The SystemT IDE: An Integrated Development Environment for Information Extraction Rules

Laura Chiticariu Vivian Chu Sajib Dasgupta Thilo W. Goetz[1] Howard Ho
Rajasekar Krishnamurthy Alexander Lang[1] Yunyao Li Bin Liu[2]*
Sriram Raghavan[3]† Frederick R. Reiss Shivakumar Vaithyanathan Huaiyu Zhu
IBM Research – Almaden [1]IBM Software – Germany [2]University of Michigan [3]IBM Research – India

ABSTRACT

Information Extraction (IE) — the problem of extracting structured information from unstructured text — has become the key enabler for many enterprise applications such as semantic search, business analytics and regulatory compliance. While rule-based IE systems are widely used in practice due to their well-known "explainability," developing high-quality information extraction rules is known to be a labor-intensive and time-consuming iterative process.

Our demonstration showcases SystemT IDE, the integrated development environment for SystemT, a state-of-the-art rule-based IE system from IBM Research that has been successfully embedded in multiple IBM enterprise products. SystemT IDE facilitates the development, test and analysis of high-quality IE rules by means of sophisticated techniques, ranging from data management to machine learning. We show how to build high-quality IE annotators using a suite of tools provided by SystemT IDE, including computing data provenance, learning basic features such as regular expressions and dictionaries, and automatically refining rules based on labeled examples.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Textual Databases; I.2.7 [**Natural Language Processing**]: Text Analysis

General Terms

Algorithms, Design, Human Factors

Keywords

Information Extraction, AQL, SystemT, Provenance, Pattern Discovery, Rule Learning

1. INTRODUCTION

Information Extraction (IE) — the problem of extracting structured information from unstructured text — has emerged as a critical building block to many enterprise applications. The value of information extraction in an enterprise application lies in its ability to generate extremely accurate results and scale to large volumes of data. In addition, the use of IE in these applications drives the need for usability [4] in terms of ease of development and maintenance is an important requirement, as developing high-quality information extraction rules (or *annotators*) is notoriously labor-intensive and time-consuming.

Challenges in developing IE rules. The development of high-quality IE rules typically consists of multiple iterations of three phases: *development*, *testing*, and *analysis* [3]. Each of these phases can be an expensive manual process.

Consider, for example, the task of extracting relationships between persons and their phone numbers. While this may sound simple, developing an accurate PersonPhone annotator is challenging in practice. (The PersonPhone annotator from SystemT's Annotator Library consists of over 300 rules.)

The *development* phase involves writing complex regular expressions to identify syntactic features (e.g., phone numbers and capitalized words), collecting dictionaries (e.g., lists of common first and last names), as well as writing rules to combine these basic features into larger concepts (e.g., full names). For instance, a rule for identifying candidate person names may look for *"a match of a dictionary of first names followed immediately by a match of a regular expression identifying capitalized words"*. The development of each rule may involve the following two phrases.

In the *testing* phase, the annotators are executed on a collection of documents and the developer manually examines the extraction results (i.e., *annotations*) – potentially thousands of them – to determine their correctness and identify expected annotations that are missing.

In the *analysis* phase, the developer seeks to understand the causes of the mistakes, i.e., the incorrect and missing annotations. For example, suppose the text "`Morgan Stanley, fax: 205-4493`" is identified as an instance of PersonPhone relationship. The developer will need to identify the rule that incorrectly identifies `Morgan Stanley` as a person, and find out why the rules for identifying fax numbers in order to subtract them from the set of phone candidates do not capture `205-4493`. With annotators comprising of hundreds of rules with complex interactions, answering such questions is far from trivial. Furthermore, once the causes are identified, the developer must determine concrete refinements to the existing rules to correct the mistakes in the next iteration of the process. Typically, a large number of candidate refinements may exist. For instance, to avoid identifying `Morgan Stanley` as a person, one may remove `Morgan` from

*Work partially done while at IBM Research – Almaden.
†Work done while at IBM Research – Almaden.

SIGMOD'11, June 12–16, 2011, Athens, Greece.
ACM 978-1-4503-0661-4/11/06.

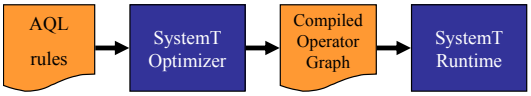

Figure 1: The architecture of SystemT.

the dictionary of first names, or remove `Stanley` from the dictionary of last names, or even add a dictionary of organization names and use it to filter candidate persons. This is typically a manual "trial and error" process, where multiple candidate refinements are implemented and evaluated.

Demonstration Focus. In this demonstration, we showcase SystemT IDE, the integrated development environment for IE rules. SystemT[1] [2, 5] is a state-of-the-art rule-based IE system developed in IBM Research and successfully embedded in multiple IBM enterprise offerings, for a variety of extraction tasks: identifying named entities (e.g., person, organization, location), relationships between these entities (e.g., a person's phone or address), financial events (e.g., company mergers and acquisitions), opinions and sentiments, and many more. Figure 1 illustrates the architecture of SystemT, which consists of *AQL* [8], a declarative language for expressing IE rules, a *cost-based optimizer* for compiling AQL rules into an operator graph (i.e., execution plan), and a *runtime engine* for efficiently executing the operator graph on an input document collection and generate output annotated documents. Rule development in SystemT is an iterative process as described earlier. The SystemT IDE makes use of sophisticated techniques to assist rule developers throughout all stages of the development cycle.

2. OVERVIEW OF THE SystemT IDE

The architecture of SystemT IDE is depicted in Figure 2. It consists of the following six main components. (1) The *AQL Editor* provides essential editing capabilities such as syntax highlighting for AQL rules. (2) The *Annotation Viewer* allows one to examine output annotations, as well as compare them with a gold standard, or with the results of a previous version of the annotator, to avoid regressions. (3) The *Annotation Provenance Viewer* generates and displays the *provenance* of annotations, which allows the developer to quickly discover the annotator's components responsible for a mistake, and focus the debugging process accordingly. (4) The *Contextual Clue Discoverer* speeds up the process of collecting and discovering important contextual clues from the input document collection. (5) The *Regular Expression Learner* takes as input a user-specified (simple) regular expression and produces a more precise (and more complex) regular expression, thus relieving the developer from the burden of manually writing complex regular expressions from scratch. (6) Finally, the *AQL Rule Refiner* automatically generates a set of concrete refinements ranked according to how much each improves the overall accuracy of the annotator, given a set of labeled output annotations.

To the best of our knowledge, the SystemT IDE is the first integrated development environment for IE rules that significantly simplifies and automates the rule writing process by providing ready-made explanations for annotations, discovering useful contextual clues, allowing a developer to write simpler regular expressions, and replacing the primarily manual rule refinement process with that of selecting from a ranked list of concrete refinements.

[1]Available for download at http://alphaworks.ibm.com/tech/systemt

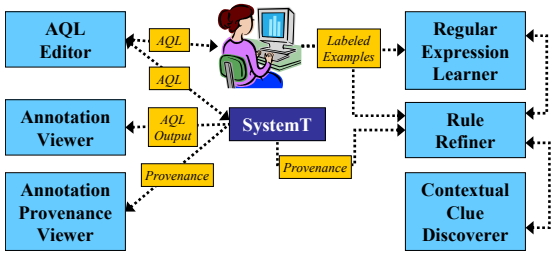

Figure 2: The architecture of SystemT IDE.

3. DEMONSTRATION OVERVIEW

Demonstration Scenario. We assume an email semantic search application, and use a sample of the Enron dataset as underlying collection of emails. In this context, we wish to specify semantic queries such as "`Tom phone`" in order to find Tom's actual phone number, as opposed to emails containing the words `Tom` and `phone`. A crucial step in enabling such queries is the ability to extract from emails entities of type Person and Phone and the relationship between them.

We start our demonstration with a naive set of rules for identifying PersonPhone relationships, as an example of a first version of annotator that a rule developer would devise during initial iterations of developing the annotator. Figure 3-A includes a screenshot of the *AQL Editor* showing a subset of this annotator. We use this naive annotator to illustrate the main constructs of SystemT's AQL rule language, as well as basic functionalities of the *AQL Editor* such as syntax highlighting and hyperlink navigation.

Next, we execute the annotator on our collection of Enron emails, and examine its output in the *Annotation Viewer* (Figure 3-B) We show that the naive annotator suffers from both low precision and low recall: it generates incorrect annotations (e.g., `Morgan Stanley, fax: 205-4493`), while missing correct ones (e.g., as in `Call Emma, x33650`).

The rest of our demonstration focuses on several features of the SystemT IDE that reduce the manual effort involved in all stages of the development by (semi-)automating some of the labor-intensive tasks. We now describe how each of these features facilitates the refinement of the naive PersonPhone annotator in one iteration of the development process.

Annotation Provenance Viewer. For each annotation in a document, SystemT IDE displays a visual diagram illustrating its *provenance* [1]: *how* it has been generated by the annotator when applied to that document. Figure 3-C shows the provenance of `Morgan Stanley, fax: 205-4493` which enables the developer to understand that it is the rule FullName, which uses a dictionary of complete names to identify candidate persons, that is responsible for generating the incorrect annotation. Therefore, to eliminate the incorrect annotation, the developer may remove the entry `Morgan Stanley` from this dictionary. The method used in generating provenance diagrams is based on techniques from the field of *data provenance* and is explained in [7].

Contextual Clues Discoverer. This component of SystemT IDE clusters the context surrounding annotations in order to detect frequently occurring patterns, and presents them in a meaningful way. For instance, Figure 3-D shows the result of clustering the region of text between incorrect

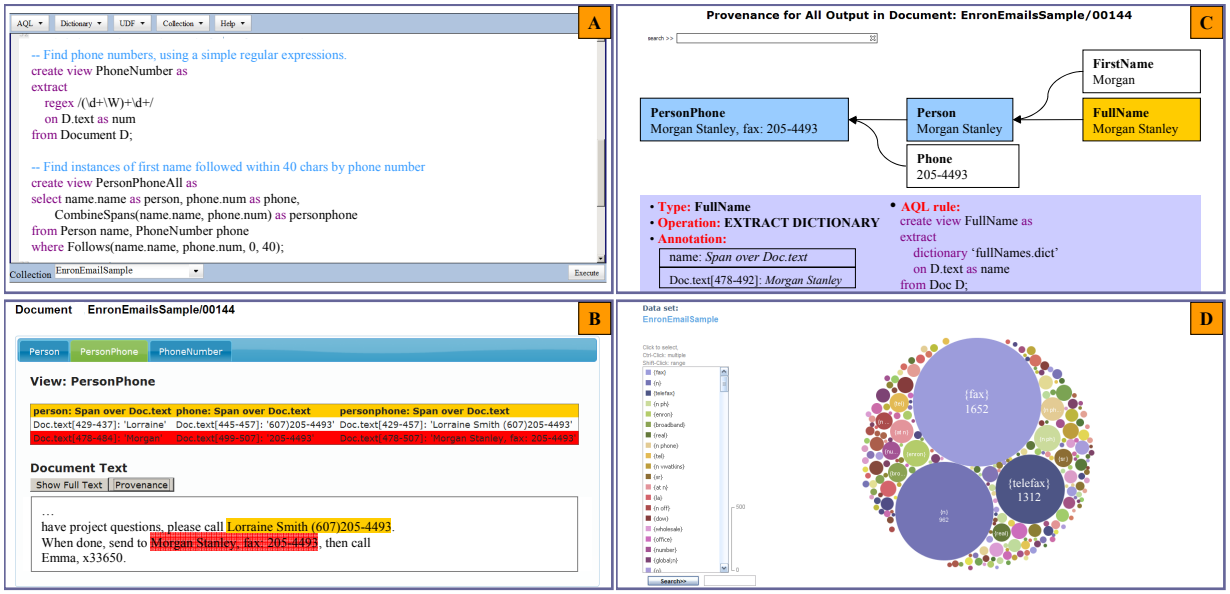

Figure 3: (A) AQL Editor; (B-C) Annotation and Provenance Viewers; (D) Contextual Clue Discoverer.

PersonPhone pairs, which reveals frequent occurrences (note the large bubbles) of clues such as `fax` and `telefax`. This enables the developer to improve the precision of the annotator by adding a rule that filters out PersonPhone pairs if the Phone component is preceded by a fax number clue. Furthermore, the system may discover that `call at` frequently occurs with correct PersonPhone annotations, and at the developer's request, it can explore other contexts in which the same clue occurs. This reveals that `call at` also co-occurs with extension numbers that are not currently captured by the Phone annotator, and accordingly, the developer can add the appropriate rule(s) to improve recall.

Regular Expression Learner. We demonstrate the ability of SystemT IDE to learn a complex regular expression, starting from a much simpler regular expression R_0 that captures with high recall but low precision the target entity, and a set of positive and negative matches for R_0. For example, this feature allows the developer to start by specifying a simple regular expression such as $(\backslash d+\backslash W)+\backslash d+$ (blocks of digits separated by a non-word character) that identifies correct phone numbers, but also incorrect ones such as social security numbers, and IP addresses. The system will then automatically generate a refined (more complex) regular expression that captures most of the positive examples given as input, and as few negative examples as possible. That is, the resulting regular expression has much higher precision with a similar recall. The hill climbing algorithm behind the Learner is described in [6].

AQL Rule Refiner. Finally, we demonstrate SystemT IDE's ability to automatically suggest concrete refinements at the level of an entire annotator. For example, given a set of positive and negative examples in the output of the PersonPhone annotator, the system suggests multiple refinements that improve the precision of the annotator, such as: $R1$) Filter PersonPhone if region between the Person and Phone pair contains another person's name; $R2$) Filter Phone if the immediate left context contains tokens such as `fax` or `f#:`; $R3$) Remove `Morgan Stanley` from the dictionary of full names. The refinements are ranked according to how much each improves the precision with the same recall. The developer may choose one or more refinements to be automatically applied, or even enhance a refinement before applying it, based on domain knowledge, e.g., enrich the filtering dictionary automatically suggested in refinement $R2$ with additional entries such as `efax`. The methods underlying the Refiner are detailed in [7]. While the Provenance Viewer, the Contextual Clues Discoverer, and the Regex Learner are individual components useful by themselves, several of their internal techniques are integrated in the Rule Refiner in order to determine annotator components responsible for a large number of mistakes, and generate specific types of refinements.

4. REFERENCES

[1] J. Cheney, L. Chiticariu, and W. Tan. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[2] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, and S. Vaithyanathan. SystemT: An Algebraic Approach to Declarative Information Extraction. In *ACL*, 2010.

[3] L. Chiticariu, R. Krishnamurthy, Y. Li, F. Reiss, and S. Vaithyanathan. Domain Adaptation of Rule-based Annotators for Named-Entity Recognition Tasks. In *EMNLP*, 2010.

[4] L. Chiticariu, Y. Li, S. Raghavan, and F. R. Reiss. Enterprise Information Extraction: Recent Developments and Open Challenges. In *SIGMOD (Tutorial)*, 2010.

[5] R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu. SystemT: a System for Declarative Information Extraction. *SIGMOD Record*, 37(4):7–13, 2008.

[6] Y. Li, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. V. Jagadish. Regular expression learning for information extraction. In *EMNLP*, 2008.

[7] B. Liu, L. Chiticariu, V. Chu, H. V. Jagadish, and F. Reiss. Automatic Rule Refinement for Information Extraction. *PVLDB*, 3(1):588–597, 2010.

[8] F. Reiss, S. Raghavan, R. Krishnamurthy, H. Zhu, and S. Vaithyanathan. An Algebraic Approach to Rule-Based Information Extraction. In *ICDE*, 2008.