Exploring a Framework for Identity and Attribute Linking Across Heterogeneous Data Systems

Nathan Wilder
University of Tennessee
Knoxville, Tennessee
nwilder@vols.utk.edu

Jared M. Smith
University of Tennessee
Knoxville, Tennessee
jms@vols.utk.edu

Audris Mockus
University of Tennessee
Knoxville, Tennessee
audris@utk.edu

ABSTRACT

Online-activity-generated digital traces provide opportunities for novel services and unique insights as demonstrated in, for example, research on mining software repositories. The inability to link these traces within and among systems, such as Twitter, GitHub, or Reddit, inhibit the advances in this area. Furthermore, no single approach to integrate data from these disparate sources is likely to work. We aim to design Foreseer, an extensible framework, to design and evaluate identity matching techniques for public, large, and low-accuracy operational data. Foreseer consists of three functionally independent components designed to address the issues of discovery and preparation, storage and representation, and analysis and linking of traces from disparate online sources. The framework includes a domain specific language for manipulating traces, generating insights, and building novel services. We have applied it in a pilot study of roughly 10TB of data from Twitter, Reddit, and StackExchange including roughly 6M distinct entities and, using basic matching techniques, found roughly 83,000 matches among these sources. We plan to add additional entity extraction and identification algorithms, data from other sources, and design tools for facilitating dynamic ingestion and tagging of incoming data on a more robust infrastructure using Apache Spark or another distributed processing framework. We will then evaluate the utility and effectiveness of the framework in applications ranging from identifying malicious contributors in software repositories to the evaluation of the utility of privacy preservation schemes.

CCS Concepts

•Information systems → Data mining; Information extraction; •Software and its engineering → Domain specific languages; Development frameworks and environments;

BIGDSE'16, May 16 2016, Austin, TX, USA

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4152-3/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2896825.2896833

Keywords

Entity Extraction, Entity Identification, Identity Linking, Domain Specific Language, Big Data Architecture

1. INTRODUCTION

Imagine if you could open your browser, head to a website, and type in the name of an entity, whether that is a friend, foe, government, company, or even yourself, and immediately upon hitting a button, see the path of that entity across the entirety of the open internet. Social media accounts, comment histories, notable life events, blog posts, owned domains, personal businesses - all data a user has publicly shared about themselves returned from a name or alias. Foreseer, a framework for identity and attribute linking across heterogeneous data systems, aims to realize that potential. By accessing cross-domain, open data sources through one consistent interface in the form of a Domain Specific Language; the framework should assist in building powerful and quickly-scripted analysis against this collective knowledge. The framework should also be flexible enough for a broad range of identity and user-activity-based applications and a broad range of source data types, such as a real-time API or a static data dump. The Foreseer framework is aimed at very large datasets, so scalability and distributed stability were also architecture requirements. The design calls for a broad set of analysis primitives to help users take advantage of this architecture, and an extensible API for additional user-built modules to accommodate domain or source specific analytic needs.

1.1 Motivation

Limited academic or public research appears to exist for solving the problem of linking entities among heterogeneous data sources. The undeveloped nature should allow us ample room to innovate in this area. Since Foreseer is particularly targeted at intelligence collected and correlated across multiple sources, it has the potential to provide insights over a broader and more comprehensive range than a single source or single method analysis system. For example, to fully understand a person's openly-shared public life, one will probably glean one set of information from their Twitter activities and another set, with non-trivial differences, from their Reddit activities. Each facet of someone's public internet behavior could be added to a collection that would likely provide a more complete picture. The purpose of Foreseer is to provide a framework to perform this linking especially where there may not be a clear connection between identities. It is unlikely for there to be a single way to do this multi-source identity extraction, identification, and linking; therefore, we chose to focus on creating an extensible framework.

We considered a number of applications that might be supported by this framework. Given the assumption that each new link will bring with it an additional set of identity attributes and activity traces, modeling interactive user behavior with multi-sourced data seems to enhance a wide range of tasks. These may include product recommendations based on a user's discussed or demonstrated preferences; personalized search results and text auto-completion that considers not only previous user searches but also any word groupings he or she has publicly used; identity theft prevention by requesting additional authentication factors when current behavior does not match past user behavior; support of anonymous reporting via alternate phrase suggestions that discourage author attribution; privacy assistance by highlighting potentially revealing data; or id of potentially malicious users whose activity matches that of previous trolls, spammers, or malware authors. The specific target application chosen for the evaluation of Foreseer and discussed in this paper is the discovery of vulnerabilities in open source applications and then connection of the entities responsible back to any other online presence they use. This target application can be divided into two tasks. One task is vulnerability discovery, which will be addressed in later phases of this project. The second task is the linking of responsible user identities to accounts they may have on other online systems. Linked accounts may also shed some light upon the intent of any involved developers.

2. RELATED WORK

Although the benefits of cross-domain identity linking appear to be obvious, we were able to find surprisingly few publications on the subject. Most take approaches that focus on a single aspect or method for ascertaining identity. The related work can be roughly subdivided into username-only analysis, social connection analysis, author attribution via text analysis, and multi-aspect analysis through the various elements provided on a user information page (sometimes also called a profile page).

2.1 Username Analysis

Much of the work done focuses on usernames, likely because it relates to identity in an obvious way. One example is Perito et al. [18], who used exact and substring username matching combined with an assessment of the randomness of the username. Some researchers, such as Zafarani et al. [20], used behavioral choices in username selection as the basis for linking, which includes exact matches as well as substring and more general pattern matches (like name length or character range). Perito's team saw accuracy of around 78%, while Zafarani's group achieved 92% accuracy. Zafarani et al. also assessed exact username matching, substring matching, and letter patterns as solitary approaches with accuracy rates of 77%, 63%, and 49% respectively. These methods are limited by the fact that many people will probably be 'invisible', simply because they chose site-differing or very common usernames. Other username disambiguation research considers the username along with similar types of data such as email addresses. In Goeminne et al. [12], the researchers looked at using several different algorithms that used usernames, email addresses, first and last names, and public key certificate data to find duplicate identities in open-source software projects. This type of approach relies heavily on email addresses, which are often not available for many non-software-repository sources. And even when all data is present false positives are a significant issue (and become more problematic the more common a name is).

2.2 Social Connection Analysis

Other researchers took a connection or relationship based route to identity linking. Golbeck et al. [13] used graphs of social connections to link users across sites. Of the roughly 4 million accounts reviewed, about 8,000 had accounts on multiple sites. Combined with limited friendship data this resulted in less than 100 linked accounts for most of the 55 site-to-site pairings but between 100 and 1000 in 7 cases and over 1000 in two cases. In another research endeavor, the activity-profile-based approach described by Mockus [17] was used to identify relationships between mentors and successors. Some of these identified mentors and successor pairs are likely to be the same person using different IDs, while other genuine mentor-successor pairs likely preserve this functional relationship across multiple systems.

2.3 Authorship Analysis

Several researchers have delved into the work necessary to link authors to their works. de Vel et al. [11] attributes emails to their authors with 90.5% average accuracy. Zheng et al. [21] used a long list of features extracted from online messages and was able to achieve accuracy of 97% for English and 88% for Chinese. Caliskan-Islam et al. [8] took on the related problem of linking programmers to their compiled code. Although, Caliskan-Islam's group were able to get accuracy above 90% for programmer groups of up to 20, the accuracy dropped to 51% for a group of 600. While there seems to be value in authorship analysis, text analysis approaches must be considered against its inherent limits. Author discovery requires significant training samples (in size or quantity, preferably both) and the minimum block size of attributable text (for a given confidence level) is not always clear. Also, writing variation across source systems, audiences, and topics does not seem to have been evaluated thoroughly if at all. (de Vel's team did look at a few topics, but got cross-topic individual author results ranging down to 60%.)

2.4 Multi-aspect Identity Analysis

Of all the body of related work, two particular projects are somewhat closer to the types of applications we are aiming to facilitate with Foreseer. The first is analysis done by Malhotra et al. [16] This research assessed likelihood of identity matches across sites by using the presence of various user elements like a profile picture, display name, and description along with analysis tuned to the element type (such as one method with display-names, another for descriptions, and another for images). This team reached 67% accuracy for an identity's highest confidence match and 75% accuracy within the top 3 matches. The second research effort was by Jain et al. [14] This team evaluated user content (such as posted text), attributes (like usernames), and social connections in order to find common identities across Twitter and Facebook. Using an evaluation sample of 543 Twitter users, the group was able to find a correct Facebook account 39% of the time. With each of these research teams, the focus was on the individual analysis methods rather than a framework within which to run them and aggregate their results.

2.5 Data Processing Frameworks

Another way to view work related to Foreseer is from the framework rather than the potential application perspective. The world of big data is populated with numerous frameworks, platforms, and architectures. These tools range from low level processing interfaces to highly abstract, application-specific models. At the lowest level exists the data storage and access platforms, of which Hadoop [6] is probably the most well known. Above this one will find generalized data processing platforms like MapReduce [5], Spark [2], and Storm [3]. Encapsulating the general data processing, one usually finds pipeline and data/query abstraction frameworks such as Crunch [1], Cascading [10], Pig [7], or Hive [4]. And finally at the highest level, toolkits seem to be either methodology based (machine-learning, graphing), media/format based (image, stream processing), or domain-specific (scientific data processing).

The Foreseer framework targets analysis of the user activity traces of an application, with the intention of enabling identity linking. This analysis domain is built with a range of methods, and runs against data in a wide variety of formats. We are not aware of any current frameworks within this division of data processing. The nearest frameworks may be the Semantic Web projects that provide environments and toolkits to link text-based entities over different sources of written data. Volz et al. [15] designed Silk, a Linked Data integration framework. Linked Data is Semantic Web information tagged with standardized labels. Silk uses an XML markup language for configuration. The tool pulls data from its sources using RDF (Resource Description Framework, the metadata format for Linked Data) paths, applies a metric or custom comparison function, and stores matching results. Ceccarelli et al. [9] built the Dexter entity-linking framework. Unlike Silk, this team used a modular design - retrieval modules for each source and system modules for core linking, statistical evaluation, utility functions, and a REST API. Dexter's configuration and control are done via the API. The Dexter linking process involves retrieving data, translating it into a common JSON schema, extracting potential entities, and analyzing known entities that may result in links. One other entity connecting system was developed by Weichselbraun et al. [19] and named Recognyze. Recognyze uses three stages to reach its results - access and query of unstructured data, extraction of named entities using a library of rule-based analyzers, and discovery of links to public Linked Data entities. Despite some similar aspects, these entity linking tools differ in a several ways from Foreseer. They all work with extracting entities (essentially keywords) from text only. This textual data may need to be structured in a standard format (like RDF), and the range of methods supported is only what is needed for these limited value types. These systems do not use (or need) any kind of behavioral or activity-based models.

3. APPROACH

We have targeted the Foreseer framework to support linking identities with various data processing applications. Such applications typically involve data discovery, understanding, and analysis. Foreseer follows that modular design. Data discovery entails collection of information from a desirable source. However, to allow for data and model reuse, we propose that this information should not just be retrieved and stored as static values void of context. Instead the data should be viewed as dynamic traces in a data ecosystem. The stored information should be held with metadata that gives insight into its source provenance, location, direction, and role. Data understanding is the translating or mapping of a source's system model and schema onto a common abstraction. Such an abstraction can allow analysis to be performed against like elements and attributes across sources. And finally data analysis is the evaluation of source knowledge by machine learning and other analytic techniques. This analysis is intended to reveal relationships among specific traces or data groupings. This design leads to a high-level pipeline architecture built with three main components. These three Foreseer components are Seer, Obsidian, and Scry as shown in Figure 1 and described in the following sections.

3.1 Seer

Seer implements data collection operations against a single application or system, such as Twitter. Based on a provided structure, it assists with exploring the structure and actual data generated by a system. And it uses that structure to locally store and provide access to the source traces. Other components may then access this information through Seer. Data analysis workflows often begin with a series of retrieval and database insertion scripts. These scripts are capable of customization to support heterogeneous sources, and may be optimized for the exact data needs of a specific analysis question. However, this approach requires roughly the same effort for each additional data source. It also may need a significant re-write if the analysis approach or target features change. And it pushes an idea of the data as a static entity instead of a dynamic one.

While one might view a single data item, say a tweet, as more or less a static datum once posted. But to generalize that view and see Twitter as just a collection of tweets may miss the rich relationships induced by readers, other entities, and events. Each tweet is information that represents some other data consumed by the author and now transformed into the user's text. That tweet may in turn be read by others who are then inspired to write their own tweets. There are many detectable traces of this data ecosystem beyond the posted content itself such as tweet timestamps, user login or last activity times, parent tweets (for replies), re-tweet meta data, earlier tweets with the same hashtag, and recent trending hashtags. The role and other description of these traces relative to the larger system creates a kind of data ecosystem model. And that model may well be accurate in its structure even if it cannot determine the path of every data particle that traverses it. We suggest that this idea of a data ecosystem can provide a reusable abstraction of each source that may be applied to a wide range of analysis tasks within identity-related data mining.

To facilitate this dynamic view, the Seer component is designed to retrieve, clean, and insert information into local database caches or other parts of the Foreseer process. Information from a source may be collected continuously, pulled periodically with scheduled tasks, or downloaded as needed. Seer sources with their dynamic ecosystem models are defined using the source description portions of the Fore-

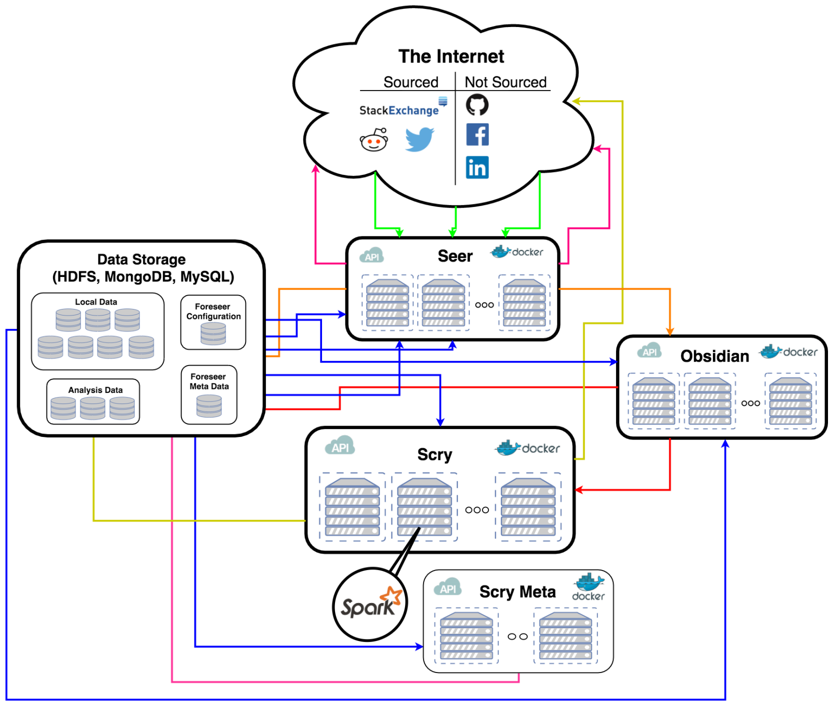

Figure 1: Foreseer Architecture Diagram. [1]

seer language and, like other framework settings, entered by using the JSON configuration format. Complex source tasks may be handled by setting appropriate parameters to refer to user-provided modules (currently Python scripts). Thus, for existing analysis jobs translated into Foreseer, original retrieval scripts may still exist in some form. But these prior scripts should now be able to take advantage of code reuse and should be transformed to collect the data using a dynamic system model.

For the initial iteration of the project, static retrieval scripts were used. These scripts then became starting points for evaluating the ongoing design of Seer. This is our plan for the Seer component and encompasses both some implemented portions and the larger infrastructure still being developed. Once realized, Seer will be able to ingest both stale and streaming data locations for a single source (e.g. a data dump and the Streaming API for Twitter). Seer will merge the data schema from each location (as per the user provided source description) and present a single view to other components. With the data sources we have encountered, there are often many ways to discover their traces, and each way may provide different portions of the total system data. This feature of a unified view should also assist with those sources whose access method results in different users receiving different views of that data. For example, Google search results may differ between two search scripts with different IPs or different API users.

3.2 Obsidian

The dynamic data systems presented by Seer may produce copious amounts of information - all of it with the labels (field/column names, type names, table/collection names, etc) and models (entity models and relationships, explicit links between data entities, etc) of a single source. In order to take that specific information and apply it to analysis working with dissimilar sources, it must first be translated into a common form. This translation is performed by a component called Obsidian. The module maps source-specific labels to common Obsidian ones. For example, one source may have a column called *user*, another *username*, another may use *email* for the same purpose, and another may just use *name*. Obsidian may map them all to a username role. Some source labels may even translate to multiple Obsidian attributes. For example, in some sources the aforementioned *email* may translate to both a username and an email. For now, these translations are manually chosen by analysts as a part of the configuration data that defines a source. However, we also plan for a set of regular expression rules to suggest potential source-specific to general label links. And the meta data of specific analysis attempts (collected via Scry Meta, discussed later) may be used to iterate over potential specific to general translations and suggest the best set of links (perhaps by looking at the outcomes, accuracy, precision, etc. of the applied analysis model). Although it is our expectation that in most cases using meta analysis instead of manual label linking will not be desirable in terms of time and computing resources when compared against any likely outcome gains.

The data translation described above is done with the Obsidian portion of our data description and modeling language. The initial language design consists of a JSON schema that outlines a source's data structures and two sets of tags that perform the data translation - mappings and markers.

[1]For a larger version, see the architecture diagram on the web at http://web.eecs.utk.edu/~jms/foreseer/architecture_diagram.png

Mappings represent a generic modeling of relevant elements within the organization of the source data as well as the attributes that describe those elements. By *organization* here we are referring to the source's grouping of the data into tables, collections, etc. Specific data items within the source that are likely to be extracted and used within Foreseer are also highlighted. This set of tags are called markers. Markers are used to build detailed views of entities independent of the structure of the source or sources from which they are derived. Markers do not necessarily correspond to the organization of data within the originating source.

These concepts may be difficult to visualize; but take for example, the case of a source with two tables - one for users and one for posts. The users table would likely be tagged with a mapping element of *identity* while the posts table would probably be tagged as a *message* element. The users table might have a field of *user* that could be mapped to an attribute of *name* and might also be tagged with a marker of *username*. The posts table might have a field for the date created and another called *user-id*. The *date-created* field would correspond to a mapping attribute of *created* but also a marker of *time-active* (to indicate times the user was active online). The *user-id* field would likely translate to an attribute of *author*. This *author* mapping would not have a direct marker, but instead should refer to the field in the users table that has a mapping attribute of *id* (essentially describing a foreign key relationship). That cross table reference would allow joining together relevant markers in *posts* with those in *users*. To summarize, attributes apply to a Obsidian element that has been mapped from the raw source. On the other hand, markers apply to an identity created from analysis results and located in the Foreseer database. Essentially, Obsidian resolves schema conflicts between two sources by mapping them to the framework data abstraction. Initial versions of Obsidian have already been used to translate and interpret the sources of Twitter, Reddit, and StackExchange.

3.3 Scry

The final stage in the data pipeline is the actual data analysis itself, handled by the Scry component. This framework module uses the analysis portion of our custom language to define processing steps to be run (and re-run) as needed. The initial trials of this component used Obsidian tags within commands. Although source-specific Seer tags are also supported. Seer sources and Obsidian translation are designed to be configured either by the Foreseer command shell or by JSON configuration files. Configuration commands entered in the shell will update the central configuration database. If a JSON file is used, it is opened, parsed and copied into the database when either a Foreseer service is started or a file load command is issued. Unlike Seer and Obsidian, Scry is manipulated via the Foreseer shell; and, as such, the main focus of shell development has been for Scry operations. Scry scripts (rulesets) are built from a series of one or more steps (rules), each issuing a Foreseer command. Once a ruleset is created, specific instances may be started, stopped, and monitored at will. In addition to predefined rulesets, the shell allows direct parsing of Scry commands for on-the-fly analysis tasks.

A range of analysis commands have been planned out, but in the first phase the only command sufficiently implemented to provide reliable results was *match*. *Match* takes three parameters. The first is an array of field references in Seer or Obsidian format of *source-name.location-name.table-name.field-name*. Depending on the source, *table* could also be synonymous with *collection* or *element*; and *field* synonymous with *marker* or *attribute*. Special selector values exist for each identifier position to allow for use of all or a subset of sources. For example, location names can be a specific source location or a format type (i.e. *mysql*, *mongo*, *api*, etc). The second parameter of *match* is an array of restrictions on how the operation is done. These include qualifiers such as *one-to-one* (only include matching of one field to one other), *diff-src-match* (match field values only to field values from other sources, not within a single source), and *limit* (stops matching after some given number of matches are found). The final parameter of *match* is an array of restrictions applied to input data before matching is attempted). These include *limit* (restricts the number of records checked), filter (similar to the *WHERE* $x = y$ SQL statement), and order-by (controls the ordering of items evaluated).

Other Scry commands being implemented include *submatch* (partial value matching), *timematch* (used to evaluate the times that entities are active or have existed), and *wordfreq* (basic word frequency analysis of user created text). Other future commands should include more advanced text analysis such as word order and phrase usage, named entity extraction (finding proper names of people, organizations, and places), social connections graphing, and behavioral analysis beyond basic activity times.

For the initial phase of the project, Scry was implemented using Python modules, and run against the data without any distributed processing. However, our design outlines the use of Apache Spark for processing. Spark has gained a significant amount of traction in the data science community since its inception, and it should enable Scry to process massive data sources in parallel across clusters of machines. Additionally, running the framework on top of Spark should allow us to easily scale out the data processing capacity by adding more servers and worker nodes to the Spark cluster.

Scry also includes a component that we call Scry Meta. Scry Meta is designed to be fed from the Foreseer Meta data store, which is in turn fed from the Seer, Obsidian, and Scry services. Scry Meta (SM) digests information received from these components, such as what mappings were created, what users used what fields, which analysis commands are used by which users, what the outcome metrics of this analysis were, how each framework user labels and comments on elements they create, and other various meta data about the entire Foreseer process. Scry Meta then attempts to learn from this data in order to suggest updates to the Foreseer configuration. These updates might entail adjusting field mappings, prioritizing certain source data elements when pulling from the internet, adjusting performance and scaling settings in the system, and changing other configuration values that may lead to process improvements. Overall, the Scry Meta component should have the potential to allow us or other users to learn how we build tasks using the system and perhaps make the process of analysis more accurate, responsive, and relevant.

3.4 Data Storage

The data storage of Foreseer is key to maintaining analysis results, configurations, meta data on each component's process, and locally-stored raw source data. We use a combination of HDFS, MongoDB, and MySQL; but with data access in code designed in a modular manner that allows extension to support other storage systems. Since Scry utilizes Spark for identity analysis, HDFS is used for its persistent storage. HDFS can store anything from raw JSON or text to Apache Hive tables and serialized Python objects. Currently, HDFS is responsible for storing intermediate analysis data that is still in process, finished analysis data as long as desired, and anything that we may want to cache for computation with Spark. In MongoDB and MySQL we store raw source data (local data in Figure 1), configuration data and long-term result data.

As of now, our unparsed Twitter and Reddit data is stored in MongoDB as BSON (MongoDB's JSON equivalent, i.e. Binary JSON). From there, we do filtering and extraction of the data elements into structured tables in MySQL. We also use MySQL for storing the Foreseer configurations. Every component in the system pulls from the Foreseer Configuration tables to know how to set up their environment, perform processing, chose which fields of raw data to use, and more. This allows us to isolate the framework settings from each component, and more easily build an architecture where each component runs as a service and can pull the most updated configuration values from one central place. Our design permits consistent scaling as the number of Seer, Obsidian, or Scry instances may be increased across many physical or virtual machines without having to worrying about conflicting settings on each instance. Regardless of the user shell or component service updating the configuration, the changes will occur in one physical location. Each component also caches the latest configuration locally for use if the data store for the Foreseer configuration is inaccessible.

In MySQL we also keep the Foreseer Meta tables. These tables store statistics, meta data, and other framework traces that may lead to insights in how users are building on the system and their use of the shell, DSL, and other elements. This data can be pulled and analyzed with the Scry Meta component to attempt to learn how to best adjust the parameters of the entire system to facilitate a more optimized analysis process both overall and per user.

4. RESULTS

For the first phase of this project only the *Match* command was implemented sufficiently to allow for actual data analysis results. The match operation was run against the three initial sources using StackExchange's *DisplayName* field, Reddit's *name* field (in the users table), and both Twitter's *ScreenName* and *Name* fields. For this analysis run, potential matches were not assessed from within each source - only inter-source matches were considered. StackExchange in particular had a large number of duplicate display names due to the fact that no unique constraint was applied and due to StackExchange's structure that utilized a separate user record for each of their sites that a user has ever used. Even among Twitter users there were still a number of duplicates for common names. So, in order to increase the likelihood of quality results, names with more than 10 duplicates per source were excluded from matching. Reddit, on the other hand, enforces uniqueness on their *name* field, so no duplicates were present. Capitalization was ignored for all fields, but non alphanumeric characters were considered.

The number of users usable for each site varied widely.

Table 1: Match results by site (TwSN: Twitter ScreenName; TwN: Twitter Name)

.	**StackEx**	**Reddit**	**TwSN**	**TwN**
Values	3,948,845	241,958	2,492,872	1,985,172
Matches	75308	2734	4237	71071
SE	-	1367	3588	70353
Reddit	1367	-	649	718
Tw SN	3588	649	-	-
Tw N	70353	718	-	-

These numbers are summarized in Table 1. Reddit had 241,958 imported users, all of which had unique names. Twitter had 2,492,872 users all of which were unique in their ScreenName, but only 1,985,172 distinct values were present in the Name field. StackExchange had 8,045,089 DisplayNames, but only 3,948,845 were distinct. The statistics on the final results of this matching run were 83169 matches. Of these, 73941 were between StackExchange and Twitter with 7861 between Twitter and Reddit and 1367 between StackExchange and Reddit. Among the StackExchange and Twitter matches, 70353 came from the Twitter Name field and only 3588 from the Twitter ScreenName field. Among Reddit and Twitter matches, 718 came from Name and 649 from ScreenName.

It seems highly likely, based on these results, that the larger number of matches between StackExchange and the Twitter Name field are due to the lack of unique constraints. The oversized volume of matches suggests that those results may not be of the highest quality. However, spot checking suggests a more nuanced situation. First, some duplicates may be explained by the significant minority of accounts found to be disabled or deleted. This would include older accounts that were removed and then recreated. Second, much of the duplicate clustering on the StackExchange side comes from different StackExchange sites, but actually reflects the same account. On the other hand, most of the twitter duplicates do seem to be different people. However, the impact of this on the value of the Twitter data seems minimal as only one of those users needs to be the same person as the StackExchange user for the match to be valid. Duplicate groupings were counted as a single match in the overall statistics. Besides the enforcement of uniqueness and the larger pools of imported users, the higher numbers for Twitter and StackExchange may also be partly due to differences in the way users approach those sites. Both seem to be utilized by those who are concerned about their online reputations (either professional or personal) and the cultures of both sites seem to encourage sharing real details about oneself. On the other hand, Reddit appears to be heavily attended by those who wish to remain anonymous or at least not share any specific details about their lives.

5. FUTURE WORK

We will continue to develop the key components of the infrastructure, adding novel analysis methodologies and applications. More specifically, we will start from implementing the domain-specific data description and analysis language and proceed to devise approaches for analyzing cross-domain data sources, creating modifiable configurations for the system's behavior, implementing natural language processing and machine learning algorithms more extensively in Scry commands, building a more robust statistics capability into the framework to easily provide confidence scores, and implementing a web front end with result visualizations. Additionally, we will be continuing integration of Apache Spark as the back end to power the analysis primitives in Foreseer DSL and the underlying processing of entities by Scry. Among our targeted features and work to be done is making Foreseer "privacy-respecting". We will attempt to ensure that both open data and custom "plugged-in" data sources ingested by the system anonymize personal identities to uphold the privacy standards required by the domain of the specific data source being ingested. Our ultimate goal will be exploring specific applications that can be built on top of this framework using the linking methods provided, such as analyzing malicious code commits in public repositories and linking the authors back to their information across the web, and analyzing privacy preservation properties of the anonymization algorithms when auxiliary data sources are added.

6. CONCLUSION

In this paper we have presented a framework to support the task of linking identities and attributes among heterogeneous data sources. The three core components of this framework - Seer, Obsidian, and Scry - work together to provide an extensible foundation for building data science applications that should be flexible enough to support project requirements ranging from a researcher's quick academic query to an organization's strategic software systems. Early framework results from simple matching across data sources were not overwhelming, but they provided a reasonable starting point for more innovative analysis and helped guide the framework development. The progress already seen is likely only the beginning of Foreseer's potential, and it will require significant effort and likely traversal of many more technical and theoretical hazards before it is fully realized.

7. REFERENCES

[1] Apache Software Foundation. Apache crunch, January 2016. https://crunch.apache.org/.

[2] Apache Software Foundation. Apache spark - lightning-fast cluster computing, January 2016. https://spark.apache.org/.

[3] Apache Software Foundation. Apache storm, January 2016. https://storm.apache.org/.

[4] Apache Software Foundation. Hive, January 2016. https://hive.apache.org/.

[5] Apache Software Foundation. MapReduce tutorial, January 2016. https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html.

[6] Apache Software Foundation. Welcome to apache hadoop, January 2016. https://hadoop.apache.org/.

[7] Apache Software Foundation. Welcome to apache pig, January 2016. https://pig.apache.org/.

[8] E. D. R. H. K. R. R. G. Aylin Caliskan-Islam, Fabian Yamaguchi and A. Narayanan. When coding style survives compilation: De-anonymizing programmers from executable binaries. 2015.

[9] D. Ceccarelli, C. Lucchese, S. Orlando, R. Perego, and S. Trani. Dexter: An open source framework for entity linking. In *Proceedings of the Sixth International Workshop on Exploiting Semantic Annotations in Information Retrieval*, ESAIR '13, pages 17–20, New York, NY, USA, 2013. ACM.

[10] Concurrent, Inc. Cascading | application platform for enterprise big data, January 2016. http://www.cascading.org/.

[11] O. de Vel, A. Anderson, M. Corney, and G. Mohay. Mining e-mail content for author identification forensics. *SIGMOD Rec.*, 30(4):55–64, Dec. 2001.

[12] M. Goeminne and T. Mens. A comparison of identity merge algorithms for software repositories. *Sci. Comput. Program.*, 78(8):971–986, Aug. 2013.

[13] J. Golbeck and M. Rothstein. Linking social networks on the web with foaf: A semantic web case study. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, AAAI'08, pages 1138–1143. AAAI Press, 2008.

[14] P. Jain, P. Kumaraguru, and A. Joshi. @i seek 'fb.me': Identifying users across multiple online social networks. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13 Companion, pages 1259–1268, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[15] M. G. Julius Volz, Christian Bizer and G. Kobilarov. Silk âĂŞ a link discovery framework for the web of data. In *2nd Workshop about Linked Data on the Web*, April 2009.

[16] A. Malhotra, L. Totti, W. Meira Jr., P. Kumaraguru, and V. Almeida. Studying user footprints in different online social networks. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, ASONAM '12, pages 1065–1070, Washington, DC, USA, 2012. IEEE Computer Society.

[17] A. Mockus. Succession: Measuring transfer of code and developer productivity. In *2009 International Conference on Software Engineering*, Vancouver, CA, May 12–22 2009. ACM Press.

[18] D. Perito, C. Castelluccia, M. A. Kaafar, and P. Manils. How unique and traceable are usernames? In *Proceedings of the 11th International Conference on Privacy Enhancing Technologies*, PETS'11, pages 1–17, Berlin, Heidelberg, 2011. Springer-Verlag.

[19] A. Weichselbraun, D. Streiff, and A. Scharl. Linked enterprise data for fine grained named entity linking and web intelligence. In *Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14)*, WIMS '14, pages 13:1–13:11, New York, NY, USA, 2014. ACM.

[20] R. Zafarani and H. Liu. Connecting users across social media sites: A behavioral-modeling approach. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 41–49, New York, NY, USA, 2013. ACM.

[21] R. Zheng, J. Li, H. Chen, and Z. Huang. A framework for authorship identification of online messages: Writing-style features and classification techniques. *J. Am. Soc. Inf. Sci. Technol.*, 57(3):378–393, Feb. 2006.