VINERy: A Visual IDE for Information Extraction

Yunyao Li
IBM Research - Almaden
650 Harry Road
San Jose, CA 95120
yunyaoli@us.ibm.com

Elmer Kim*
Treasure Data, Inc.
201 San Antonio Cir
Mountain View, CA 94040
elmer.k.kim@gmail.com

Marc A. Touchette
IBM Silicon Valley Lab
555 Bailey Avenue
San Jose, CA 95141
matouche@us.ibm.com

Ramiya Venkatachalam
IBM Silicon Valley Lab
555 Bailey Avenue
San Jose, CA 95141
venkatra@us.ibm.com

Hao Wang
IBM Silicon Valley Lab
555 Bailey Avenue
San Jose, CA 95141
haowang@us.ibm.com

ABSTRACT

Information Extraction (IE) is the key technology enabling analytics over unstructured and semi-structured data. Not surprisingly, it is becoming a critical building block for a wide range of emerging applications. To satisfy the rising demands for information extraction in real-world applications, it is crucial to lower the barrier to entry for IE development and enable users with general computer science background to develop higher quality extractors.

In this demonstration[1], we present VINERY, an intuitive yet expressive visual IDE for information extraction. We show how it supports the full cycle of IE development without requiring a single line of code and enables a wide range of users to develop high quality IE extractors with minimal efforts. The extractors visually built in VINERY are automatically translated into semantically equivalent extractors in a state-of-the-art declarative language for IE. We also demonstrate how the auto-generated extractors can then be imported into a conventional Eclipse-based IDE for further enhancement. The results of our user studies indicate that VINERY is a significant step forward in facilitating extractor development for both expert and novice IE developers.

1. INTRODUCTION

Information Extraction (IE) is the task of automatically extracting structured information from unstructured or semi-structured text. Programs performing information extraction, also known as *annotators* or *extractors*, are becoming the foundation for a wide range of emerging enterprise applications, such as social data analytics, patient record analytics, and financial risk analysis.

Extractor development has been a daunting task for many due to its high barrier to entry and steep learning curve. To be able to start developing extractors, one has to spend days, if not weeks, to acquire working knowledge of machine learning models for IE (e.g. CRF/HMM) or learn a non-trivial rule language (e.g. AQL in SystemT [10], JAPE in GATE [5], and UIMA Ruta [9]). As a result, extractor development remains a major bottleneck in satisfying the increasing demands of real-world applications based on IE. Hence, lowering the barrier to entry for extractor development becomes a critical requirement.

Previous work on improving the usability of IE systems has mainly focused on reducing the manual effort involved in extractor development [3, 6, 11, 12, 16, 17] with a few exceptions. WizIE [14, 18] lowers the extractor development entry barrier with a wizard-like environment that guides extractor development based on best practices drawn from expert developers. However, users still need to learn a non-trivial rule language, which requires at least hours of training. Special-purpose systems such as [1, 2] are effective in enabling novice users to develop extractors with minimal learning requirement but are of limited practical use due to their limited expressivity.

In this demonstration, we present VINERY, a **V**isual **IN**tegrated Development **E**nvironment for Information ext**R**action. It was designed to lower the barrier to entry for practical extractor development and enable a wide range of users to develop high quality extractors for real-world extraction tasks. VINERY consists of the following key elements: (1) The foundation of VINERY is VAQL, a visual programming language for information extraction, designed based on AQL [10], a state-of-the-art declarative language for information extraction. VAQL provides a visual abstraction for the core functionalities of AQL to ensure both expressivity and simplicity of VINERY. (2) VINERY embeds the visual constructs of VAQL in an intuitive web-based visual IDE for constructing extractors. Visual extractors created via VINERY are automatically translated into AQL and executed by the corresponding runtime engine [7] for scalability and efficiency. If needed, the auto-generated extractors in AQL can be imported into a conventional development environment(e.g. [13]) for further enhancement. (3) VINERY includes a rich set of easily customizable pre-built extractors to help jump-start extractor development. (4) VINERY provides features to support the entire life cycle of extractor development. VINERY is available as part of [8].

To the best of our knowledge, VINERY is the first visual IDE for information extraction for practical extraction tasks. Our preliminary user studies show that both novice and expert developers can benefit from VINERY — minimal effort is required to develop high-quality extractors within the UI, with merely minutes, as opposed to hours or even days, of learning required.

*Work done while at IBM

[1]Demo recording available at https://vimeo.com/131232302

Proceedings of the VLDB Endowment, Vol. 8, No. 12
Copyright 2015 VLDB Endowment 2150-8097/15/08.

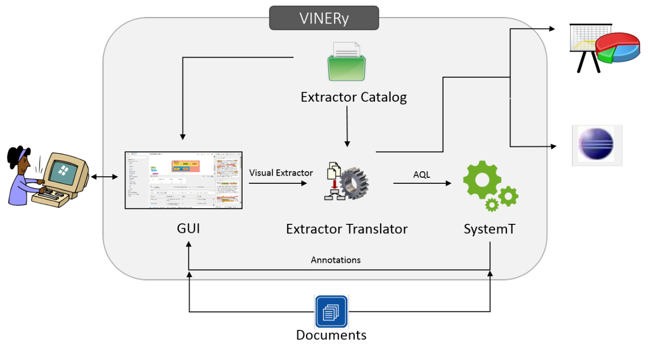


Figure 1: System Overview of VINERy

The rest of the paper is organized as follows. Sect. 2 describes the architecture of VINERY. Sec. 3 and Sec. 4 present VAQL and the key features included in the demonstration respectively. Sec. 5 summarizes the results of our preliminary user studies with VINERY. Then in the final section, we outline the demo scenario that we will use to show the effectiveness of VINERY.

2. SYSTEM OVERVIEW

Fig. 1 presents a high-level overview of the system. As shown, a user can visually construct extractors with a rich set of constructs in the UI (detailed in Sec. 3) and pre-built extractors from the *Extractor Catalog*. The visual representations of the extractors are automatically translated into performant IE code in AQL. The users can execute extractors with the underlying SystemT execution engine against an input document collection. They may analyze the execution results and further refine the visual extractors to their satisfaction. They can then either publish the extractors for deployment or, in the cases where the user may need functionalities not yet supported by VINERY (e.g. a user-defined function), they can import the auto-generated IE code into a conventional IE development environment (e.g. [13]) for further development.

3. VAQL

The core of VINERY is VAQL, designed to capture the core functionality of the underlying AQL language [15] using the same algebraic approach. In this section, we describe the main constructs of VAQL. VINERY supports two types of extractors: (1) *Pre-built* extractors refer to pre-defined extractors that work as black boxes, with a pre-defined set of customizable dictionaries; (2) *User-built* extractors are those constructed in VINERY using one or more of the following visual constructs.

Extract constructs perform extraction over the input data, including

- **Pre-built** Extracts matches using a pre-built extractor.
- **Dictionary** Extracts matches for a dictionary that consists of a list of terms or pairs of terms.
- **Regular Expression** Extracts matches for one or more regular expressions.
- **Literal** Extracts matches for a single string.
- **Sequence Pattern** Extracts matches based on a user-specified pattern.
- **Proximity** Extracts a range of tokens.

Select constructs define the output schema of the extraction results.

- **Projection** Outputs selected attributes (aka. output columns) from the input of the current extractor or from another extractor.
- **Expression** Outputs an attribute with a fixed string value or the results of a *scalar* function, where a scalar function performs an operation based on the values of other output columns from the same extractor.
- **Consolidation** Handles overlapping matches based on a user specified consolidation policy.

Filter constructs refine the extractor to filter out unwanted results.

- **Inclusive/Exclusive** Keeps/removes any extraction result that conforms to the filter.

Union merges output from two or more extractors.

Fig. 2 illustrates a *PersonPhone* relation extractor constructed using aforementioned constructs. As can be seen from Fig. 2(a), each extract construct is visualized as a box labeled with the name of the extractor (e.g. ⟨`Person`⟩ and ⟨`PhoneNumber`⟩). A sequence pattern is visualized as a nested box with a sequence of boxes (e.g. ⟨`Phone Number 1`⟩ and ⟨`Phone Number 2`⟩), while a union is visualized as a nested box with in one-level of child boxes (e.g. ⟨`Person Phone`⟩). A nested box can be expanded or collapsed to control the level of details shown for each extractor. For instance, all boxes in Fig. 2(a) are expanded to show full details. Additional details about each extractor (i.e. select and filter constructs) are included as editable properties for the extractor (e.g. Fig. 2(b)).

4. VISUAL EXTRACTOR DEVELOPMENT

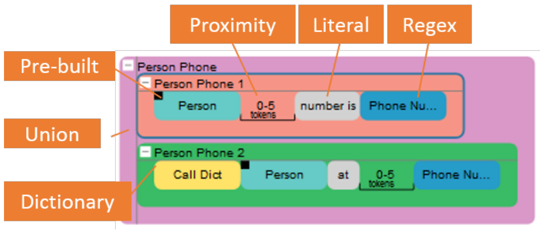
Fig. 3 depicts the key UI features of VINERY designed to support the entire life cycle of extractor development.

Project Management Each project in VINERY consists of a set of extractors and an input document collection. The *Project Panel* (Fig. 3(1)) enables a user to create a new project and load/delete/modify an existing project.

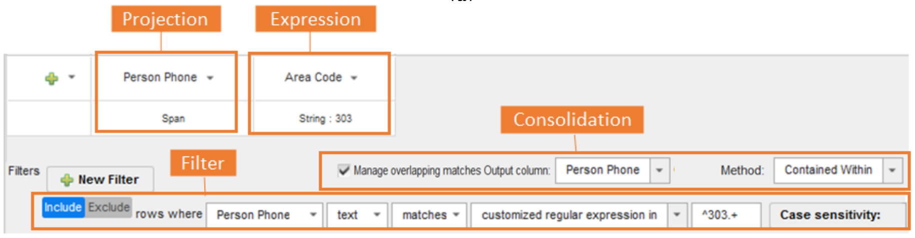
Document Management *Document Viewer* (Fig. 3(2)) allows a user to add/remove documents from local file system or HDFS for extraction. Each document may be displayed in *snippet view*, where snippets of multiple documents are displayed in the panel, or *full view*, where only one single document is displayed. For documents with tags (e.g. HTML/XML documents), the user can also turn on/off the detag option.

Extractor Construction VINERY is designed to enable the visual construction of extractors. *Canvas* (Fig. 3(4)) is the main workspace for extractor construction. The user can create atomic extractors (without nested boxes) by adding them directly on the canvas via a context menu (e.g. dictionaries) or by drag-and-dropping pre-built extractors from the *Extractor Catalog* (Fig. 3 (3)). A rich set of pre-built extractors for a wide range of domains (Fig. 3(1)) are provided to help jump start extractor development. The user can also create composite extractors (with nested boxes) by dragging multiple extractors together in sequences or unions. The *Property Pane* (Fig. 3(5)) allows the user to further refine each extractor. It has three tabs: (1) *General* tab displays descriptions about the extractor; (2) *Settings* tab shows properties that can be modified to refine the extractor's semantics (e.g. adding/removing a term from a dictionary or customize a pre-built extractor); and (3) *Output* tab allows the user to control the extractor output: The user can add /remove output columns, add filters to refine the extractor behavior, and/or specify consolidation to handle overlapping matches.

Extractor Execution and Results Exploration The user can select one or more extractors on the Canvas to execute over the input documents. Results of the execution are then displayed in the *Results Grid* (Fig. 3(6)) as well as highlighted in the Document Viewer. The Results Grid consists of a set of tabs corresponding to the set of extractors executed, each with a view of the output annotations



(a)

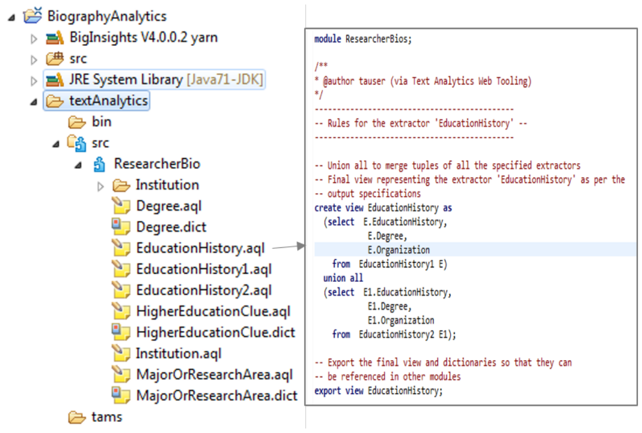

Projection
Expression
Person Phone
Span
Area Code
String : 303
Consolidation
Filter
Filters
New Filter
Manage overlapping matches Output column: Person Phone
Method: Contained Within
Include Exclude rows where Person Phone text matches customized regular expression in ^303.+ Case sensitivity:

(b)

Figure 2: Example *PersonPhone* Extractor

or results. When a tuple in the Results Grid is clicked on, the corresponding results are shown and selected in the Document Viewer. Results can also be exported as .csv files.

Publish and Deployment Once the user is happy with the extractor, she can save an extractor built via the Canvas to the Extractor Catalog. She can also deploy any extractor from the catalog on a cluster or export it for deployment in downstream applications or into a conventional IDE (e.g. [13]) for further development. The figure below shows a sample Eclipse project with auto-generated AQL extractors imported from VINERY along with a sample AQL file.



```
module ResearcherBios;

/**
* @author tauser (via Text Analytics Web Tooling)
*/
---------------------------------------------
-- Rules for the extractor 'EducationHistory' --
---------------------------------------------

-- Union all to merge tuples of all the specified extractors
-- Final view representing the extractor 'EducationHistory' as per the
-- output specifications
create view EducationHistory as
  (select  E.EducationHistory,
           E.Degree,
           E.Organization
    from  EducationHistory1 E)
  union all
  (select  E1.EducationHistory,
           E1.Degree,
           E1.Organization
    from  EducationHistory2 E1);

-- Export the final view and dictionaries so that they can
-- be referenced in other modules
export view EducationHistory;
```

5. USER STUDIES

We conducted preliminary user studies to evaluate the usability and effectiveness of VINERY in assisting both expert and novice users.

Expert Users: This informal study included four experienced IE developers with brief experience of using VINERY for IE development prior to the study. All employed at IBM with moderate to extensive experience with IE. In the pre-study, for the question "*What is your experience with Text Analytics*", the median rating was 4, on a scale of 0 (no experience) to 5 (expert).

We interviewed the participants with a pre-defined set of questions adapted from SUS usability survey [4]. Overall, the participants found that VINERY was much better than other IE development tools in terms of *learning curve*, *ease of use*, and *development time and effort required*. For the question "*In general, how does* VINERY *compare to the way you and other currently perform IE tasks?*", on a scale of 1 (much worse) to 5 (much better), the median rating was 4.25 on *learning curve*, and 4.75 on both *development time required* and *ease of use*, 5 on *effort required*.

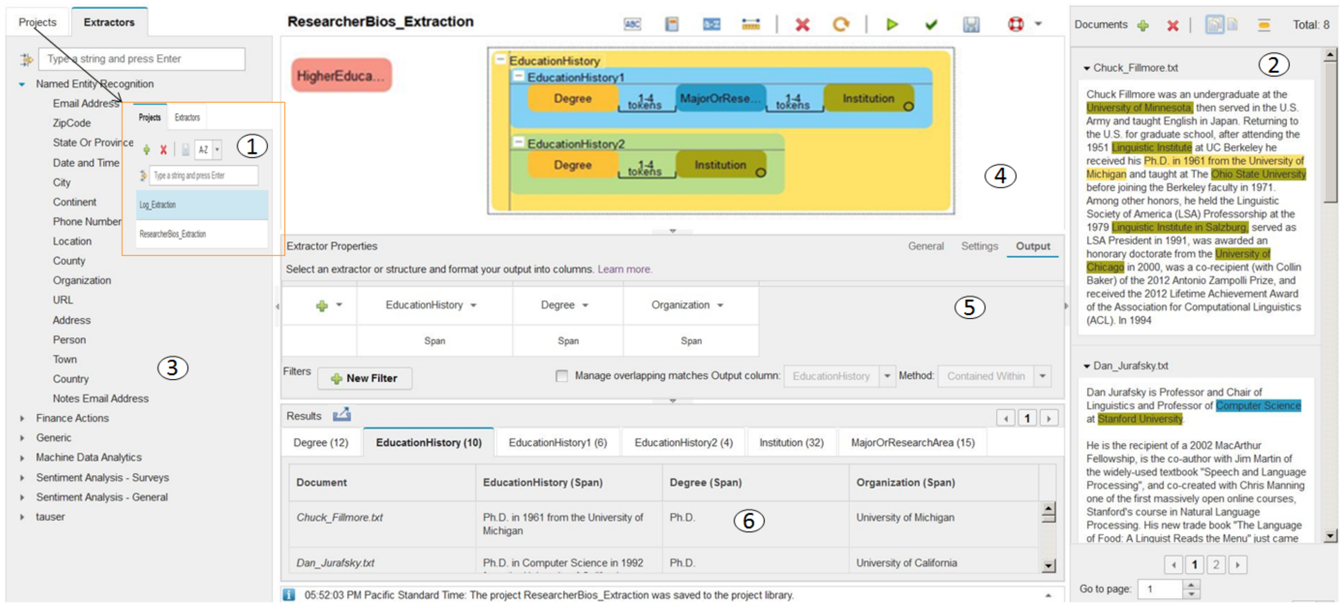
The participants also liked the main features of VINERY(described in Sec. 4). For the question "*How much do you like the main features of* VINERY", the responses were overall positive: on a scale of 1 (not at all) to 5 (very much), the median rating ranging from 3+ for *property pane* and *project management* to 4+ for others.

Novice Users: This study included 10 participants with minimal prior experience with IE, all employed at IBM. In the pre-study, for the question "*What is your experience with Text Analytics*", the median rating was 1, on a scale of 0 (no experience) to 5 (expert).

In a 60-minute session, participants were asked to develop extractors for two tasks: `Task1`: Identify mentions of company revenue by division from the company's official press releases; `Task2`: Identify education history from biographies (see Fig. 3). Authors of this paper were present to help participants during the session. After the session, participants filled out a survey about their experience.

All participants, except for one, completed both tasks under 60 minutes. On average `Task1` took 29 minutes and `Task2` took 23.6 minutes. It is worthy noting that `Task1` was adapted from the lab comparative study by [18]. In that study, the participants first performed `Task1` without WIZIE on Day 1 and then completed the same task in 90 minutes or less using WIZIE on Day 2. The participants in our study had no prior exposure to the task. However, with the help of VINERY they were able to complete the same task about trice as fast on average as the participants using WIZIE, but with much less training (minutes vs. days). This result is a strong indication that VINERY is a significant step forward in enabling extractor development for novice users.

To evaluate the usability of VIPER, in a post-study survey,we used questions adapted from SUS usability survey [4] to gather the participants' opinions about the system. Overall, the participants found the system useful and easy to learn. On a scale of 1 (Strongly Disagree) to 5 (Strongly Agree), the median rating was 4.0 for both the questions "*I would like to use the system frequently and would recommend it to others*" and "*The system was easy to use*", and 2.0



Chuck_Fillmore.txt

Chuck Fillmore was an undergraduate at the University of Minnesota, then served in the U.S. Army and taught English in Japan. Returning to the U.S. for graduate school, after attending the 1951 Linguistic Institute at UC Berkeley he received his Ph.D. in 1961 from the University of Michigan and taught at The Ohio State University before joining the Berkeley faculty in 1971. Among other honors, he held the Linguistic Society of America (LSA) Professorship at the 1979 Linguistic Institute in Salzburg, served as LSA President in 1991, was awarded an honorary doctorate from the University of Chicago in 2000, was a co-recipient (with Collin Baker) of the 2012 Antonio Zampolli Prize, and received the 2012 Lifetime Achievement Award of the Association for Computational Linguistics (ACL). In 1994

Dan_Jurafsky.txt

Dan Jurafsky is Professor and Chair of Linguistics and Professor of Computer Science at Stanford University.

He is the recipient of a 2002 MacArthur Fellowship, is the co-author with Jim Martin of the widely-used textbook "Speech and Language Processing", and co-created with Chris Manning one of the first massively open online courses, Stanford's course in Natural Language Processing. His new trade book "The Language of Food: A Linguist Reads the Menu" just came

Extractor Properties — General | Settings | Output

Select an extractor or structure and format your output into columns. Learn more.

	EducationHistory	Degree	Organization
	Span	Span	Span

Filters: New Filter — Manage overlapping matches Output column: EducationHistory Method: Contained Within

Results: Degree (12) | EducationHistory (10) | EducationHistory1 (6) | EducationHistory2 (4) | Institution (32) | MajorOrResearchArea (15)

Document	EducationHistory (Span)	Degree (Span)	Organization (Span)
Chuck_Fillmore.txt	Ph.D. in 1961 from the University of Michigan	Ph.D.	University of Michigan
Dan_Jurafsky.txt	Ph.D. in Computer Science in 1992	Ph.D.	University of California

05:52:03 PM Pacific Standard Time: The project ResearcherBios_Extraction was saved to the project library.

Figure 3: Snapshot of VINERY GUI: (1) Project Pane; (2) Document Viewer; (3) Extractor Catalog; (4) Canvas; (5) Property Pane; (6) Result Grid.

for the question "*I need to learn a lot of things to use the system.*".

The participants also unanimously found all the features discussed in Sec. 4 very useful. For the question "*How easy are the main features*", the responses were overwhelming positive: on a scale of 1 (Very Difficult) to 5 (Very Easy), the median rating was 4.5+ for all the features. One participant with brief experience of IE development even wrote that "[Compared to what he has used before] *This is 100 times better.*"

The participants felt confident in using VINERY to create extractors, despite their minimal experience in IE development. Meanwhile, they also suggested possible improvements to the system, such as adding more contextual help and a step-by-step graphic guide to extractor development.

Summary: Our preliminary results indicate that VINERY is a major step forward towards facilitating extractor development, particularly for novice users. We plan to conduct a formal study of using VINERY to create extractors for several real business applications after usability and feature enhancements based on user feedback.

6. DEMONSTRATION

We have implemented VINERY as a stand-alone web application. In this demonstration[1] we showcase the entire extractor development life cycle in VINERY with features described in Sec. 4, from project creation to the incremental and iterative process of extractor development, and the final deployment of extractors. Our demonstration centers around the task of identifying education history from biographies of DB researchers. We start by demonstrating the process of developing three atomic extractors for identifying *Degree*, *Major* and *Institution* mentions. We then show how to put them together to construct a more complex extractor for identifying mentions of *EducationHistory*. We will also illustrate how to deploy the constructed the extractor in a downstream analytics application as well as how to continue development based on the actual IE code automatically generated behind the scenes in a professional IDE [13].

7. REFERENCES

[1] A. Akbik et al. Propminer: A workflow for interactive information extraction and exploration using dependency trees. In *ACL*, 2013.

[2] A. Akbik et al. Exploratory relation extraction in large text corpora. In *COLING*, 2014.

[3] F. Brauer et al. Enabling information extraction by inference of regular expressions from sample entities. In *CIKM*, 2011.

[4] J. Brooke. SUS-a quick and dirty usability scale. *Usability evaluation in industry*, 189:194, 1996.

[5] H. Cunningham et al. Gate: an architecture for development of robust hlt applications. In *ACL*, 2002.

[6] S. Gupta and C. D. Manning. Spied: Stanford pattern-based information extraction and diagnostics. In *ACL-ILLVI*, 2014.

[7] IBM. Gumshoe, Midas, SIMPLE, etc. 2012.

[8] IBM. InfoSphere BigInsights. https://ibm.biz/bdfhnd, 2015.

[9] P. Kluegl et al. UIMA Ruta Workbench: Rule-based text annotation. In *COLING*, 2014.

[10] R. Krishnamurthy et al. SystemT: a system for declarative information extraction. *SIGMOD Record*, 37(4):7–13, 2008.

[11] Y. Li et al. Regular expression learning for information extraction. In *EMNLP*, 2008.

[12] Y. Li et al. Facilitating pattern discovery for relation extraction with semantic-signature-based clustering. In *CIKM*, 2011.

[13] Y. Li et al. SystemT: A Declarative Information Extraction System. In *ACL (Demonstration)*, 2011.

[14] Y. Li et al. WizIE: a best practices guided development environment for information extraction. In *ACL*, 2012.

[15] F. Reiss et al. An algebraic approach to rule-based information extraction. In *ICDE*, 2008.

[16] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, February 1999.

[17] B. R. Soundrarajan et al. An interface for rapid natural language processing development in UIMA. In *ACL (Demonstrations)*, 2011.

[18] H. Yang et al. I can do text analytics!: designing development tools for novice developers. In *CHI*, 2013.