied. The postcondition that results contains all of the assertions that have not yet been satisfied by the current state of the workflow.

3.8. Evaluating End Conditions

As the workflow engine attempts to recursively plan the next steps of the workflow, each recursive invocation checks to see if the goal state has been reached. To evaluate the end condition, the remaining goals of the current step in the workflow planning tree are checked for two possible cases that indicate that the goal state has been reached: when there are no remaining goals and when only exclusive goals are remaining. No remaining goal is a clear end condition that doesn't require explanation. For the second case of having only exclusive goals remaining, this indicates that all-inclusive goals satisfied. So long as none of the exclusive goals are violated, then this case indicates that the goal state has been reached.

3.9. Operator Decomposition

A characteristic of operators noted in our previous work is that an operator can be overloaded with functionality, having multiple preconditions, each of which can produce a distinct postcondition. The workflow engine must know the each operator's distinct postconditions in order to determine which operators to select to transition the workflow closer to the goal state. The workflow engine must also know the distinct preconditions of each operator in order to identify the input mapping required to produce a desired postcondition. While one strategy is to compute these on demand during the workflow planning process, the computational overhead becomes excessive. The strategy employed by our approach is to perform this computational process, referred to as operator decomposition, before workflow planning begins.

The process of operator decomposition breaks up a single, complex operator into many atomic operators. The term atomic is used in this context to refer to possessing only a single precondition and postcondition. Decomposing an ETL operator into atomic operators starts from the canonical, "sum of products" form of all of the operator's preconditions (including the options). As each precondition in our approach is modelled as a conjunctive expression, the collection of preconditions tied to an operator or option setting is already in canonical form. However, the operator's preconditions must be merged with the preconditions of the options. Note that preconditions from different settings of the same option are not merged, as an option can only have a single setting at a time. The maxterms of the resulting merged canonical form expresses the operator's discrete preconditions.

As an example, consider an operator with the following preconditions: (P1&&P2) ‖ (P1&&P3). Assume the operator has only a single option with two settings, one with no precondition and the other with the precondition P4. The merged canonical form for this example specification would be: (P1&&P2) ‖ (P1&&P3) ‖ (P1&&P2&&P4) ‖ (P1&&P3&&P4). The four maxterms of this expression indicate four discrete preconditions. Next, assume the operator has an additional option with two settings, one setting with no precondition and the setting other with the precondition P5. In this case, 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). The resulting expression contains eight maxterms, indicating eight discrete preconditions.

To merge preconditions, the underlying assertions must be merged. This entails merging assertions of the same type for the same field into single assertions. For example, a precondition composed of a field mapping assertion ("F1", true) merged with another precondition that contains a field mapping assertion ("F1", true) would produce a resultant precondition with the single field mapping assertion ("F1", true). Similarly, a precondition composed of a field state assertion ("F1", C1, {A1}, true) merged with another precondition with a field state assertion ("F1", C1, {A2}, true) would produce a resultant precondition with the single field state assertion ("F1", C1, {A1, A2}, true). When assertions are conflicting, asserting that a field mapping or state attribute is both required and disallowed, this indicates that both preconditions cannot be satisfied in conjunction. In such cases, the conflicting preconditions are excluded from the merged result.

4. EVALUATION

Results were gathered by testing a prototype implementation of the approach on the services offered by an industrial partner. Their services focus on improving the quality of their clients' data. Testing scenarios were selected from real-world business cases of this target domain that 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.

4.1. 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". The expected workflow, which was produced by the target domain, is the following sequence:

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

4.2. 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". The expected workflow – once again produced by the target domain – is the following sequence.

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

4.3. Results and Analysis

The number of workflows generated was 384 for the first scenario and 7680 for the second scenario. 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. In the manual process, the first scenarios would require 32,256 input fields and 3,072 options to map by hand, and the second scenario would require 890,880 input fields and 69,120 option to map by hand. Through our approach, however, solutions were generated automatically in less than one minute for both scenarios.

Analysis of these solutions revealed only 16 distinct workflows in each scenario, all of which had compositions matching the expected result and fulfilled the specified intents. The 16 distinct compositions resulted due to the intents being under-constrained. Certain option settings were ambiguous, thus multiple option configurations attained extraneous services as these were not excluded by the goal specifications.

5. RELATED WORK

Relatively little literature exists that ties workflow as developed by the ETL community to planning from artificial intelligence. The former community generates workflows manually, while the latter automates plans that are, essentially, workflows. Some communities have considered how the two areas can fit together, which is central to our work. The relational community enables operations that involve data stored in relational databases to be expressed through a high-level, declarative language, SQL. Statements in SQL are automatically mapped to workflows known as query plans. However, a query planner is only able to map SQL statements to generic set operations, not to custom operations. Query optimization also does not incorporate domain knowledge, relying instead upon the expertise of the querying user.

Sun [15] recognized that the existing process to produce a workflow was "highly manual and cumbersome effort" and sought to automatically generate workflows as a solution. The approach employed a domain ontology to generate all possible workflows, then incorporated user-specified constraints to filter the results until only a single solution was remaining. Our work differs in that we constructively determine a solution for a given goal set instead of generating all possible workflows and iteratively pruning the excess solutions through a decision tree.

Research by Zhang et al [16] employed patterns to capture and integrate domain knowledge from experts. The approach associated these patterns with a task, scenario, and solution as an attempt to provide domain independence. In contrast to our work, this research just makes suggestions and does not fully generate workflows, leaving reliance upon domain experts in the specification

process. Further, the approach cannot generate unique workflow solutions, merely regurgitate what it has already seen.

Ambite and Kapoor [17] incorporated concepts of both domain ontology and artificial intelligence planning. Their approach introduced predicate logic to represent domain knowledge and employed a planner to generate a workflow solution for a given user request. The workflows the planner was capable of building, however, were only a simplification of ETL workflow, having only a single way to connect and sequence the available operators. Such an approach would be infeasible to apply to domains with complex workflows.

Previous research by Xiao, Thompson, and Li [18][19][20] also employed a planner. This work was capable of generating all of the possible workflows that transformed a given initial set of input fields to a specified set of output fields. Though the 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. Consequently, even invalid workflows were included in the solutions generated.

6. CONCLUSIONS AND FUTURE WORK

The scale of big data causes the compositions of data science workflows to grow increasingly complex. With the turnaround time for delivering solutions becoming a greater emphasis, stakeholders cannot continue to afford to wait the hundreds of hours it takes for domain experts to manually evaluate and select an optimal workflow. The work presented in this paper introduces an extensible approach to automatically transform declarative goal statements into fully-specified ETL workflows. Like how query optimizers transform SQL statements into specific query plans, our AI planning approach auto-generates workflows by following an A* search strategy to sequence and connect operations that fulfill the requirements specified by a user's intents. The practical contribution is an intuitive interface that facilitates rapid composition and maintenance of workflows, much like SQL provides for relational database queries. Results for real-world industrial cases were gathered and analyzed to demonstrate (in a generative manner) the validity of the approach.

There are several opportunities for further research in this problem area. Intelligent result filtering is an enhancement that can improve ease of use, enabling ambiguous goals that inflate the number of workflows generated to be identified and pruned by users for a refined result set. Discovering interchangeable elements of workflow specification, analogous to identity statements in query optimization, could revolutionize how businesses and scientists analyze and reason about data science workflows, while opening the way for derivative research into optimization. Optimization refers to the process of refactoring a workflow to improve it in some sense. For instance, 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, or minimizing workflow runtime. If equivalence relations are discovered for workflow operations, then workflows could be optimized similar to query optimization, by transforming from one form to another that produces the same result at a cheaper cost. These costs could be defined in a cost-model that users can adjust to have particular factors weighed more greatly than others, including: runtime, financial cost, and level of data quality.

REFERENCES

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

[2] "Core techniques and technologies for advancing big data," NSF Program Solicitation, March 29, 2012.

[3] 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.

[4] Deneke, W., Eno, J., Li, W., Thompson, C., Talburt, J., Loghry, J., "Towards a domain-specific modeling language for customer data integration workflow," GPC Workshops, IEEE Computer Society, 2008, pp. 49-56.

[5] Deneke, W., Li, W., Thompson, C., "Automatic Composition of ETL Workflows from Business Intents," Second International Conference on Big Data Science and Engineering (BDSE), Sydney, Australia, December 3-5, 2013.

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

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

[8] IBM Corporation, InfoSphere DataStage, URL: http://www-01.ibm.com/software/data/infosphere/datastage/, Accessed: July 04 2015.

[9] Microsoft, SQL Server Integration Services, URL: http://msdn.microsoft.com/en-us/sqlserver/cc511477.aspx, Accessed: July 12, 2015.

[10] Oracle Corporation, Oracle Data Integrator, URL: http://www.oracle.com/technetwork/middleware/data-integrator/overview/index.html, Accessed: July 12, 2015.

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

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

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.