Colt: Concept Lineage Tool for Data Flow Metadata Capture and Analysis

Kareem S. Aggour, Jenny Weisenberg Williams, Justin McHugh, Vijay S. Kumar
GE Global Research
Knowledge Discovery Lab
Niskayuna, NY 12309 USA
1-518-387-4047

{aggour, weisenje, mchugh, v.kumar1}@ge.com

ABSTRACT

Most organizations are becoming increasingly data-driven, often processing data from many different sources to enable critical business operations. Beyond the well-addressed challenge of storing and processing large volumes of data, financial institutions in particular are increasingly subject to federal regulations requiring high levels of accountability for the accuracy and lineage of this data. For companies like GE Capital, which maintain data across a globally interconnected network of thousands of systems, it is becoming increasingly challenging to capture an accurate understanding of the data flowing between those systems. To address this problem, we designed and developed a concept lineage tool allowing organizational data flows to be modeled, visualized and interactively explored. This tool has novel features that allow a data flow network to be contextualized in terms of business-specific metadata such as the *concept*, *business*, and *product* for which it applies. Key analysis features have been implemented, including the ability to trace the origination of particular datasets, and to discover all systems where data is found that meets some user-defined criteria. This tool has been readily adopted by users at GE Capital and in a short time has already become a business-critical application, with over 2,200 data systems and over 1,000 data flows captured.

1. INTRODUCTION

The recognition of data as a strategic asset critical to business operations has made corporate data management substantially more important over the past decade [1]. Infrastructure for corporate data management in companies such as GE Capital is comprised of a global network of thousands of systems managing thousands of different types of data. In the wake of the global financial crisis of 2007-2009, the US Federal Government recognized the need for companies to maintain accurate and complete data to effectively manage and report risk. Specifically, the Dodd–Frank Wall Street Reform and Consumer Protection Act of 2010 [2] grants the US Federal Reserve (Fed) expanded oversight of large banks and other companies designated as Systemically Important Financial Institutions (SIFIs). Today, the Fed mandates that all SIFIs have the capability to track and report the lineage of data used for financial risk classifications and other important decisions.

SIFI regulations created a data challenge of maintaining a cohesive view of data flows between systems which create, manipulate or store financial data. While GE is no longer a SIFI as a result of its significant divestitures from GE Capital, the desire to address the broader data management challenge persists. In addition to the ability to capture the cohesive view of data flows, it is important to be able to interactively query and perform ad hoc reporting for internal information sharing and oversight. We built a Concept Lineage Tool ('Colt') to address this challenge. With Colt, users can capture information about different types of systems and metadata about the data flowing between them, and answer questions about the lineage of those datasets. This information is made available for interactive exploration in the form of a directed graph (data flow network) where nodes represent data producers and/or consumers (hereafter referred to more generally as 'data stores'), and edges represent data flows between those data stores.

Colt enables users to fully characterize data flows using an extensible set of hierarchical taxonomies, including many potential combinations of *businesses*, *products* and *concepts* associated with data flowing between systems. Further, we have implemented a rich set of features to facilitate the mining of the data flow network, including the ability to view the entire network, view only those systems upstream and/or downstream of a single system or collection of systems, and/or view the network filtered by terms selected from the data flow taxonomies. Colt provides key features to understand how data propagates through the network, including tracing the lineage of data upstream from a system to its origin system(s) or downstream to its destination(s). This tracing capability is critical for assessing the potential impact of changes to one or more of the data stores. Users can also validate the network to ensure that no node is sending out information that it neither creates nor receives from another node. Finally, Colt enables administrators to create and edit the network.

While Colt was initially built to address specific needs within GE Capital, it was purposefully designed to be highly reusable within other organizations and even other industries outside of financial services. The model and taxonomies used for capturing metadata about the data flows can be easily extended, modified, or entirely replaced with a new model and taxonomies. For example, in a different business or industry, *businesses* and *products* may not be relevant metadata descriptors, whereas alternate categories may be required to model the data flows. Arbitrary categories and

Proceedings of the VLDB Endowment, Vol. 10, No. 12
Copyright 2017 VLDB Endowment 2150-8097/17/08.

taxonomies can be used within Colt without requiring code changes, making Colt a powerful, reusable tool.

This paper is organized as follows. Section 2 outlines the use case that this system was built to address, which has broad applicability outside of GE Capital. Section 3 summarizes related work in data lineage and network analysis. Section 4 provides an overview of the Colt system, and Section 5 details the data store and model. Section 6 reviews the user interface, and Section 7 describes some of the advanced capabilities found within Colt. Section 8 outlines the results achieved to date, and Section 9 provides a summary of the conclusions and future work.

2. INDUSTRIAL USE CASE

GE Capital is the financial arm of the General Electric Company. Its primary focus is on leasing and lending in the aviation, healthcare, and energy sectors where GE manufactures and sells industrial equipment. To identify prospects, track customers, and manage the overall risk exposure of the portfolio, GE Capital utilizes data from a wide range of internal and external data sources, storing this data across over 2,000 internal repositories.

Historically, the GE Capital sub-businesses operated in silos, independently collecting and managing the data relevant to their products and customers. Due to the disjoint nature of this approach, it was difficult if not impossible to get a complete picture of where all of GE Capital's data resided, or how it flowed from system to system. This severely impeded the ability to perform key data management activities, for example, to identify duplicate data or processes, or find the point of origination of a particular data element.

Prior to Colt, GE Capital like many organizations relied upon the preparation of a large number of static Visio sheets to document their data flows. It was a laborious task to collate and manage them, and this approach prevented any sort of meaningful querying, interaction, or validation of the resulting networks.

Colt was designed and developed to address this problem by enabling the capture of data lineage metadata in a dynamic, interactive tool. The challenge of being able to capture and analyze data flows through a network of data stores is faced by many if not most medium-to-large organizations and corporate entities, particularly those integrating a large number of acquisitions.

2.1 Benefits of Data Lineage

There are many benefits to capturing and recording data lineage. First and foremost is for compliance purposes—many if not most companies and large organizations need to be able to report (either to internal oversight boards or to external regulatory bodies) where business critical data originates, where it resides within their systems, and how it is transformed as it flows through their networks. Further, data lineage can provide visibility into otherwise opaque complex, often global, networks of interconnected systems. This transparency can provide crucial insights such as identifying critical systems and discovering unexpected redundancies. Finally, data lineage can help answer key questions, such as assessing the downstream impact of changes to an organization's IT infrastructure, or tracing faults upstream to their source.

2.2 Concept Lineage

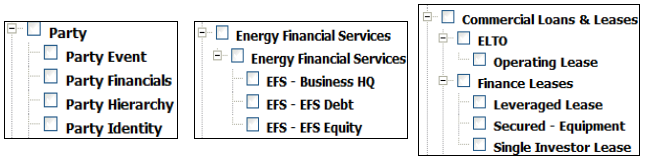
Colt is built to track *concept lineage*, where a concept is a logical grouping of multiple data elements. For example, the concept *Party Identity* may encompass data elements containing names, addresses, and phone numbers for customers, prospects, and other parties. Some concepts contain a handful of data elements, while others can represent 20 or more elements. It is sufficient for Colt to define data flows in terms of concepts, since these groups of data elements are usually kept and move together through the data stores. Further, modeling concepts is beneficial from a scalability perspective, as it reduces the volume of edge metadata that must be managed.

2.3 Taxonomies

Colt was designed and built to record complete metadata describing data flowing from one system to another. As such, the data is characterized in terms of a collection of taxonomies, including the data contents ("*concepts*") and the context in which this data exists ("*boundaries*"):

- *Concept*: the data contents (e.g. *Party Identity* for data containing customer names and addresses)
- *Concept treatment*: the action performed on the data by the consuming system (options are *create*, *read*, *modify*, and *augment*)
- *Business Boundary*: the business or sub-business for which this data exists (e.g. *GE Capital Americas*)
- *Product Boundary*: the product line for which this data exists (e.g. *Commercial Loans*)
- *Other boundaries*: Going forward, as data is further characterized, additional boundary types may be defined and associated with datasets

Concepts, businesses, and products are organized as groupings of sub-concepts, sub-businesses, and sub-products. These may be nested using an arbitrary number of levels, forming hierarchical taxonomies. Figure 1 shows excerpts from each of the three taxonomies used in Colt.



(a) Concept (b) Business (c) Product

Figure 1: Excerpts from the hierarchical taxonomies

3. RELATED WORK

Data lineage is represented as a directed network graph. In most scenarios, data lineage graphs are primarily composed of three or more abstract entities—data artifacts, the systems or transformations that operate on those artifacts, and the agents that carry out those transformations. Data artifacts here could range from tuples in a database to files or entire datasets on a local or remote file system. Sometimes, it is useful to explicitly differentiate between transformation types (e.g. OS call, SQL query) and the actual invocation of a transformation on an instance of a data artifact. While lineage networks bear some high-level

structural similarity to social networks (for example, certain nodes exhibit higher centrality measures than most others, and data flows between certain subsets of nodes are denser than others), we have not come across a formal characterization of the topological features of lineage networks.

Data Lineage: Data lineage exploration has been previously studied within various contexts. Spreadsheet tools capture lineage at the granularity of individual cells and show how the contents of a cell are derived from other cells. Provenance-aware storage systems [3][4][5] typically intercept operating system calls to record information on per-file creation, modification and deletion so that the files are easy to track, maintain and query. Scientific workflow systems [6] allow rich annotations of workflow components and record application-level lineage at the granularity of files or datasets [7][8][9][10]. A number of commercial solutions [11][12][13][14][15] supplement traditional data warehouses and business intelligence ecosystems by capturing lineage on a per-tuple basis, recording the loading and transformations that resulted in the creation of each tuple. Lineage tracking for modern Big Data ecosystems is surveyed in [16][17]. Commercial solutions in this space such as Teradata Loom [18] and Cloudera Navigator [19] track and collect per data object lineage of unstructured datasets as they flow through data cleaning and processing steps in a Hadoop-based ecosystem. Tools for collecting provenance data and assisting in provenance-based search in social media networks have been proposed in [20][21]. Finally, recent approaches propose generic support for data lineage/provenance in heterogeneous ecosystems that do not constrain the nature and types of systems involved—either by extensible transformation of all lineage information into a generic log [22], or into a generic property graph [23], or via a portable general-purpose provenance library that applications can tap into [24], or by implementing the Open Provenance Model [25]. Our work is similar to these latter approaches in that Colt manages lineage of data across generic heterogeneous data systems.

Lineage Representation and Storage: Lineage graphs have been represented using different models and persisted in a variety of data stores. In some solutions, lineage information is stored in meta-files co-located with the data or as metadata within the headers of the files themselves, an approach that is not amenable to querying or visualizing the lineage information. Relational models have been used for storing and querying lineage in [3][4][13] but does not provide native support for graphs; hence even simple network traversals must be expressed via complex joins between tables. XML is another popular model for representing lineage networks [9][10][25]; however, the hierarchical nature of XML is a drawback when representing say, entities with multiple parents. Finally, graph-based models are increasingly being used for lineage. Both property-graph models [5][12][23] and semantic graph models [26][27] have built-in capabilities for efficient operations on graphs. Colt is built on a semantic model of lineage. Our work differs from these others in that Colt maintains lineage information at the concept level, with metadata describing flows in the context of concepts, businesses, and products. Their hierarchical organization enables rich queries—a key feature which is not readily available in other graph-based provenance models.

Lineage Network Analysis and Visualization: Analysis and visualization of lineage networks can yield insights that help improve the corresponding applications. The Provenance Challenge [28] put forth nine representative queries that reflect the typical questions asked of lineage graphs of scientific workflows. The analysis queries issued against data lineage networks are a subset of general social network analysis algorithms and can be broadly categorized as follows. (1*) Lineage tracing* relates to determining the lineage of a data artifact, often tracing it upstream to its source(s) one step at a time [3][4][5]. (2) *Impact assessment* relates to the traversal of the network downstream from a given node of interest and can be viewed as the converse of upstream tracing. Examples of queries in this category include: "What is the estimated impact of a change to a data artifact or transformation process?" [11][13][14]. (3) *Summarization* relates to the analysis of lineage networks where most of the nodes and edges are involved. Validation of a lineage network, i.e., ensuring that all data artifacts can be traced back to reliable sources, can be considered to be a specific form of summarization. (4) *Path-finding and traversal* refers to a more generalized walk-through of a lineage network. An example query in this category is: "Traverse a subset of paths where only data artifacts of a specific type flow between systems." Colt supports each of these four categories of network analysis queries. Furthermore, Colt can filter these queries at any level and any combination of the concept, business, and product hierarchies.

Lineage in Financial Services: While federal regulations such as CCAR (Comprehensive Capital Analysis and Review) and BCBS (Basel Committee on Banking Supervision) regulation number 239 mandate that financial institutions deemed systemically important maintain lineage of critical data and other information used for risk and capital assessments, we are not aware of any Fed-recommended guidelines or methodologies for lineage capture and management within the financial domain. Consequently, organizations have adopted a range of solutions towards lineage analysis for ad hoc reporting purposes, including: (1) using enormous spreadsheets and Visio diagrams, an approach that is laborious and not automated, (2) utilizing the collective capabilities of multiple off-the-shelf commercial lineage tools, an approach that is expensive and not customizable, or (3) developing complex software, often resulting in systems that are highly customized and non-extensible.

Prior art indicates that certain large-scale financial institutions are developing their own in-house solutions and systems for tracking data lineage [29][30]. Additionally, anonymous case studies point to Informatica/Diaku Axon [31], Collibra [32], IBM's InfoSphere Metadata Workbench [33] and DATUM as the leading third-party vendor tools of choice for data lineage management in medium to large financial enterprises. Our own qualitative evaluation of third-party tools such as Axon for the GE Capital use case revealed shortcomings with respect to creating custom metadata groupings and extensible hierarchical taxonomies. Colt bears similarity to the Ontology and Provenance financial services offered by Bloomberg [34][35] in that it utilizes a semantic model to represent data concepts from multiple sources, thereby allowing business analysts to capture key information in a consistent, standardized manner. However, Colt differs in its use of semantics and the associated W3C standards-based technologies to also model lineage of data flowing between systems and to allow interactive querying of the lineage information. Importantly, the model underpinning Colt is flexible and extensible enough to support business logic changes to

the metadata taxonomies (e.g. additional hierarchies or updates to the existing taxonomy).

4. HIGH-LEVEL SYSTEM OVERVIEW

The goal of this work is to provide GE Capital with the capability to characterize how key data moves through a large number of data systems. To this end, we designed and developed a reusable solution enabling users to capture information about systems and metadata about the data flowing between them, and then interactively explore this information to answer business-critical questions.

4.1 Requirements

Key requirements for Colt included the ability to: (1) display the entire network of systems, (2) starting from a particular system, view the systems feeding data to or consuming data from that particular system, (3) create, modify, and delete data flows using the appropriate concept-boundary taxonomies, (4) filter the network views using those taxonomies (e.g., only show data flows containing *Party Identity*), (5) validate the network to assess if the representation is logical, and (6) starting from a specific node, trace the data in that node to discover its origination or destination points.

In addition, a set of administrative users must be able to edit the network, performing tasks such as adding systems and defining the flows between them, and generating text-based reports of the data flows.

4.2 Architecture

To meet these requirements, we designed a system based on the conceptual architecture shown in Figure 2. The data storage layer is comprised of a semantic triple store. An HTML and JavaScript web-based user interface allows users to perform most of the functions outlined above. For the remaining functions, a set of Java microservices enables functionality such as network tracing, which requires substantial overhead to execute.

4.3 Data Storage

Semantic technologies were chosen to model and store the network data. The decision to use semantic technologies in lieu of a more traditional relational or NoSQL store was made for multiple reasons. First, a store with intrinsic recursive graph query capabilities enables the efficient network traversal functionality required in Colt. Second, semantic models enable a high level of expressivity for representing the hierarchical data flow metadata in an easily extensible format. This allowed Colt's development to be accomplished in stages, with the graph being monotonically extended over time. A traditional database environment would have required a large number of schema migrations or the introduction of a convoluted schema as extensions occurred. Using a NoSQL system would have simplified the schema approach, but would have introduced significant overhead at query time. A W3C standards-compliant semantic triple store is used to store the data model and instance data as tuples [36]. Data is queried using the SPARQL query language, standardized by the W3C [37]. Finally, the use of semantic models increases the capability to infer additional knowledge about the lineage networks through automated reasoning capabilities.

5. SEMANTIC MODEL

A semantic model authoring technology called SADL (Semantic Application Design Language) was used to create the semantic model and ingest instance data [38]. SADL provides a controlled English language that compiles directly to a proper subset of OWL 1.1. Using controlled English allows for descriptive models to be generated in a manner that is comprehensible to subject matter experts who are not familiar with the Semantic Web. This was beneficial because it allowed GE Capital IT domain experts without Semantic Web experience to directly contribute to the creation of the model, shortening development time. The SADL ingestion tools further provide a template language which simplifies the loading of instance data to the triple store.

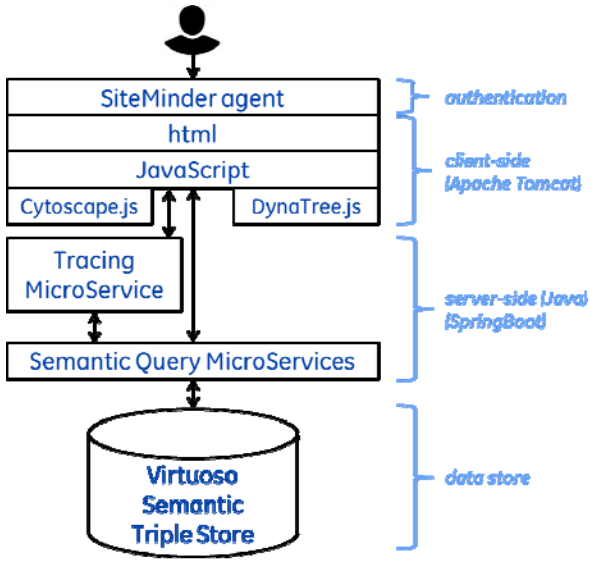


Figure 2: Colt conceptual system architecture

A semantic model was built to represent data systems and data flows between them. Figure 3 is a simplified depiction of how the model represents a producer system sending data to a consumer system. The semantic model captures metadata about the nodes, including a node name, type and a brief description. While we primarily use the terms 'system' or 'data store' when describing the nodes in Colt, nodes can be any one of the following types: (1) a database or data warehouse, (2) an external source, (3) an application, (4) a model/analytic, (5) a report, or (6) a process. Each of these types of data stores is represented with a distinct icon in the user interface.

Figure 3: data flows from a producer to a consumer

The data flow itself is modeled as a potentially complex combination of concepts and boundaries, as shown in Figure 4, with the asterisks representing one-to-many relationships. A data flow is associated with one or more structures called *boundary groups*, which in turn may contain one or more concepts and treatments and *boundary sets*. Each boundary set is a combination of boundaries (business and product boundaries, at present). It should be noted, however, that the data model and application support an arbitrary number of boundary types, such that

additional boundary types can be added solely via additions to the instance data with no semantic model or code changes required.

A data flow may consist, for example, of *Contract Terms* concepts and *Party Identity* concepts for *Single Investor Lease* products in the *Fleet Services* business (a sub-business of *North America Commercial Loans and Leases*). The *Contract Terms* data is read and *persisted* by the consumer, and the *Party Identity* data is *modified* by the consumer. Figure 5 shows this sample boundary group as it appears in the Colt interface.

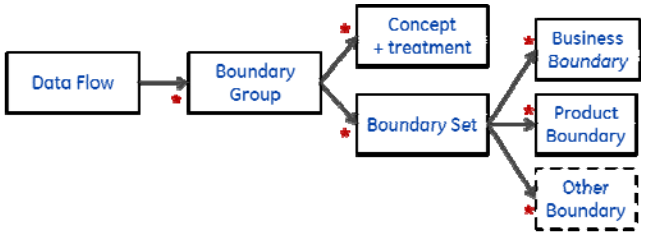


Figure 4: A data flow is a complex structure containing concepts and their treatments as well as boundaries

Further, the hierarchical nature of the concept and boundary taxonomies allow for inheritance, and therefore it is assumed that the selection of a parent implies the selection of the children. For example, if a data flow includes a Concept C, then it is assumed to include Subconcepts C_a, C_b and C_c.

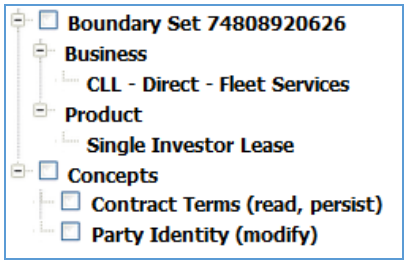


Figure 5: Example boundary group

5.1. Model in Depth

The semantic model used in Colt descends from a generic process model originally built to describe the flow of materials in an industrial manufacturing setting. In this scenario, materials are tracked as they pass through process steps. Each step can create, modify, inspect or consume material. In the event of a failure that introduced an unsuitable element into the process, the model was used to determine all impacted products. This use case is very similar to the Colt use case, where the data flows are tracked to maintain context related to which data is created, used or inspected by various data systems. The original factory model was extended to reflect GE Capital IT's terminology.

At the core of the semantic model is a concept called the MoP, short for 'material or process'. This concept originates from the model's industrial lineage, used to describe the tendency of some factory data systems to refer to both materials and process steps in the same context (requiring disambiguation to occur later). The MoP was used to describe either a material at a point in time or a particular instance of a process step being performed. The usage is the same in Colt, with data systems and data flows both extending the MoP.

Once the MoP establishes a concept of endpoints in a process segment, it is necessary to have a means of determining the temporal context in which these references occur. Given a process of several steps operating on the same data flow or material, one must be able to order them. One could use the MoPs directly, but this would induce a problem when counting unique materials or process steps. There is no problem when each step creates or meaningfully alters material because then any material appears only a single time. This fails in most common cases, however. When material passes through a non-destructive evaluation, the material is unchanged on exit. An attempt to count the number of unique materials (or process steps) by naively looking at the MoPs will produce an incorrect count.

To counter the above problem, the concept of a process slice was introduced. The process slice forms an over-arching structure on which the process steps are hung (Source, Material, Destination). For each observation in the process chain, there is a separate process slice which, in turn, is associated with a MoP. This allows MoPs to occur multiple times without ambiguity. Conceptually, this is similar to a film strip which contains sequential images. If viewed as independent images, counting the number of entities leads to an incorrect number should any entity appear in multiple images. When treated as a sequence of images which are dependent on each other, an accurate count can be made.

With these components, the basis for a semantic model supporting the capture of data stores and metadata on data flows is in place. There is a mechanism to determine ordering and a collection of objects representing our items of interest (data systems and data flows). What is now needed is the ability to describe the payload of the flow.

Concepts, the coarse-grained taxonomy of data objects exchanged between systems in the flows, define the items of interest. These are a hierarchal structure where concepts have sub-concepts. Concepts in Colt may have specific operations applied which document any changes to the state of the data. Colt associates the concept with its treatment to better characterize how the data may be mutated as it is handled. Finally, boundaries are designed as hierarchal structures of variable depth. The system is designed such that the inclusion of a boundary entails the inclusion of all of its children. A concept, boundary, and treatment collection is collected into an object known as the Boundary Group. A given Data Flow is associated with one or more boundary groups. The collection of all the boundary groups associated with a given data flow give a full depiction of the exchange between the pair of IT systems.

5.2. Extending the Boundaries

Boundaries were designed with the expectation that they would be subject to extension over time. The use case required that the boundary model meet the following four criteria:

1. The number of elements of a given type of boundary must be extensible.
2. The number of types of boundaries must be extensible (business, product, etc.).
3. The depth of the boundary hierarchy must be extensible.
4. These alterations must be able to be made at runtime without modifying the code, restarting the system, altering the underlying semantic model, or migrating the existing data.

To satisfy the first requirement, the boundaries are modeled to describe a hierarchy of variable depth. The boundary has a name and collection of boundary elements. Each element has its own name, rank, and parent. The ranks begin at one, implying this is the least granular. Elements are grouped by their rank and their parent.

To satisfy the second requirement, the boundary types are not fixed in the model. Instead, they are only described in instance data filling the model. This meets the requirement by simply adding more instance data to describe the new boundary hierarchy.

The third requirement is met by not modeling the depth directly. As described above, the depth is treated as instance data. This allows for the hierarchy to become deeper by simply altering the instance data associated with the model.

Basing the entire boundary hierarchy on the instance data and calculating it as needed satisfies the fourth and final requirement. Because the specifics of the boundary hierarchy are not reflected in the semantic model, the alteration of instance data alters the apparent boundary hierarchy.

5.3. Ingestion of Instance Data

Instance data for Colt was initially entered using a template-driven pipeline which uses an OWL model (derived from the SADL model), templates, and a set of records extracted to CSV format. The templates are an internal format used by SADL which specify the association between columns in the input data and the triple patterns to generate. The templates are also able to specify a set of transformations which may be applied to incoming the data.

Initial data coming into the system was imported from three types of sources. The first is an extract from a change management database and was used to get canonical names and metadata for the IT systems used as the data stores (producer and consumer nodes). The second source was derived from Visio and other documents which had been assembled to catalog data flows for previous reviews. These were transcribed to CSV prior to loading. The last data source was the collection of taxonomies used internally to describe the concepts, sub-concepts and boundary elements. These were run through the ingestion pipeline as the materials became available. These datasets provided the foundation of the Colt data when it was first launched. Since launch, the user interface described next has been used to enter thousands of data flows.

6. USER INTERFACE

A web-based user interface allows users to create, manage, view, and interactively explore the lineage network. Third-party Javascript packages Cytoscape.js [39] and DynaTree.js [40] are leveraged to support network visualization and menu trees, respectively. Each node in the visualized network represents a GE Capital data system (or a relevant system external to GE Capital). Each edge in the graph represents the data flow from one system to another. Figure 6 shows a sample network wherein seven systems are sending data to a system called *System H*, which in turn sends data to *System I*. Note that this is a small example—the application is capable of displaying large networks with thousands of nodes and edges. The creation of data by a given system is represented in the network using a reflexive loop, as shown in Figure 7.

The user may explore the network in several ways, via the interface shown in Figure 8. Users may choose to view the entire network of data systems and data flows. Alternatively, the user may select one or more systems and choose to display all systems that feed the selected systems ("upstream" systems) and/or consume from the selected systems ("downstream" systems). At any point, a user may click on an individual system in the network and expand the graph to include systems upstream and/or downstream of the selected node. The user may choose to display the concepts or boundaries along the edges, as well.

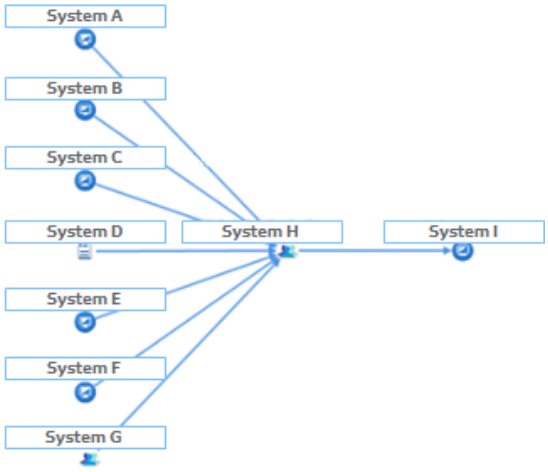


Figure 6: Example network containing 9 nodes

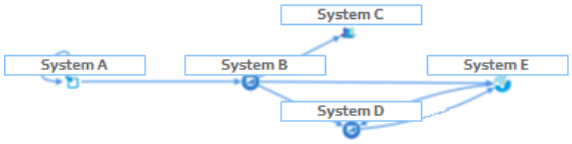


Figure 7: Left-most reflexive loop indicates data creation at *System A* node

Further, the user may apply concept or boundary filters to the network. For example, a user interested only in *Party Identity* data may apply a filter to only display the subset of the network containing this concept, hiding from view all nodes and edges not relevant to *Party Identity*. Users can filter down the network by specifying any combination of concepts, businesses, and products—Colt supports combinations expressed using conjunctive normal form; i.e., if a user selects more than one entry within a group (e.g., concepts), the system treats those with an OR operand. If the user makes multiple filter selections across different groups (e.g., business and product filter), the system treats those with an AND operand. Thus, a typical filter may look like "(*concept1* OR *concept2*) AND *business1*".

Colt supports two filter modes that treat the concept/boundary hierarchies in different ways: (1) 'any', mode and (2) 'exact' mode. Filtering using 'any' mode will display edges that contain exact matches, supersets *and subsets* of the filter, whereas filtering using 'exact' mode will only display exact matches and supersets of the filter.

Finally, the interface allows a subset of users to manage the network. Administrators may add, modify, and remove systems (as shown in Figure 9), manage boundary groups by selecting entries from the hierarchical taxonomies (as shown in Figure 10), and link systems with boundary groups to create data flows (not shown).

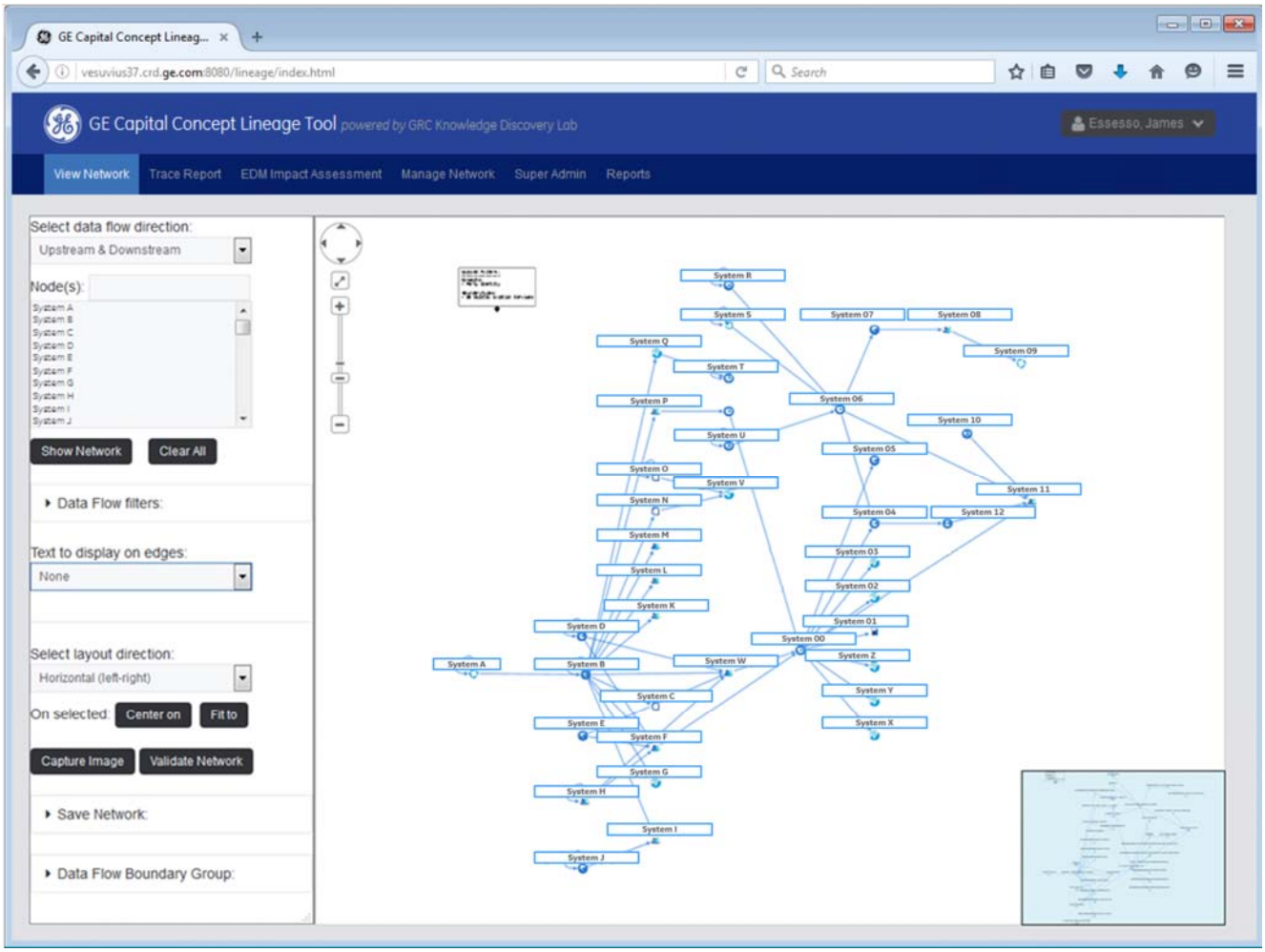


Figure 8: Colt UI to visualize and explore a network

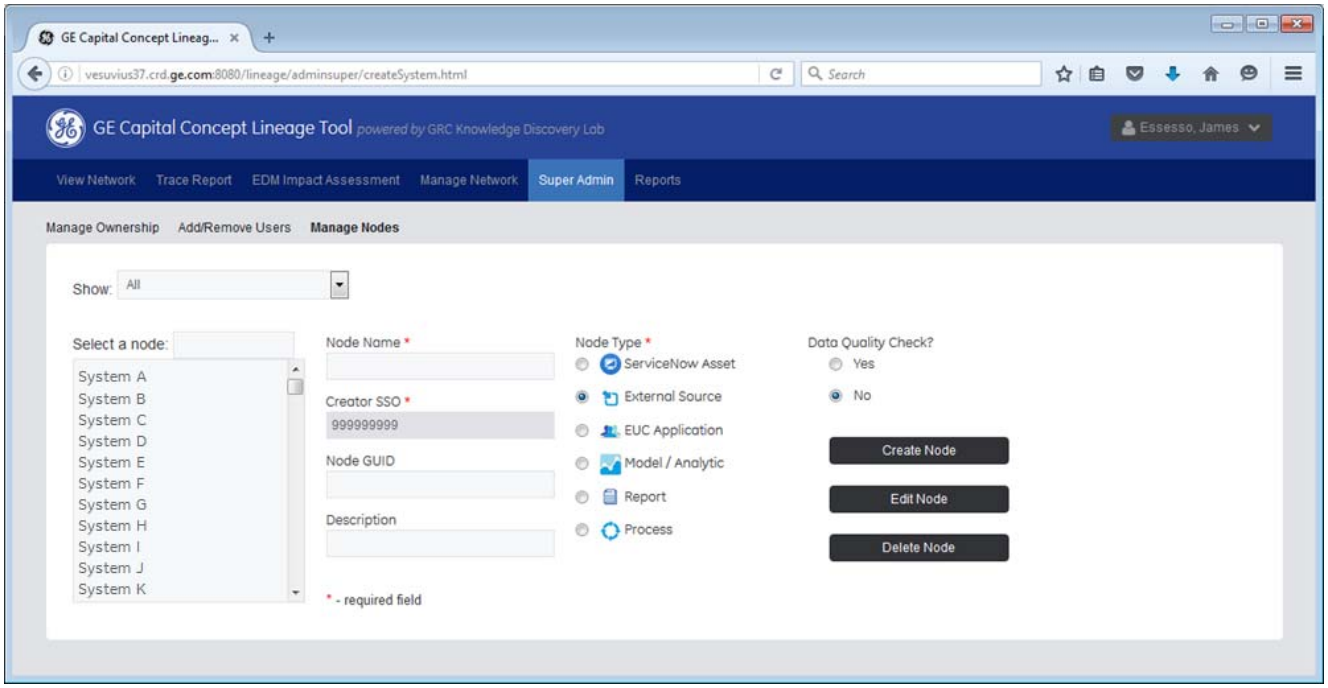


Figure 9: Interface allowing administrators to manage data systems used in the network

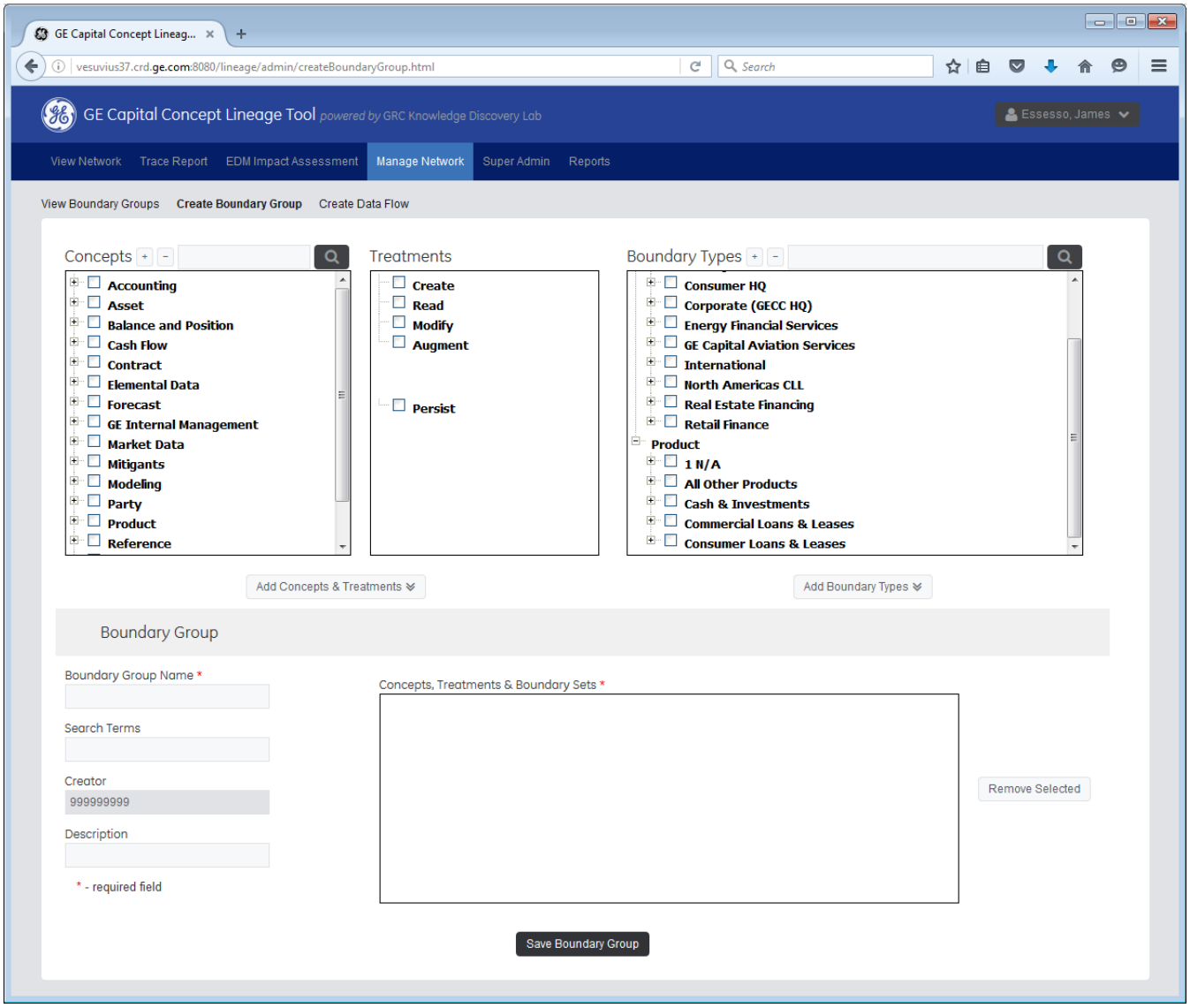


Figure 10: Colt interface allowing administrators to manage boundary groups

7. ADVANCED CAPABILITIES

In addition to being able to capture and explore the network of data systems and data flows, GE Capital users also need to answer complex questions regarding the origin and destination of particular types of data. They also need to be able to assess whether inconsistencies exist in the network as captured. This section describes how Colt provides these capabilities.

7.1 Lineage Tracing & Impact Assessment

GE Capital users need the ability to trace data concepts from a given point in the network upstream to its points of origin (typically its point of creation), or downstream to its final points of destination for impact assessment. For example, a user may be working with *Party Identity* data for *Single Investor Lease* products in the *Fleet Services* business, in the *System A* enterprise data warehouse. The user may need to provide information showing the system from which that data originated and all intermediate locations along the way until it reached the *System A* data warehouse. In another instance, the user may wish to retire a system and wishes to see all downstream systems that may be impacted.

To simplify the text for discussion purposes, we will focus on the challenge of tracing upstream from a node. However, the explanation below can be equally applied, without loss of generality, to tracing downstream by simply reversing the order of the nodes (tracing from receiver to sender for upstream vs. tracing from a sender to a receiver for downstream).

An example of tracing is illustrated in Figure 11. The tracing routine evaluates each system upstream of the given system, determines if it is passing relevant data, and if so then evaluates each system upstream of that system, and so forth, until the final points of origination are reached. In this example, an upstream trace is performed on *System Z*. The first system to be considered is *System Y*. Next, *Systems W* and *X* are considered, and so forth. When tracing the data, the algorithm must not only consider the concept, but also the boundaries of interest. In this figure,

concepts, businesses, and products are represented in the form {*c,b,p*}, since the metadata to be traced is defined by the triples {concept, business, product}. When tracing upward from *System Y*, note that *System W* is determined to be included in the trace (since it sends the data of interest {*c,b,p*}), but *System X* is excluded from the trace because it sends product *q* instead of product *p*, and thus is not relevant to this trace. Note that this is a highly simplified example, and in reality each edge contains many different concept-business–product combinations, and each must be checked in turn to see if any relevant data is sent to the traced node.

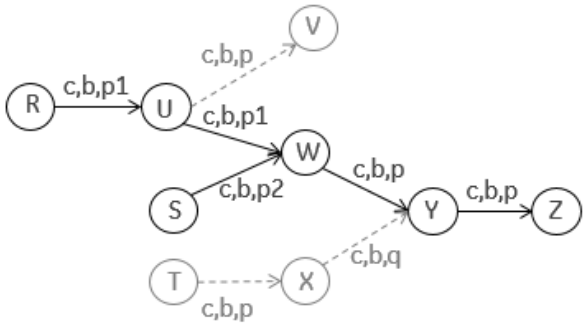


Figure 11: Network tracing for *System Z* (with grayed/dashed segments determined to be irrelevant for the trace)

The tracing routine is recursive, with each iteration dependent on the results from previous iterations. Each iteration evaluates a single edge and determines if its data flows are relevant to the trace based on the data that has been passed through the previous nodes. As such, tracing upstream involves computing the intersection of the data flows on the current edge with the intersection of all the data flows linking the current edge to the downstream node. If the result of the intersection is not empty, the node is included in the trace. If the intersection is empty, then the node in question does not send relevant data to the traced system and thus is not included in the trace results.

Further complexity arises from the fact that the tracing routine must consider the concept and boundary hierarchies. For example, assume that product *p* has subsets *p1* and *p2*. When tracing {*c,b,p*} upstream from a system, it is not sufficient to look for product *p* because the feeding systems may send combinations of *p1* and *p2* that are then merged together to form the complete data for product *p*. Thus, any trace must identify and include any data flows with subsets of the given concepts and boundaries. In the figure, this is illustrated by the fact that Systems U and S are included in the trace because they send data for products *p1* and *p2*, which are subsets of product *p*. Likewise, a system may send a superset of the product of interest, which needs to be included in the trace, as well. The evaluation of each edge can be expressed by the set equation:

$$S_{A1\rightarrow A2} \cap S_{A2\rightarrow T} \neq \emptyset \rightarrow keep \quad (1)$$

where:

$$S_{A2\rightarrow T} = S_{A2\rightarrow A3} \cap S_{A3\rightarrow A4} \cap \ldots \cap S_{An\rightarrow T} \quad (2)$$

and $S_{Ax\rightarrow Ay}$ = set of triples sent from node Ax to node Ay.

Further, a user may choose to apply a filter to the tracing routine, which will limit the trace results to data flows that meet the filter criteria.

The algorithm used for the network tracing is as follows:

1. Identify concepts/boundaries feeding target system T
2. Convert all concepts/boundaries to concept-business-product triples at the lowest hierarchy level
3. Get all systems that feed data to node T, concatenate with name of node T to form a node trace path, and push those trace paths onto empty trace stack
4. Add T to traced network
5. While trace stack is not empty:
   a. Pop trace path ("A") from stack (will take the form A1:::A2:::….:::T)
   b. If path has a loop with one or more nodes between (e.g., A:::B:::C:::B:::D repeats node B) discard path and return to (a) (to avoid recursive loops in paths)
   c. Query for concept-business-product triples that flow from the single-node hop from A1 to A2 in path
   d. Determine whether there is an intersection in the {c,b,p} between A1→A2 and A2→T paths
      i. If YES, add A1 to traced network
      ii. Store the resulting intersection of {c,b,p} triples that flow from A1 all the way to T
      iii. Get names of all systems that feed data to node A1, concatenate with previous node path and push those trace paths onto stack (e.g., A0:::A1:::A2:::….:::T)
      iv. If NO, continue to evaluate next entry on stack

Figure 12: Tracing algorithm

Using the network tracing feature, users can quickly determine the origination points for data feeding a particular system that intersect with the destination's concepts and boundaries. The feature can also be used to trace downstream to destination points in the same manner. Figure 13 shows a sample trace result, for *System A* traced downstream.

7.2 Validation

GE Capital requires a mechanism to perform sanity checks on the network, to identify where it may be incomplete or incorrect. To this end, a validation feature has been implemented. The validation routine works on a per-system basis, determining whether the data that a given system sends is possible given the data that the system creates or receives. Specifically, a system will fail the validation routine if it is found to be sending data that it is neither creating nor receiving.

The validation algorithm consists of identifying the input and output triples for the given system, and checking if each output is contained within the input set. As with the tracing algorithm above, the data must be considered in triples of the form {*c,b,p*} in order to preserve the concept-boundary combinations. Again, further complexity arises from the consideration of the concept and boundary hierarchies. For example, consider Figure 14, assuming that product *p* is comprised of subproducts *p1* and *p2*. *System W* will fail validation, because it only receives subproduct *p1*, which is not enough for it to send product *p*. On the other hand, *System X* will pass validation because it receives subproducts *p1* and *p2*, which are sufficient to product *p*.

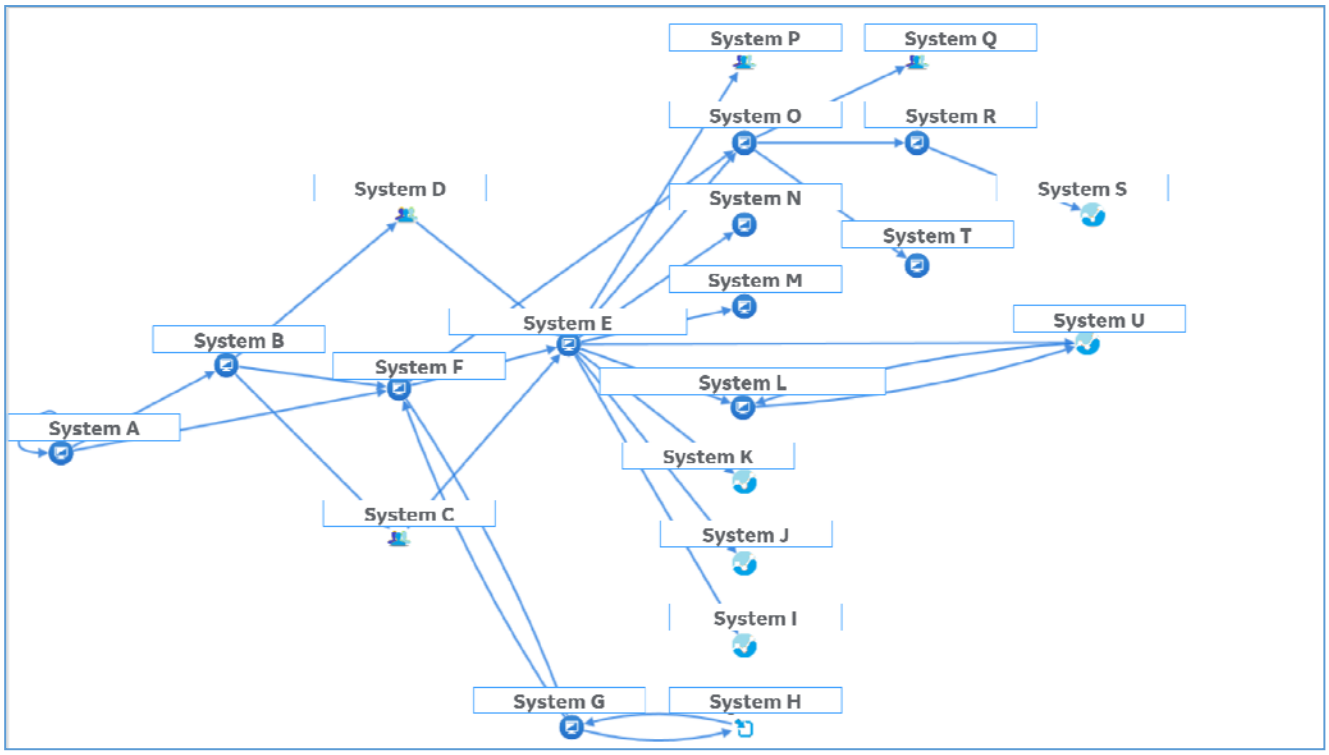


Figure 13: Downstream trace result for *System A*

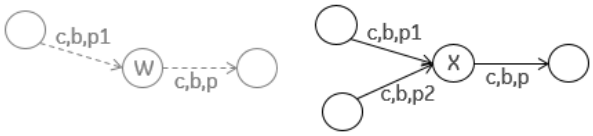


Figure 14: *System W* fails validation because p1 is only a subset of p. *System X* passes validation.

The user can choose to perform validation on a single node, or on all of the nodes displayed in the network. The algorithm for validation is as follows:

For each node *N* to validate:

1. Retrieve all lowest-level concept-business-product triples sent by node *N*
2. Retrieve all lowest-level concept-business-product triples created or received by node *N*
3. Perform an intersection of the triples sent and received and subtract the intersection from the triples sent by *N*
4. If the set of remaining triples sent is not empty, node *N* fails validation, else passes

Figure 15: Validation algorithm

This algorithm can be expressed by the set equation:

$$S_S - (S_C \cup S_R) \neq \emptyset \rightarrow fails \quad (3)$$

where S_S = set of triples sent, S_C = set of triples created, and S_R = set of triples received by node *N*.

In practice, if a system fails validation, this failure is most likely due to a wrong or incomplete network representation (e.g., a downstream data flow has been added to Colt without a corresponding upstream data flow) than a problem in the underlying data systems.

In the future, further validation criteria may be implemented on the network of systems, e.g. to identify redundancies in data creation, transmission, or storage (for data deduplication) across all data stores.

7.3 Optimizations

As a user-facing web-based tool, Colt required a reasonable level of interactivity. To meet this requirement, a number of query optimizations were implemented. Some were achieved by optimizing the SPARQL queries themselves, while others involved parallelizing long running queries through the service tier.

Query performance in SPARQL is largely related to the order of the clauses. SPARQL performs pattern matching on triples to build sets that fulfill the given pattern in the query. As a result, placing the most restrictive clauses earlier in the query decreases the size of intermediate results processed, accelerating the query. For Colt, the main views are driven by the tracing of flows between producer and consumer systems, then applying filters to the discovered flows. Thus, the queries are constructed to first bind the producers and consumers. Queries used for management are more specific to other items, such as concepts of boundary elements, and similarly reprioritize the query clause order.

For network validation and large traces, early query performance was not suitable for interactivity. To improve performance, a service was introduced to run multiple related queries in parallel and then collate the results. To be candidates for this parallelization, the queries' return values must contain some overlapping subset. The calling client provides the query service with a collection of queries to run, the collection of values to return and whether to return values from the non-overlapping sets.

The service executes the collection of queries in parallel and fuses the resulting responses into a single result set. In the Colt use case, the queries sent to the service are partitioned versions of the desired query. By partitioning on consumers or producers, the system can request the entire trace graph at once and then re-order the returned records to be visualized in a coherent manner.

8. RESULTS

Colt has been used extensively by GE Capital since it was first put into production in June 2015. At present, it contains a network of 2,200+ nodes and 1,047 edges (data flows), which represent over 47,000 unique concept-business-product combinations sent between data systems. Colt currently contains 64 concepts, 226 business boundaries, and 125 product boundaries, each organized into 2- or 3-tier hierarchies. The tool's user base is currently at 50 and growing.

With this new tool, GE Capital users are now able to reliably and accurately capture the current state of their data flows, and interactively explore a holistic view of those systems and flows. This capability was not available in the past and represents a substantial improvement in the ability to effectively and efficiently manage GE Capital's data. By requiring documentation of critical data lineage in this way, GE Capital's Chief Data Officer and team has been able to establish a standard of care for data used by the Senior Leadership Team in making strategic decisions about corporate strategies.

Crucially, Colt also allows GE Capital users to answer key questions about systems and flows. With the increased emphasis by financial regulatory oversight boards on stress testing model validation, GE Capital has been able to use the Colt system to provide supporting documentation of the sourcing of data used by such models. By centrally capturing data lineage across the organization, documenting the source of data used by these regulatory models is now a simple effort taking several minutes, rather than a herculean multi-team effort lasting several weeks.

Additionally, Colt is being leveraged by the software development lifecycle process that governs the creation, maintenance, and decommissioning of software applications. By using Colt's network analysis to identify the downstream impact from changes to systems, the approval teams for these projects can efficiently identify projects that require additional oversight.

Overall, GE Capital estimates that Colt will result in a 50% reduction in time and cost of the requirement-building phases of complex IT projects, leading to millions in productivity savings alone. These estimated savings are attributable to Colt's ability to handle complex queries about data movement across the entirety of GE Capital, enabling analysts to efficiently answer regulatory and operational inquiries without having to elicit and fuse this information from a variety of personnel and other potentially non-computable sources. In a recent study conducted in GE Capital UK, 60% of unplanned system outages were a result of ineffective impact analysis—of not understanding the relationships and dependencies between systems.

There were several lessons learned from our experience designing and developing Colt. Utilizing a semantic model to store the metadata proved beneficial, enabling a flexible schema and recursive querying. However, it took several iterations to reach this point. We initially used a rigid model (e.g., no extensible hierarchy) designed to address early system requirements. As the requirements expanded, we encountered challenges extending the model and system, going through three iterations before finalizing the model design. On the plus side, utilizing SADL to create models in structured English significantly reduced our modeling iteration time, allowing us to review the model frequently and directly with GE Capital users.

9. CONCLUSIONS & FUTURE WORK

As organizations struggle to manage more data than ever before, they often lack the ability to answer specific questions about where a given data set originated or how it may have been modified as it flows through an IT infrastructure. In a corporate infrastructure where it is common to find hundreds of systems managing thousands of different types of data, it is a significant challenge to get a cohesive picture of data flows through these systems, or to answer basic questions about the lineage of a given dataset. To address these challenges, we built a system enabling users to model GE Capital data stores and data flows, as well as to interactively explore this information to answer business-critical questions.

Colt represents data stores and other systems as nodes in a directed graph, with edges representing flows of data that are fully characterized by a set of standardized taxonomies. Each edge contains metadata that represents the complete context of the data, including the specific concepts, businesses and products to which each data flow applies. Over 2,200 nodes have been entered to date, with over 1,000 edges describing over 47,000 data flows. Colt includes key analysis features, such as the ability to trace or validate the network. These features and more have made Colt a very powerful system within GE Capital for managing and mining data flow information across the business.

To date, all data flow information has been captured and entered into Colt by hand. While this approach has been effective and provided significant value to GE Capital to date, it is easy to imagine that lack of regular maintenance in the long term could result in the data in Colt becoming incomplete or stale, and no longer reflective of current data flows. In the future, value could be added by auto-discovering the existence of network connections via network packet sniffing, extract-transfer-load (ETL) tool log mining, database query log mining, and similar technologies. Such capabilities could be used to automate the detection of connections between nodes, which can be used to: (i) identify new edges, (ii) confirm existing edges, and (iii) identify stale data in Colt.

10. ACKNOWLEDGEMENTS

The authors would like to thank George Brandman, Rob Casper, John Clark, Daren Clarke, Mark Linehan, Subbaiah Maneyapanda, Roberto Maranca, Glenn Nielsen and Harsh Sharma.

11. REFERENCES

[1] McAfee, A. and Brynjolfsson, E., "Big Data: The Management Revolution", Harvard Business Review, 90 (10): 60-68, Oct, 2012.

[2] The Dodd-Frank Wall Street Reform and Consumer Protection Act, https://www.govtrack.us/congress/bills/111/hr4173/text, Jul. 2010.

[3] Sar, C. and Cao, P. "Lineage File System". Technical Report, Stanford University, Jan 2005.

[4] Muniswamy-Reddy, K., Holland, D. A., Braun, U., Seltzer, M. I. "Provenance-Aware Storage Systems". In Proc. of USENIX Annual Technical Conference, 2006: 43-56.

[5] Gehani, A. and Tariq, D. "SPADE: Support for Provenance Auditing in Distributed Environments", In Proc. of 13th ACM/IFIP/USENIX Middleware Conference, 2012: 101-120.

[6] Yu, J. and Buyya, R. "A Taxonomy of Scientific Workflow Systems for Grid Computing". ACM SIGMOD Record: 34(3): 44-49, 2005.

[7] Missier, P., Belhajjame, K., Zhao, J., Roos, M., Goble, C. "Data Lineage Model for Taverna Workflows with Lightweight Annotation Requirements", In proc. of 2nd International Provenance and Annotation Workshop (IPAW), 2008: 17-30.

[8] Anand, M. K., Bowers, S., McPhillips, T. M., Ludascher, B. "Exploring Scientific Workflow Provenance using Hybrid Queries over Nested Data and Lineage Graphs". In proc. of 21st International Conference on Scientific and Statistical Database Mgmt (SSDBM), 2009: 237-254.

[9] Bose, R. and Frew, J., "Composing Lineage Metadata with XML for Custom Satellite-derived Data Products". In proc. of 16th International Conference on Scientific and Statistical Database Mgmt (SSDBM), 2004: 275-284.

[10] Foster, I., Vockler, J., Wilde M., Zhao Y. "Chimera: A Virtual Data System for Representing, Querying and Automating Data Derivation". In proc. of 14th International Conference on Scientific and Statistical Database Mgmt (SSDBM), 2002: 37-46.

[11] Tomingas K., Tammet T., Kliimask M., Jarv P. "Automating Component Dependency Analysis for Enterprise Business Intelligence". In Proc. of International Conference on Information Systems (ICIS), 2014.

[12] Manta Tools – Manta Flow, https://mantatools.com/manta-flow.

[13] SQLdep: Data Lineage Tool for Data Warehouse Teams, https://sqldep.com/.

[14] Solidatus: Data Lineage and Impact Analysis simplified, https://www.threadneedletechnology.com/solidatus/.

[15] WhereScape RED. https://www.wherescape.com/products-services/ wherescape-red/.

[16] Glavic B. "Big Data Provenance: Challenges and Implications for Benchmarking". In proc. of the First Workshop on Specifying Big Data Benchmarks, 2014: 72-80.

[17] Wang, J., Crawl, D., Purawat, S., Nguyen M., Altintas I. "Big Data Provenance: Challenges, State of the Art and Opportunities". Proc. of the IEEE International Conf. on Big Data, 2015: 2509-2516.

[18] Overview of Teradata Loom Technology, http://blogs.teradata.com/data-points/overview-of-teradata-loom-technology/.

[19] Cloudera Navigator Lineage Diagrams, http://www.cloudera.com/ documentation/enterprise/5-5-x/topics/cn_iu_lineage.html.

[20] Gundecha, P., Ranganath, S., Feng, Z., Liu, H. "A Tool for Collecting Provenance Data in Social Media". In proc. of the 19th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, 2013: 1462-1465.

[21] Ranganath, S., Gundecha, P., Liu, H., "A Tool for Assisting Provenance Search in Social Media". In Proc. of the 22nd ACM Intl. Conf. on Information and Knowledge Management (CIKM), 2013: 2517-2520.

[22] Spillane R., Sears R., Yalamanchili C, Gaikwad S, Chinni M., Zadok E. "Story Book: An Efficient Extensible Provenance Framework. In Proc. of the 1st USENIX Workshop on Theory and Practice of Provenance, 2009.

[23] Linkurious: "How to Track and Visualize Data Lineage", https://linkurio.us/how-to-track-and-visualize-data-lineage/.

[24] Macko P., Seltzer M. "A General-Purpose Provenance Library". Proc. of the 4th USENIX Workshop on Theory and Practice of Provenance, 2012, https://github.com/pmacko86/core-provenance-library.

[25] Simmhan Y. L., Plale B., Gannon S. "Karma2: Provenance Management for Data-driven Workflows". Int. Journal of Web Services Research. 5(2):1-22, 2008.

[26] Moreau L. et al. "The Open Provenance Model: An Overview". Provenance and Annotation of Data and Processes, LNCS 5272: 323-326. 2008.

[27] Missier P., Sahoo S., Zhao J., Goble G., Sheth A. "Janus: From Workflows to Semantic Provenance and Linked Open Data". Provenance and Annotation of Data and Processes, LNCS 6378: 129-141. 2010.

[28] Moreau L et al. "The First Provenance Challenge", Concurrency and Computation: Pract. Exper.: 20(5): 577-586. 2000.

[29] Benjamin, A. R., McClennen, C. E., Santo Domingo, M. G., Dufresne, J. M., Sullivan, C. R. "Data Lineage Management Operation Procedures". US Patent# US9384231 B2, 2016.

[30] Khandelwal A., Walden C., Clarke D., Worley I., Saggu J., Sourekas S., Brizzi S. "Tracking Data Flow in Distributed Computing Systems". US Patent# US20160285701 A1, 2016

[31] Diaku Axon for BCBS 239 Compliance. https://diaku.com/wp-content/uploads/2015/04/AxonForBcbs239Compliance2.pdf, 2015.

[32] Case Study: Collibra Data Governance for a Domestic Significantly Important Bank. https://blog.knowledgent.com/case-study-collibra-data-governance-domestic-significantly-important-bank/, 2016.

[33] Marchant, H. "Exploring Data Lineage: Get a Complete Picture of your Data Flows". IBM developerWorks technical article, 2010.

[34] "Provenance delivers complete data lineage". Bloomberg Provenance Fact Sheet. https://www.bbhub.io/solutions/sites/8/2015/09/Bloomberg_Provenance_Fact_Sheet.pdf.

[35] "Bloomberg is providing critical BCBS-239 compliance support". Bloomberg Ontology and Provenance fact sheet. https://www.bbhub.io/solutions/sites/8/2015/09/Ontology_and_Provenance_BCBS-239_Fact_Sheet.pdf.

[36] W3C RDF, https://www.w3.org/RDF/.

[37] SPARQL, http://www.w3.org/TR/rdf-sparql-query/.

[38] Crapo, A., and Moitra, A. "Toward a Unified English-Like Representation of Semantic Models, Data, and Graph Patterns for Subject Matter Experts". Int. Journal Semantic Computing 7:215. 2013.

[39] Franz, M., Lopes, C.T., Huck, G., Dong, Y., Sumer, O. and Bader, G.D., "Cytoscape.js: a graph theory library for visualization and analysis", Bioinformatics, 32 (2): 309-311, 2016.

[40] Wendt, M. DynaTree.js, http://wwwendt.de/tech/dynatree/.