IJDC | *General Article*

Towards Automated Design, Analysis and Optimization of Declarative Curation Workflows

Tianhong Song
Department of Computer Science
UC Davis, CA

Sven Köhler
Department of Computer Science
UC Davis, CA

Bertram Ludäscher
Department of Computer Science
UC Davis, CA

James Hanken
Museum of Comparative Zoology
Harvard University

Maureen Kelly
Harvard University Herbaria

David Lowery
Harvard University Herbaria

James A. Macklin
Agriculture and Agri-Food Canada

Paul J. Morris
Harvard University Herbaria

Robert A. Morris
Harvard University Herbaria
Computer Science Department,
University of Massachusetts

Abstract

Data curation is increasingly important. Our previous work on a Kepler curation package has demonstrated advantages that come from automating data curation pipelines by using workflow systems. However, manually designed curation workflows can be error-prone and inefficient due to a lack of user understanding of the workflow system, misuse of actors, or human error. Correcting problematic workflows is often very time-consuming. A more proactive workflow system can help users avoid such pitfalls. For example, static analysis before execution can be used to detect the potential problems in a workflow and help the user to improve workflow design. In this paper, we propose a declarative workflow approach that supports semi-automated workflow design, analysis and optimization. We show how the workflow design engine helps users to construct data curation workflows, how the workflow analysis engine detects different design problems of workflows and how workflows can be optimized by exploiting parallelism.

Received 13 January 2014 | *Accepted* 26 February 2014

Correspondence should be addressed to Tianhong Song, Department of Computer Science, University of California, Davis. Email: thsong@ucdavis.edu

An earlier version of this paper was presented at the 9th International Digital Curation Conference.

The *International Journal of Digital Curation* is an international journal committed to scholarly excellence and dedicated to the advancement of digital curation across a wide range of sectors. The IJDC is published by the University of Edinburgh on behalf of the Digital Curation Centre. ISSN: 1746-8256. URL: http://www.ijdc.net/

 doi:10.2218/ijdc.v9i2.337

Introduction and Motivation

Data curation is critical in many areas, such as the production and use of scientific data collections and repositories. For example, natural science collections data can often be rife with errors and inconsistencies, and reuse of collections data to address scientific questions imposes concerns of data quality and fitness for use upon the collection's management community. We have continued our development of Kuration 1.0 (Dou et al., 2011), a software package and prototype for automating data curation pipelines with the Kepler scientific workflow system (Ludäscher et al., 2006). Several curation tools and services are integrated into this package as actors, enabling the construction of workflows to perform and document various data curation tasks.

The typical structure of a data curation workflow includes an input actor that reads a dataset from a remote or local source, a number of data curation actors that implement different data validation methods, and an output actor that writes the result into a file or a database.

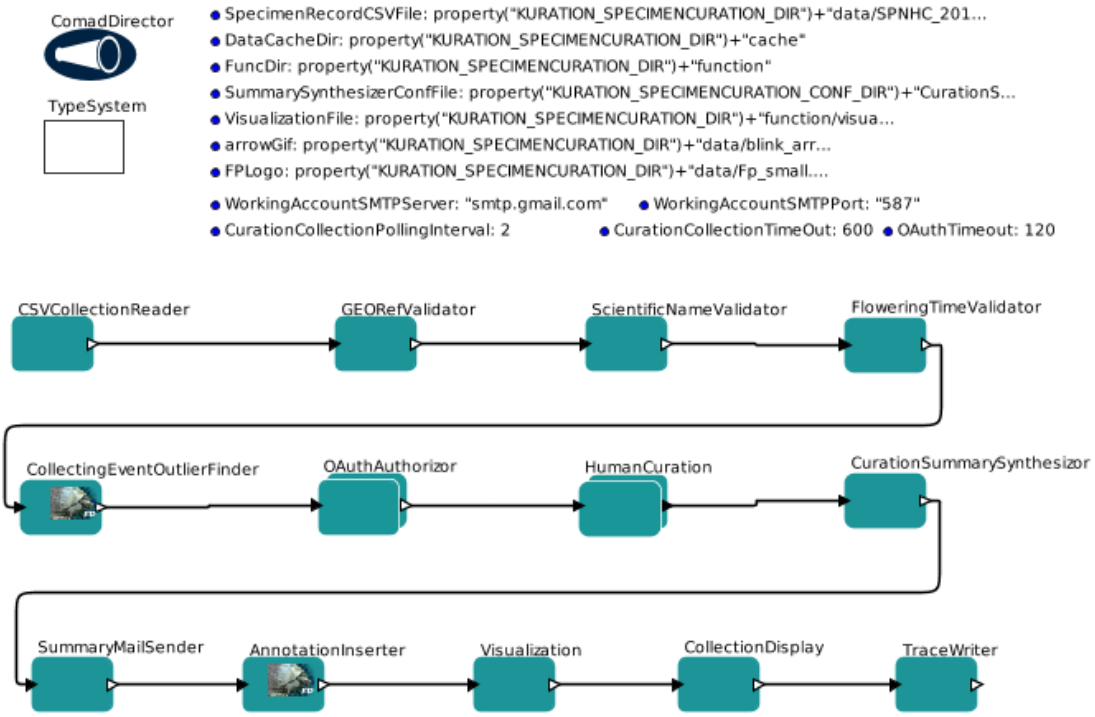

Figure 1. Kuration 1.0: A Kepler/COMAD data curation workflow for collection-oriented data quality control.

Figure 1 shows a data curation workflow in the Kepler workflow system (Dou et al., 2012) developed by using the COMAD workflow model (McPhillips et al., 2009). Boxes are *actors* (also known as processors, steps or modules) and arrows connecting them are *data channels* indicating the data flow between actors.

COMAD and the related data assembly line approach (Zinn et al., 2009a) are an improvement over the earlier and "conventional" way of designing workflows in Kepler, as they simplify the workflow design. Practical experience with our first

Kuration prototype, while promising (Dou et al., 2012), also yielded a number of technical challenges: 1) the use of remote services on large data collections, together with an unoptimized workflow execution model, creates scalability problems; and 2) the design, configuration and maintenance of workflows are not only time-consuming but also can be error-prone, e.g., the workflow design itself might have problems, such as incorrectly ordered actors in a workflow (see Q_3 below).

In order to tackle these and related issues, we propose a visionary system that (semi-) automatically detects workflow design problems and optimizes workflow design. In the first phase, the design engine selects actors from a library of existing actors based on the user's requirements and puts them together into a "workflow story" (i.e., a sequential order of actors). Alternatively, users can provide their own workflow. In the second phase, static analysis techniques are applied before runtime, which anticipate how a workflow might behave during runtime. Using workflow graph and actor configurations, corresponding data dependency information can be captured, which in turn can be used to identify possible design problems. In the third phase, the candidate designs are further improved or optimized (e.g., exploiting parallelism) in order to achieve better performance. The flowchart in Figure 2 shows how our system works.

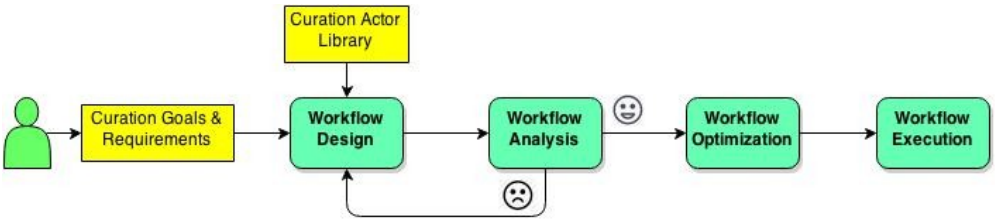

Figure 2. Overview of the proposed workflow design, analysis and optimization system.

The remainder of this paper is organized as follows: first, we introduce our workflow model; next, we describe detailed design, analysis and optimization techniques and scenarios with examples; finally, we conclude with additional implementation details and a discussion of future work.

Workflow Model

The collection-oriented workflow modelling and design (COMAD) model with nested data model and scope configuration has shown great advantages for workflow design, such as improving workflow reusability, simplicity, predictability and ease of use (McPhillips and Bowers, 2005). Data curation workflows also deal with collection-oriented data in most cases, so we propose a simplified workflow model that follows the principles of the COMAD model and models data curation workflows.

 doi:10.2218/ijdc.v9i2.337

Data Curation Workflow Model

Typically, the input of data curation workflows is structured as a *data stream*, which consists of a set of *records*. Each record contains a set of attribute-value pairs (or *data items*) and the value is a concrete data value.

A *workflow* consists of a set of actors, and the connection between two actors indicates how data flows. Each actor has exactly one input port and one output port. The input port takes a record from the data stream and invokes the actor on it, which produces an output with a similar structure to the input. One exception is for the first actor, usually the reader, which can be invoked with a special trigger by the workflow system while the actor's input port is empty. Workflows in our model will always be linear workflows without looping and branching. Instead, looping and branching will be handled implicitly within the actors if needed (as in COMAD model).

Each data validation actor also contains a black-box and a configuration. The black-box implements the actual data-processing logic. It is wrapped with a well-defined configuration, which includes a set of *read scope* functions for selecting relevant parts of the input data stream and a set of *write scope* functions for combining the result of the black-box with the unselected part of the data stream to form the output of the actor. Each black-box has a set of input ports and a set of output ports. For each data item read by a black-box on its input ports, the black-box either validates the data item and *updates* it, or *reads* this data item only during validation of other data items. The unselected part of the data stream is transported, *bypassing* the black-box, and forms part of the output of the actor. The above three types of dependencies between the input and output of the black-box (*update*, *read*, *bypass*) are crucial to establish the provenance of data items. Similar dependency declarations are used to infer detailed provenance in Bowers et al. (2012).

Abstract Provenance Graphs

In order to perform analysis and optimization on a data curation workflow, more information besides the workflow graph needs to be captured. Since each actor has a configuration that declares what data type the actor reads and writes, and the input data schema could be provided by the user, by combining the configurations of all the actors in a workflow with the input data schema, the behavior of each actor can be inferred by type matching. Here, we propose a data-oriented abstract provenance graph (Zinn and Ludäscher, 2010) called an *Abstract Data Dependency Graph (ADG)*, which captures fine-grained data-dependency information of a workflow before workflow execution.

An ADG has a set of nodes, **N,** that represent data items (except for a source node representing workflow input and a sink node representing workflow output) and a set of edges, **E**, that represent dependencies among data items. Since different types of dependencies are created by actors, each edge has a label, **T:A** that indicates what type of dependency, **T**, the edge represents and which actor, **A**, creates this dependency[1]. In the following, we distinguish different types of dependencies. The variables *X, X'* and *Y* denote nodes in **N** that are data items; *Source* and *Sink* are special nodes representing workflow input and output, respectively; and A is an actor. To indicate how a variable is used, we use subscripts:

[1] For simplicity, the actor name is not shown in some of the examples.

- *update:* Data item X is validated and replaced by an updated item X' during an invocation of actor A, i.e., $X' := A(X_{update})$.
- *read:* Data item Y is used only as reference and stays unchanged after an invocation of actor A, i.e., $X' := A(X_{update}, Y_{read})$.
- *bypass:* Data item X is not used and remains the same after an invocation of actor A, i.e., $X := A(X_{bypass})$.
- *input:* Only for the edges coming out of the source node to data item X as part of the workflow input, i.e., $X := Source_{input}$
- *output:* Only for the edges going into the sink node from data item X as part of the workflow output, i.e., $Sink_{output} := X$

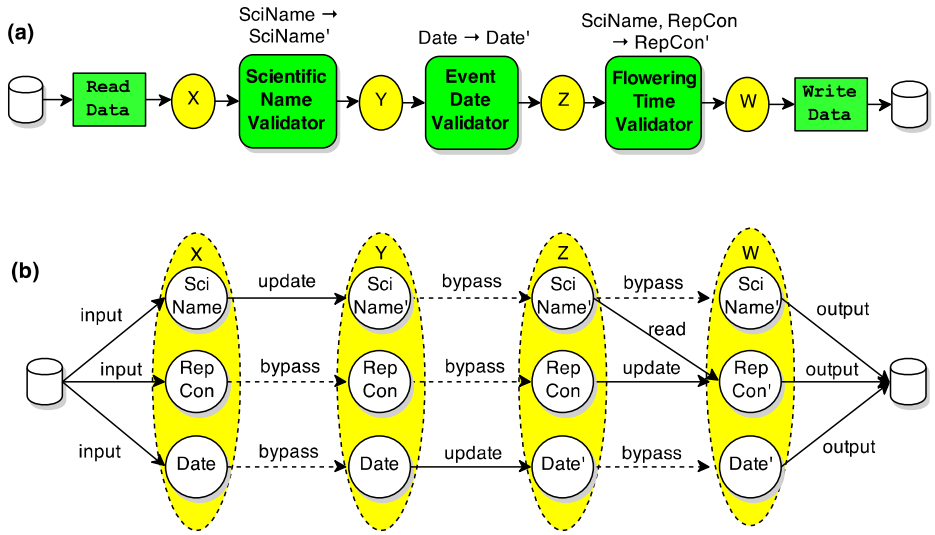

Figure 3. Example of a data curation workflow (a) and its abstract data dependency graph (b).

Figure 3 shows a workflow and a corresponding ADG. Each actor in the workflow creates some dependencies among data items in the ADG. For example, "Flowering Time Validator" validates and updates "Reproductive Condition" and reads "Scientific Name" as reference during invocation. "Event Date" is not used by the actor so it is bypassed.

Workflow Design

In some cases, the user has a dataset and wants to run data curation workflows to perform data quality control work on the dataset. Here, we will focus on a more specific case where the user has a collection of data records and wants to perform data quality control only on certain data fields. As part of the input of our workflow system, the user will provide an input schema and select the data fields on which to perform data quality control work. For example, consider a biologist examining a set of specimen records who wants to check whether the data fields labeled "eventDate" and "scientificName" are valid.

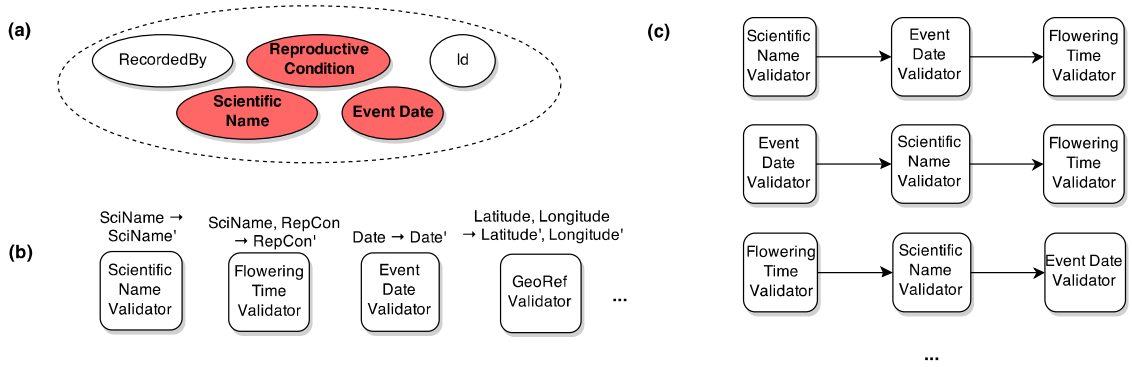

Figure 4. Example showing how the design engine works: (a) an input data schema and a set of data fields (highlighted) that need to be curated, (b) an actor library, and (c) some of the candidate workflows generated by the design engine.

After the user's requirements are specified, the design engine will try to pick a set of actors that can curate the specific data fields that the user selected and check whether all of the inputs of the actors are available in the input schema provided by the user. If some of the actors cannot be run, then the system will inform the user that some of the data curation work cannot be done. At the same time, if for some of the data fields that the user selects, no actor can be found that can perform the validation work, the system will inform the user that those data items cannot be curated.

If a set of actors can be obtained that meets the user's requirements, then the design engine will generate a linear workflow using the set of actors and pass it to the analysis engine. If the analysis engine finds that this workflow has problems such that this workflow cannot be executed properly, then the design engine will either generate another workflow and pass it to the analysis engine again until a workflow without problems can be found, or inform the user that no workflow without problems can be obtained.

Of course, the workflow can be provided by the user, in which case this workflow will be fed into the analysis engine directly.

Workflow Analysis

The analysis engine starts with the ADG of an input workflow. Since ADGs contain fine-grained dependency information, potential design problems can be detected by applying a set of graph queries that act as *constraints* on the graph and checking whether the graph violates any of them. Some of the problems can be recognized as a graph pattern, e.g., a certain type of edge cannot occur after another type. In this case, we can simply query the ADG and check the result. If the result of a certain query is empty, which indicates that a certain pattern is not present in the graph, then its corresponding workflow does not have this type of problem. Otherwise, the workflow has this problem. Here, we show examples of constraints that can be written in the regular path query, which can be applied to ADGs to detect potential design problems.

Duplicate Updates/Actors

In some cases, the same data item is validated multiple times in a data curation workflow. This is generally not necessary unless the user intends to do so. A constraint, "no duplicate updates," can be enforced to warn the user about unnecessary validation steps, which can be formulated as:

$$Q_1 \qquad \text{update} \,.\, \text{bypass}^{*} \,.\, \text{update} = \emptyset$$

More specifically, the same actor may occur multiple times in a workflow but only one occurrence is needed. Thus, the constraint "no duplicate actors" can be enforced, which can be formulated as:

$$Q_2 \qquad \text{update:A} \,.\, \text{bypass}^{*} \,.\, \text{update:A} = \emptyset$$

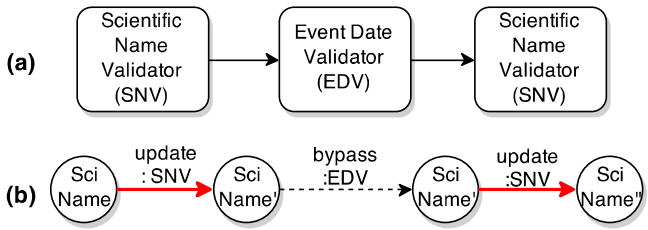

Figure 5. An example of duplicated updates/actors.

Workflow Ordering

In a workflow, one actor may read the output of a second actor; that is, the first actor depends on the second actor. A typical case in data curation workflows is that one validation actor validates a data item but also needs to read another data item as a reference during validation. In such cases, we need to enforce a constraint that each data item used as a reference must be validated first. Otherwise, actors may yield incorrect results. This constraint can be formulated as

$$Q_3 \qquad \text{input} \,.\, \text{bypass}^{*} \,.\, \text{read} = \emptyset$$

Sometimes, the above constraint is so strong that, for some data items, no actor is available to perform validation. In such situations, we can enforce a weaker constraint that states, for each data item, that if a validation actor exists in the workflow that validates this data item, then this actor must be upstream of any other actors in the workflow that read this data item. This constraint can be formulated as:

$$Q_4 \qquad \text{read}^{-1} \,.\, \text{bypass}^{*} \,.\, \text{update} = \emptyset$$

 doi:10.2218/ijdc.v9i2.337

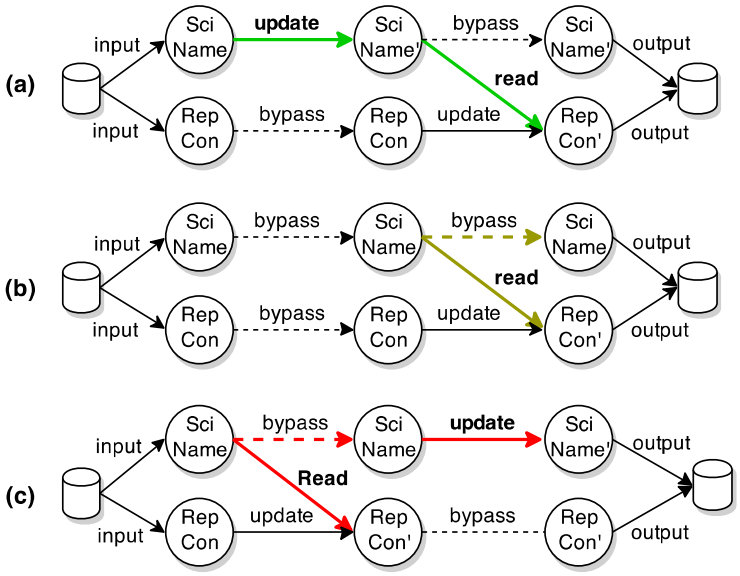

Figure 6. Three cases of workflow ordering: (a) meets strong and weak constraint, (b) only meets weak constraint, and (c) doesn't meet any constraint.

In Figure 6(a), the strong constraint is met since "SciName" is validated first, before being read. In Figure 6(b), the weak constraint is met since there is no "Scientific Name Validator" in the workflow. In such cases, it is permissible to use un-validated "SciName". In Figure 6(c), neither of the constraints is met. There exists a "Scientific Name Validator" in the workflow that validates "SciName, " but un-validated "SciName" is read before it is validated, which violates both constraints.

Workflow Optimization

ADGs can also be used to improve workflow design. Instead of constraints, *optimization opportunities* can be discovered by querying the ADG. If the query result is not empty, then some of the opportunities have not been exploited. Also, after the corresponding workflow has been improved, the same query can be applied again (whether or not the result is empty) to check whether this type of opportunity has been fully exploited. Here, we show examples of optimization opportunities written in the regular path query.

Data Forwarding

In collection-oriented workflows, each actor works on different parts of the input data and the rest of the data is forwarded, bypassing the actor. If a workflow contains several actors, then it is possible that some data items may be bypassed many times before an actor reads or validates them. This situation can be optimized by forwarding each data item to the actors that use it directly. This opportunity can be formulated as:

$$Q_5 \qquad \text{bypass} \,.\, \text{bypass} \,.\, (\text{read}|\text{update}) \neq \emptyset$$

After the workflow is improved, we can apply the above query (Q_5) again to make sure all data forwarding is optimized.

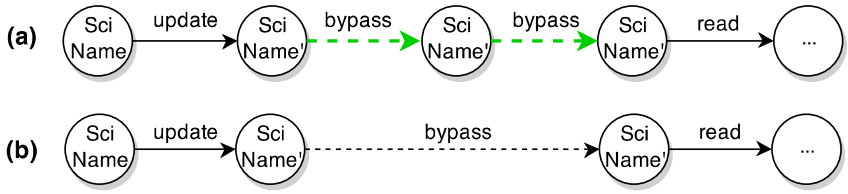

Figure 7. Example showing how data items can be forwarded more efficiently.

In Figure 7(a), a data forwarding opportunity exists since "SciName" is not changed along multiple adjacent "bypass" edges. In Figure 7(b), all data forwarding opportunities have been exploited.

Parallelism

In Figure 3, the "Event Date Validator" can be run in parallel with the other two actors in the workflow since it only works on "Event Date", which is not used by the other two actors. This situation can also be represented as "Flowering Time Validator" must be a downstream actor of "Scientific Name Validator." In order to exploit data parallelism in a workflow, actor dependency information must be captured first. Dependencies among actors can be captured as partial orders between actors. The query to capture those partial orders can be formulated as:

$$Q_6 \qquad \text{update:Actor\_A} \,.\, \text{bypass}^{*} \,.\, \text{read:Actor\_B} \neq \emptyset$$

After dependency information is obtained, the analysis engine will try to rebuild a workflow in alternative ways that do not violate those dependencies (not illustrated).

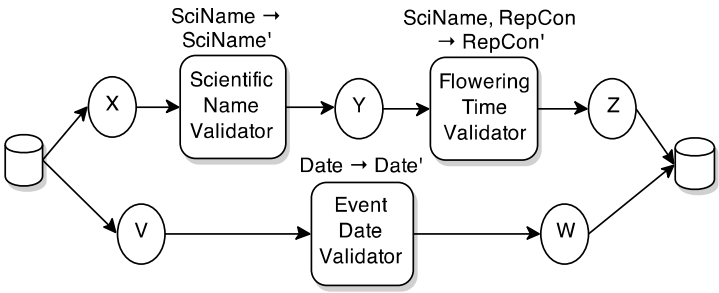

Figure 8. Example of a data curation workflow with parallel execution.

In Figure 8, "Scientific Name Validator" and "Flowering Time Validator" are not independent since "Flowering Time Validator" reads the output of "Scientific Name Validator." On the other hand, since "Event Date Validator" only works on "EventDate," it is independent of the other two actors and can be run in parallel with them. The workflow in Figure 8 is more "parallel" than the workflow in Figure 3; all data items

generated by the actors will immediately be passed to the actors that read the items or form part of the final result of the workflow during execution. Thus, the workflow can be executed more efficiently.

Prototypical Implementation

We have implemented the first prototype of the proposed system using DLV[2] for the workflow design, analysis and optimization engine. As an underlying parallel execution platform, we use Akka[3], an actor-oriented data flow execution engine that has shown higher throughput than comparable Kepler workflows.

The input of the system is specified as a set of Datalog facts, including the workflow specification, input data schema and actor configuration. The system is implemented as Datalog programs, which perform design, analysis and optimization on the input of the system. The output of the system is a valid and optimized workflow encoded as a set of Datalog facts with some comments. These results will be presented to the user as feedback. If the user decides to use the improved workflow, the workflow will be fed into the execution engine. If not, the user will modify the requirements or specify the workflow directly and run the system again. The workflow in the execution engine will be transformed to a java source file, which will be executed through the Akka workflow execution framework.

Related Work

Workflow analysis and optimization in general have been widely studied. Basu and Blanning (2000, 2007) proposed a formal approach to workflow analysis based on data flow analysis using metagraphs. Meda et al., (2010) extended the data flow analysis approach above and proposed a graph traversal algorithm that maintains workflow correctness by detecting data flow errors, but they focus on distributed computing where data may be inconsistent among different computing nodes. Zinn et al. (2009b) developed X-CSR, an optimization technique that minimizes the data shipping cost in distributed settings by determining whether data stream fragments are relevant to an actor, thereby allowing irrelevant fragments to be bypassed. Ghoshal et al. (2013) took a similar approach to ours where static analysis is used to identify provenance information prior to a program execution. However, they focus on using static (abstract) provenance to better understand concrete provenance, not to improve workflow design. Cohen-Boulakia et al. (2014) have developed an approach to detect and resolve workflow design problems based on graph refactoring. Similar work has also been done in the domain of business process management (Brogi et al., 2008) and process-aware information systems (Weber et al., 2008). However, little work focuses on collection-oriented workflows. BioVeL (Vicario et al., 2011) is a virtual e-laboratory that supports biodiversity research by providing a web service and workflow library for data processing and curation.

2 DLV: http://www.dlvsystem.com
3 Akka: http://akka.io

Conclusion and Future Work

In this paper, we propose a declarative system that supports automated design, analysis and optimization of data curation workflows. We start with a limited use case in which the user provides a set of natural science collection-oriented data and specifies what data fields they intend to run quality control work on. The design engine will construct a workflow according to the user's requirements, and a generate-and-test loop is invoked until an error-free workflow design is obtained. The obtained workflow will be further optimized to achieve better performance and eventually be executed by the underlying workflow engine.

The resulting workflow can be optimized in different ways according to different optimization criteria. We only provide selected examples of such optimization, but more optimization techniques can be applied in the future. At the same time, we have only produced a prototype of our proposed system, and different parts of the system can be improved. For example, the user interface of our system is not very user friendly, and workflow execution is not automatic (it requires manual parameter setting).

Acknowledgements

This work is supported in part by NSF:DBI:0960535, NSF:DBI:1356438 and NSF:DBI:1356751.

References

Basu, A., & Blanning, R.W. (2000). A formal approach to workflow analysis. *Information Systems Research*, *11*(1), 17–36. doi:10.1287/isre.11.1.17.11787

Basu, A., & Blanning, R.W. (2007). *Metagraphs and their applications*. New York, NY: Springer. Retrieved from http://books.google.co.uk/books?id=9DlhUIJklqUC

Bowers, S., McPhillips, T., & Ludäscher, B. (2012). Declarative rules for inferring fine-grained data provenance from scientific workflow execution traces. In P. Groth & J. Frew (Eds.), *Lecture Notes in Computer Science: Vol. 7525. Provenance and Annotation of Data* (pp. 82–96). doi:10.1007/978-3-642-34222-6_7

Brogi, A., Corfini, S., & Popescu, R. (2008). Semantics-based composition-oriented discovery of web services. *ACM Transactions on Internet Technology*, *8*(4), 1–39. doi:10.1145/1391949.1391953

Cohen-Boulakia, S., Chen, J., Missier, P., Goble, C., Williams, A.R., & Froidevaux, C. (2014). Distilling structure in Taverna scientific workflows: A refactoring approach. *BMC Bioinformatics*, *15*(Supplement 1), S12. doi:10.1186/1471-2105-15-S1-S12

Dou, L., Cao, G., Morris, P.J., Morris, R.A., Ludäscher, B., Macklin, J.A., & Hanken, J. (2012). Kurator: A Kepler package for data curation workflows. *Procedia Computer Science*, *9*, 1614–1619. doi:10.1016/j.procs.2012.04.177

Dou, L., Zinn, D., McPhillips, T., Kohler, S., Riddle, S., Bowers, S., & Ludäscher, B. (2011). Scientific workflow design 2.0: Demonstrating streaming data collections in Kepler. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering* (pp. 1296–1299). IEEE. doi:10.1109/ICDE.2011.5767938

Ghoshal, D., Chauhan, A., & Plale, B. (2013). Static compiler analysis for workflow provenance. In *Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science* (pp. 17–27). New York, NY: ACM Press. doi:10.1145/2534248.2534250

Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., … Zhao, Y. (2006). Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, *18*(10), 1039–1065. doi:10.1002/cpe.994

McPhillips, T., & Bowers, S. (2005). An approach for pipelining nested collections in scientific workflows. *ACM SIGMOD Record*, *34*(3), 12–17. doi:10.1145/1084805.1084809

McPhillips, T., Bowers, S., Zinn, D., & Ludäscher, B. (2009). Scientific workflow design for mere mortals. *Future Generation Computer Systems*, *25*(5), 541–551. doi:10.1016/j.future.2008.06.013

Meda, H.S., Sen, A.K., & Bagchi, A. (2010). On detecting data flow errors in workflows. *Journal of Data and Information Quality*, *2*(1), 1–31. doi:10.1145/1805286.1805290

Vicario, S., Hardisty, A., & Haitas, N. (2011). BioVeL: Biodiversity Virtual e-Laboratory. *EMBnet.journal*, *17*(2), 5–6. doi:10.14806/ej.17.2.238

Weber, B., Reichert, M., & Rinderle-Ma, S. (2008). Change patterns and change support features – Enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering, 66*(3), 438–466. doi:10.1016/j.datak.2008.05.001

Zinn, D., Bowers, S., McPhillips, T., & Ludäscher, B. (2009a). Scientific workflow design with data assembly lines. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science* (Article 14). New York, NY: ACM Press. doi:10.1145/1645164.1645178

Zinn, D., Bowers, S., McPhillips, T., & Ludäscher, B. (2009b). X-CSR: Dataflow optimization for distributed XML process pipelines. In *Proceedings of the 2009 IEEE 25th International Conference on Data Engineering* (pp. 577–580). IEEE. doi:10.1109/ICDE.2009.72

Zinn, D., & Ludäscher, B. (2010). Abstract provenance graphs: anticipating and exploiting schema-level data provenance. *Lecture Notes in Computer Science: Vol. 6378. Provenance and Annotation of Data and Processes* (pp. 206–215). doi:0.1007/978-3-642-17819-1_23