I Can Do Text Analytics! Designing Development Tools for Novice Developers

Huahai Yang* **Daina Pupons-Wickham**** **Laura Chiticariu***
Yunyao Li* **Benjamin Nguyen**** **Arnaldo Carreno-fuentes***
**IBM Research - Almaden* ***IBM Software - Silicon Valley*
San Jose, CA, USA
{hyang,daina,chiti,yunyaoli,nguyenb,acarren}@us.ibm.com

ABSTRACT
Text analytics, an increasingly important application domain, is hampered by the high barrier to entry due to the many conceptual difficulties novice developers encounter. This work addresses the problem by developing a tool to guide novice developers to adopt the best practices employed by expert developers in text analytics and to quickly harness the full power of the underlying system. Taking a user centered task analytical approach, the tool development went through multiple design iterations and evaluation cycles. In the latest evaluation, we found that our tool enables novice developers to develop high quality extractors on par with the state of art within a few hours and with minimal training. Finally, we discuss our experience and lessons learned in the context of designing user interfaces to reduce the barriers to entry into complex domains of expertise.

Author Keywords
Novice developer; text analytics; best practices; information extraction; extraction plan; workflow guide.

ACM Classification Keywords
H.5.2. Information Interfaces and Presentation: User Interfaces - interaction styles, user-centered design

General Terms
Human Factors; Design; Measurement; Languages.

INTRODUCTION
Text analytics (also known as information extraction) refers to techniques that distill structured information from unstructured or semi-structured text. For example, one may want to use text analytics to extract the revenue numbers from a large number of companies' annual reports; to identify people mentioned in the transcripts of recorded conversations; or to infer consumers' purchase intention from their Twitter messages. Text analytics is crucial for maximizing the value of information embedded within all sources of text, such as emails, web pages, news reports, clinical records, social media and many others. Not surprisingly, text analytics has emerged as a critical building block for a wide range of applications, including brand management, competitive intelligence, customer relationship management, regulatory compliance, fraud detection, and life science, among many others [8, 10, 31].

However, developing high-quality text analytics programs, also known as *annotators* or *extractors*, is a very complex task. Currently, developing extractors to solve real world problems requires extensive knowledge and experience in technical areas such as machine learning and rule systems. It has been said that machine learning based text analytic solutions "come packaged in a Ph.D." [44]. Speeding up the extractor development life cycle and reducing its barrier to entry becomes a critical requirement to fulfill the increasing demands for text analytics [3].

Not surprisingly, there have been considerable efforts in improving the usability of text analytics systems. Lightweight and interactive text analytics development environments such as the Data Wrangler [11, 20] and MashMaker [9] are easy to use and allow even non-technical users to do certain types of analytics. However, they offer only a limited range of capabilities and do not allow developers to build the sophisticated extractors required by real world applications. While tools designed to reduce the manual effort involved in text analytics development [1, 25, 26, 27, 42] are effective for expert developers, using such tools for text analytics remains challenging for novices: unlike typical programming tasks, text analytics requires a data-driven approach that is unfamiliar to most programmers. In our experience, we have found that novice developers, when facing a text analytics task, simply do not know how to approach the problem.

Our contributions are summarized as follows. (1) We have distilled the methodology and best practices followed by trained linguists and expert text analytics developers via interviews. (2) We have designed and implemented WizIE[1],a novel guidance tool that significantly lowers the barriers to entry into text analytics. At a high-level, WizIE comprises of two major components: the *Workflow Guide* and the *Extraction Plan*. The Workflow Guide takes the user through the end-to-end development process based on the methodology and best practices of trained linguists and expert developers. Whereas the *Extraction Plan* maintains a model or specification for the extraction task: the user builds this plan over time by combining text snippets and clues. Each element of

CHI 2013, April 27–May 2, 2013, Paris, France.

[1]WizIE is available in two IBM enterprise software products: IBM InfoSphere BigInsights [17] and IBM InfoSphere Streams [18].

the plan is associated with partially generated code to extract those pieces of text. Our approach bridges the gap between a full-fledged development environment and lightweight authoring tools. It helps novice users learn the proper way of approaching text analytics problems and how to harness the full power of the underlying text analytics system. (3) WizIE has been developed using a user centered design approach. The development process went through a series of design iterations and user studies. In our latest user evaluation, we found that our tool has enabled novice developers to write high quality extractors on par with state of the art within a few hours after a minimum amount of training.

RELATED WORK

Text Analysis Development Environment

Many text analysis software suites, such as GATE [5], IBM Content Analytics Studio [16], SAP BusinessObjects ThingFinder Workbench [41], and VisualText [43], include a development environment. These development environments usually provide comprehensive support for writing text analytics programs, such as an editor and facilities for executing and testing an extraction program, and some of them may include tools designed to reduce the manual effort involved [1, 25, 26, 27, 30, 42]. While these tools are effective in assisting expert developers, novice users (including experienced software developers in other domains) find that developing text analytics remains a challenging task that is suitable "for geeks" only [4]. Our first user study attests to such difficulties.

Recently, some lightweight and interactive text analytics development environments such as the Data Wrangler [11, 20] and MashMaker [9] appear to support casual users to perform some text analytics tasks. However, as far as we are aware, the limited capabilities included in these lightweight solutions have yet to permit sophisticated extractors to be built as demanded by real-world business applications. In contrast, our work attempts to lower the barriers for novice developers to harness the full powerful of a complete text analytics solution.

Tools to Lower Barriers to Programming

Although we found limited existing research on lowering the barriers to entry for developing text analytics programs, we have drawn inspiration from the long and venerable line of research on facilitating novices to learn general purpose programming [21, 32, 38]. One approach transforms text-based programming tasks into visual and direct manipulation tasks to leverage people's well tuned cognitive capabilities of dealing with the physical world [2, 14]. Many of these systems model themselves on physical metaphors that the target audiences can easily relate to [19, 35] or build on declarative constructs that non-programmers are already familiar with [34]. We borrow similar strategies in designing our tools. However, as we shall describe in detail, developing text analytics programs has its unique challenges distinct from general purpose programming.

The field of end user programming investigates means to support programming that is "primarily for personal, rather public use" [22]. Some text analytics development work could fall into such categories, while most would not. Nevertheless, the historical focus of the field on novices provides useful ideas related to the current research. Programming by demonstration [6], extracting and re-purposing existing content [45], or recording user actions with simplified languages [24] are some of the existing strategies. In these systems, while novices can quickly create some interesting programs, more complex tasks remain difficult to implement.

Our work bears some resemblance to research on lowering the barrier to entry to machine learning [36, 37]. Similar to Gestalt [36], our work provides explicit support for both code and data, enables easy transition between implementation and analysis, and guides users through the implementation pipeline. However, unlike Gestalt, which focuses on one specific machine learning task, WizIE is a generic tool designed for supporting arbitrary text analytics tasks. As a result, WizIE needs to overcome additional challenges such as how to guide novice users to turn a high-level business requirement into concrete text analytics tasks and how to provide generic task-independent structured guidance.

Learning Support

Due to the distinctive nature of text analytics tasks, our work pays great attention to teaching the proper way of approaching text analytics problems. Learning science, particularly, Bruner's theory of scaffolding [46] provides a theoretical foundation for designing the learning support feature of WizIE. In a sense, our tool is an instance of software-realized scaffolding [12, 39], where the design goal of the user interface is to give novice developers sufficient support to promote learning the best practice of text analytics. Concretely, two types of learning support were explicitly designed as major components of WizIE. The *Workflow Guide* was designed to teach the steps necessary for successful development of extraction programs. Users learn the conceptual and procedural knowledge of text analysis by performing activities prescribed by the guide. Another important component is the *Extraction Plan*, a specification tool that allows users to form a high level conceptualization of the problem and its solution within the concrete context of source document collections. The Extraction Plan also ties closely to the underlying programming language, generating code templates to serve as the means for self-disclosure [7], where novice users can gradually learn the full power of the underlying language, while initially staying in the comfort zone of a familiar graphical and direct manipulation environment. While a model and specification driven approach has its place in professional software engineering [13] where it is used by *expert* developers, the use of such an approach to help *novice* developers to bridge the gap between high level requirement and concrete implementation is a new direction.

DESIGN PROCESS

This work was initiated as part of the tooling effort for SystemT [3], a state-of-the-art text analytics platform built around the text analytics programming language AQL (Annotation Query Language) and commercially available [17, 18]. While our tool has been implemented for AQL, our design ideas can be applied to any text analytics rule language.

In this section, we first briefly discuss AQL at a level of detail necessary to understand the ideas in this paper, followed by our design process.

Background

AQL provides many constructs for text analytics, including primitive extraction operators for finding parts of speech, matches of regular expressions and dictionaries, as well as higher-level set operators such as sequence, union, filter and consolidate, to name a few. AQL programmers build text analytics programs by composing these operators together into sets of *rules*, or *statements*. As an example, consider the task of extracting names of people from text. We will call each name a *person name* for lack of a better term. A text analytics program for this task would make use of primitive operators to first identify *basic features* of person names, including matches for dictionaries of honorifics (e.g. "`Dr`", "`Mr`") and common first names and last names, or matches for regular expressions that identify syntactic features (e.g. capitalized words or initials). Next, the *sequence* operator can be used to build *candidate* person names from basic features. In our example, a possible rule is to identify a sequence of a first name immediately followed by a last name, while another possible rule is to identify a sequence of a honorific followed by two capitalized words. Candidate person names generated via such rules must then be merged into a single set using the *union* operator, and further *filtered and consolidated* to remove invalid candidates and resolve overlap among the remaining candidates. An example filter rule would be to subtract from the set of all candidate persons, those that overlap with another type of entity (e.g., in the text "`Bill and Melinda Gates Foundation`", `Bill` and `Melinda Gates` are not generally considered person names, since they are part of an organization name). An example consolidate rule would be to remove candidates completely contained within other candidates (e.g., in the text "`Mr. Bill Gates`", the two individual sequence rules mentioned above would identify two candidates `Bill Gates` and `Mr. Bill Gates`; then the consolidate rule would remove the former, which is redundant.)

Original Design and User Study

From the start, tooling has been recognized as a strategic priority for SystemT. In addition to standard development tools such as an editor for AQL and facilities to execute and analyze the output of AQL programs, a considerable investment has been made to develop automated or semi-automated tools to assist developers. For example, three separate approaches were designed and implemented just to support the creation of regular expressions: a regular expression builder in which the developer composes a regular expression by manipulating various GUI controls representing basic elements of regular expressions; a regular expression generator, where developers specify examples of text, and a regular expression covering the examples is automatically generated; and the AQL editor, where users can edit regular expressions directly. Each of these approaches was designed in close collaboration with existing expert AQL developers.

However, the first formal usability study with *novice* text analytics developers revealed many problems. In that study, all five participants (four male and one female) were professional software developers with 3+ years of experience working with the Eclipse development environment, upon which all the tools were developed. Two participants had some limited experience with regular expressions through creating code to extract information from text files.

In the study, we asked participants to create extractors to identify telephone numbers within email text. Participants were provided a corpus of email files on which to run the extraction tasks, and asked to utilize the three aforementioned regular expression tools. A Getting Started guide and online help were available for use during the session.

Three of the five participants were able to complete the task within the two hour time limit. The two participants who failed to complete the tasks used the Getting Started guide for over an hour and were still not successful. The overall impression of the ease of use of the product was not very positive: only two of the participants rated the product's ease of use as above neutral (on a 5 point scale).

This first usability test showed that the key usability issues in the tool were due to high level conceptual difficulties, rather than an issue of individual user interface component design. Interviews with the participants indicated that they did not understand how to approach the problem, and that they had a poor grasp of the work flow. For instance, when creating a project for the first time and facing an empty project workspace, they were unsure of the next steps. Participants also had issues uncovering the prerequisite steps for configuring project properties and running AQL programs.

Design for Novice Users

Since the syntactic and semantic aspects of the AQL language were well covered in the Getting Started guide and online help, we realized the existence of a large gap between the AQL experts and novices in terms of the "big picture" of text analytics. Since we did not want the novice users to overcome the gap "the hard way" (i.e. via trial and error), or subject them to an expensive extended training regime, a plan was put in motion to revise the user interface so that it helps novice users get up to speed as quickly as possible.

Elicit Knowledge from Expert Users

The first step we took was trying to understand what expert users know about the big picture of text analytics. We had meetings with two expert AQL developers once a week for two hours over a period of two months. In the meetings, we probed their thought processes with sample extraction tasks, and encouraged them to codify their implicit knowledge with a set of "best practices". The outcome of this collective reflection process is a methodology for extractor development.

Methodology for extractor development

Figure 1 illustrates the general methodology for the development of high-quality, high performance extractors used by text analytics experts, consisting of the following four phases.

- **Extraction Specification**: In this initial phase, extraction tasks are defined. This critical step attempts to match the abstract business requirements with the concrete reality of

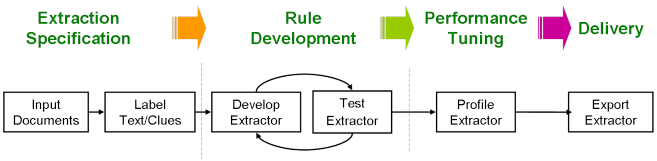


Figure 1. Methodology for Extractor Development

the data, a process often involving manual examination of the source text documents and/or look up of domain knowledge.

- **Code Development**: Once extraction tasks are well defined, code is written for each individual task. Writing code is almost always an iterative process, multiple iterations are required to produce high-quality extractors even for simple tasks. Tests are performed between iterations, until the quality of the extractor reaches the desired level.

- **Performance Tuning**: After the developer is satisfied with the quality of the extractors, the code is profiled to identify any performance (mainly running speed) bottlenecks and further fine-tuned to ensure high performance.

- **Delivery**: Finally, the code is packaged to be easily embedded in various applications.

In this study, we were concerned with the first two steps.

Design Features to Guide Novice Users

In order to define the Extraction Specification, developers must first examine sample source text to determine (1) what information should be extracted, and (2) which pieces of text in the source file (also known as clues, or features) could be leveraged in order to extract the desired information. Only after the developer is armed with such information, can she start writing the extractor. Once a first version of the extractor is ready, the developer tests it on the sample source text to assess its quality. If the quality is insufficient, the developer must identify additional clues, refine existing ones, and repeat the process until the desired quality is achieved. Unfortunately, this process is often labeled as an "art" by expert developers. In particular, such a data centric, test driven development approach is largely unknown to many programmers proficient in object oriented design methodology. In our initial user study, two of the participants faced a seemingly insurmountable difficulty precisely due to this paradigm shift.

This type of high level conceptual difficulty seems to go beyond the six barriers identified by Ko [23] for novice programmers, but still falls within Norman's *Gulf of Execution* [33], where user's intention does not match the available actions provided by the system. For example, in the original design, the focal point of the user interface is the AQL editor. However, the objects that AQL statements operate on, the source text documents, are hidden in the background. Novices would not realize the need to examine the source text to look for clues. To overcome such a Gulf, our first design decision was to bring the source text documents to the forefront and the center of the user interface. The goal is to make the work objects explicitly visible to the users. To make the effect of code changes immediately visible, the source documents display also serves as a place to show the results of extraction, where extracted text snippets are highlighted.

In the remainder of this section, we describe the major features introduced in the first iteration of our design of the tool, and the user studies conducted with it. The user studies prompted a second design iteration, where some of the initial features were enhanced to arrive at the current design. In the next section, we shall describe the major features in more detail in the context of the current design.

To make the source documents the center of operation, users must first select a sample collection of text documents immediately after initiating a project. Such ordering constraints naturally lead us to a wizard-like user interface design that includes all the steps in the expert's methodology, a *Workflow Guide*. From the perspective of learning science, the Workflow Guide serves the purpose of supporting a scaffolding process, where users learn to perform a complex task through clearly presented and incrementally built-up activities. Unlike a wizard, the tool was not designed as a sequence of modal dialogs that force users to step through, but as a guide that suggests the proper steps to the user. We designed the guide as an ordered set of collapsible panels, where each step of the methodology takes up a panel. When a step is not in focus, it can be folded up. All the steps are optional, except for the mandatory first step of defining the source text document collection, which avoids the "blank screen" problem observed in the first user study. The entire guide can be folded up if not needed, once the user becomes an expert developer. From the perspective of scaffolding, the guide can be said to fade away as a learner becomes a self-sufficient user.

The Workflow Guide suggests using the mouse to mark, within the source document display, portions of the text that can be used as clues. Users then assign a label to each marked clue. The labels are collected in a panel to the right of the document display. Because clues representing low level concepts can be combined to form larger units of extraction representing higher level concepts, we use a tree structure to organize all the marked clues. For example, house number might be a pattern of text that starts with a number, street name a pattern of text that ends with "`St.`", "`Ave.`" and so on; the combination of them can form an extraction unit for street address, which combined with state, zip code and country could form a full address. These clues form a natural hierarchy and therefore, are displayed as such.

Essentially, through direct manipulation, novice users are instructed by the user interface to form a conceptual level plan of extraction. We therefore call such a tree of labels an *Extraction Plan*, a model or specification for the extraction task. In addition to the bottom-up approach of labeling the clues then combine them, users can also start with a high level concept and break it down into lower level concepts and finally map to the text snippets as clues in the documents. In addition, users can move the branches of the tree around to reorganize the extraction plan. All these operations are designed to follow the principle of direct manipulation [15].

For the Extraction Plan to be more useful, so that users are motivated to build one, we added capability to automatically generate AQL code templates for each clue in the extraction plan, based on the type of clue. The generated code has the added benefit of self-disclosing the underlying language for novices to learn [7]. The code generating dialogs also make available other automation tools such as regular expression builders. Therefore, the Extraction Plan effectively becomes a hub that links user interface components together.

In summary, in our attempt to help novice users learn to write extractors following a similar process as the experts, we have developed two novel user interface components, a Workflow Guide that suggests the best practices and an Extraction Plan that induce users to approach extraction problems in the proper way. The Extraction Plan resembles a model driven development tool, where a novice user builds a high level conceptual model via direct manipulation, then uses the model to drive code generation. To the best of our knowledge, using such a model-driven development approach for novices is a new way of lowering the barriers to programming.

Formative Evaluations of the Design

Working with two novice users of AQL, we used low fidelity prototypes on paper and on white board to evolve our design. The finalized prototype was then implemented. Once the implementation was completed, we conducted two evaluation studies: (1) the short term usage of the system in a lab setting, and (2) a real world long term usage.

Lab Comparative Study

Setting. This study compared extractor development performance with or without our new design, and was conducted during a 2-day training session of AQL. On Day 1, 14 novice developers of text analytics were given a thorough lecture on the topic, shown code of example extractors, and given exercises to develop extractors using the original user interface. Towards the end of Day 1, participants were asked to solve an extraction problem: developing an extractor to identify mentions of a company's revenue grouped by divisions in the company's official annual report. On Day 2, the new user interface (with Workflow Guide and Extraction Plan) was introduced to the same 14 participants, and its features demonstrated and explained with examples. Participants completed the same exercise as on Day 1, but using the new UI. Clearly, such a study desigin could not avoid a potential learning effect. However, as the purpose of the study was formative rather than summative, we considered such a limitation an acceptable compromise since the cost of running a well controlled experiment would be too high at this early stage.

Results. On Day 1, no participant was able to complete the exercise after 90 minutes. In the after-lab survey, one participant wrote *"I don't think I would be able to recreate the example on my own from scratch"*; another participant suggested that *"I'd like the [extractor] editor to be more visual"*. On Day 2, all participants were able to complete the exercise under 90 minutes. In fact, two participants created extractors with accuracy and coverage of over 90%. Overall, the participants were much more confident about creating extractors. One participant wrote *"My first impression [of the User interface] is very good"*. On the other hand, another participant asserted that *"The nature of the task is still difficult"*.

For the question "How easy was it to build extractors", the median rating was improved from 4 to 3 from Day 1 to Day 2, with 1 being "Very easy" and 7 being "Very difficult". Similarly, the median ratings of easiness of "setting up the project", "running the project" and "viewing and navigating results" were all improved from 4 to 2 (meaning "fairly easy"). However, the median usefulness ratings of each components of the interface was all 4, between 'useful' and 'not useful'. As one participant wrote, participants "need more exposure and experience" to better judge the usefulness of the interface components.

Real-world Use Study

In this study, participants used the interface over the course of several months to complete a real project. At the end of the project, participants were interviewed about their experience.

Setting. Ten participants were tasked with building an end-to-end application for the pharmaceutical industry based on information available from public data (e.g. SEC filings). The project spanned over a period of 12 weeks, with each participant contributing 4 to 5 hours/week to the project. Four of the participants were responsible for developing the text analytics component using the tool to extract key information related to the pharmaceutical industry (e.g. drug names, clinic trials) from financial reports. None of the participants had any text analytics experience prior to the project. In addition to the recording of the training given to the participants in our earlier lab comparative study, the participants had access to additional training materials and online references, as well as assistance from a text analytics expert.

Interviews. We interviewed the 4 participants (3 male, 1 female) responsible for the text analytics development, They identified themselves as: developer (2), infrastructure and solution architect (1), and application and integration software IT specialist (1). Participants used the tooling for an average of 39 hours.

All interviewees felt the tooling was easy to use, but none felt that they could have followed the flow without the tutorial. Three especially liked the feature to mark snippets and clues. Unfortunately, only one participant used the Extraction Plan. According to the participants, the low utilization of the Extraction Plan was due to two reasons. One was that the navigation from the Extraction Plan to the editor was not always clear. The other was that some actions such as creating a dictionary were cumbersome in the user interface, and as a result the interviewees started to do things manually. It was clear that a redesign was needed in order to meet our goal of providing a bridge for novice users. As a result, we revisited our overall design of the Extraction Plan as well as lower-level interactions within the UI. The updated design is described in the following section.

CURRENT DESIGN

We now describe our current design with a focus on the first two phases of extractor development: extraction specification

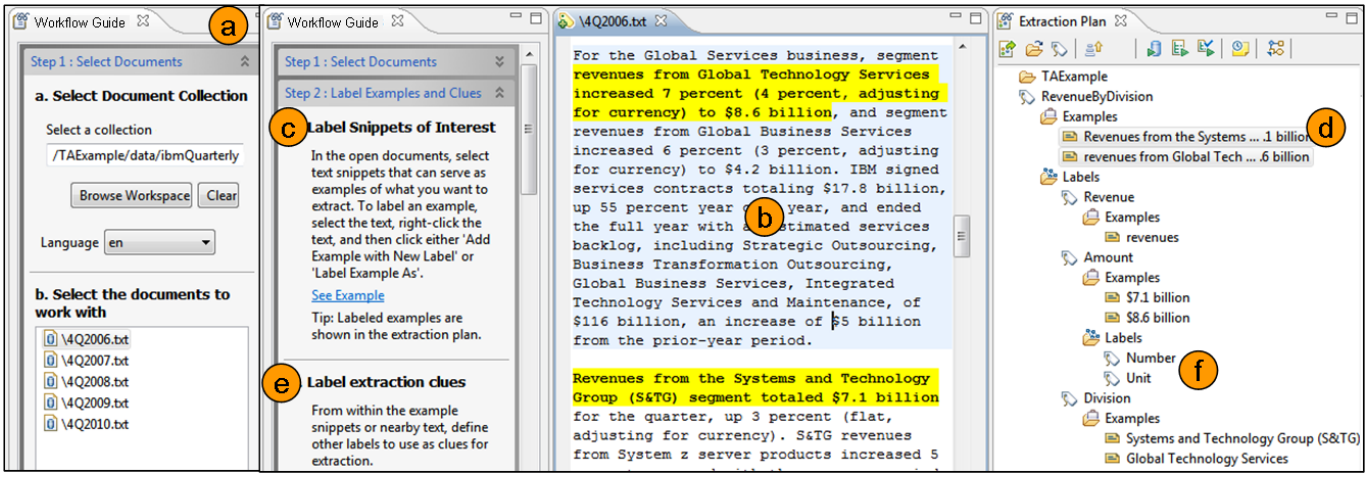


Figure 2. Extractor Development with WizIE: Extraction specification.

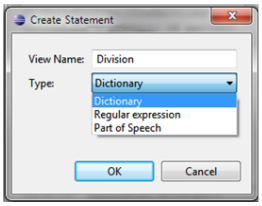
and rule development. For illustration purposes, we assume a high-level business requirement of identifying trends in revenue generated by each business division of a company based on its press release. In the rest of the section, we use (a) – (m) with reference to Figures 2 and 3.

Document selection. Step 1 of the Workflow Guide (a) instructs a user to select a set of documents and specify the language of those documents. It lists the individual documents within the chosen set. The user can then select any individual document within the set and open it in the main editor area within Eclipse (b) to begin working with it. The Extraction Plan is empty as the user has not yet begun to label any snippet of text or write any code.

Identifying snippets of interesting text. The first part of Step 2 of the Workflow Guide (c) instructs the user to identify snippets of interesting text. In our case, the user would start by labeling example snippets of revenue by division by selecting the relevant portion of text (e.g., "`revenues from Global Technology Services increased [] to $8.6 billion`", and choosing Add Example with New Label from the context menu. The user then fills out a simple dialog with the label name and parent label if any. The extraction plan is updated with the new label and the example (d). Additionally, the embedded context menu is updated to include the new label, to facilitate future labeling.

Once the user has a good set of examples, she can move onto the next step, labeling extraction clues, as indicated by the second part of Step 2 of the Workflow Guide (e). The user would break down the examples into meaningful chunks or clues. The process of recording clues is identical to that of recording full examples, except that the user can now choose a parent label to build the hierarchy. In our example, meaningful clues would include: the word *revenue*, the division name, and the amount, all recorded in the Extraction Plan (f). Notice how the Extraction Plan effectively guides the developer in writing actual rules: by analyzing the text of interest and chunking it into meaningful clues, the code structure has begun to be defined.

Developing the extractor. Step 3 of the Workflow Guide (g) informs the user how to proceed to start writing code. At this point, the Extraction Plan moves from being a way to organize text into a launch point for generating AQL statements to extract the text. Because AQL is a powerful language with many constructs, one of our goals was to avoid overwhelming the user by showing the (long) list of all possible constructs. Therefore, we have designed the New AQL Statement menu to first expand into three entries: Basic Features, Candidate Generation and Filter & Consolidate, corresponding to the three main parts of an extractor, as explained earlier in the context of the Person extractor. Each category unfolds into a small list of AQL constructs commonly used in that category. For example, the Basic Feature menu leads to a list (shown below) consisting of primitives: dictionary, regular expression, part of speech, whereas the Candidate Generation category consists of constructs such as sequence and union.




```
create dictionary DivisionDict
from file '<path to your dictionary here>'
with language as 'en';

create view Division as
extract dictionary 'DivisionDict'
    on R.<input column> as match
from <input view> R;
output view Division;
```

Each category also corresponds to an individual AQL file automatically created to store the code for that category. For example, if the user selects "Dictionary" as the type of Basic Feature AQL statement to implement the label Division, then the template for creating a dictionary (shown above) is automatically added to the code in the editor, in the corresponding file *basics.ql*. The user can then edit the template to point to a dictionary of interest (h). Alternatively, if the user had a list of division names identified within the Extraction Plan, they could simply multiple select those items and opt to have a dictionary created for them automatically. A similar facility allows the user to load multiple examples into the regular expression generator tool, if the examples are best captured with a regular expression rather than a dictionary. The Ex-

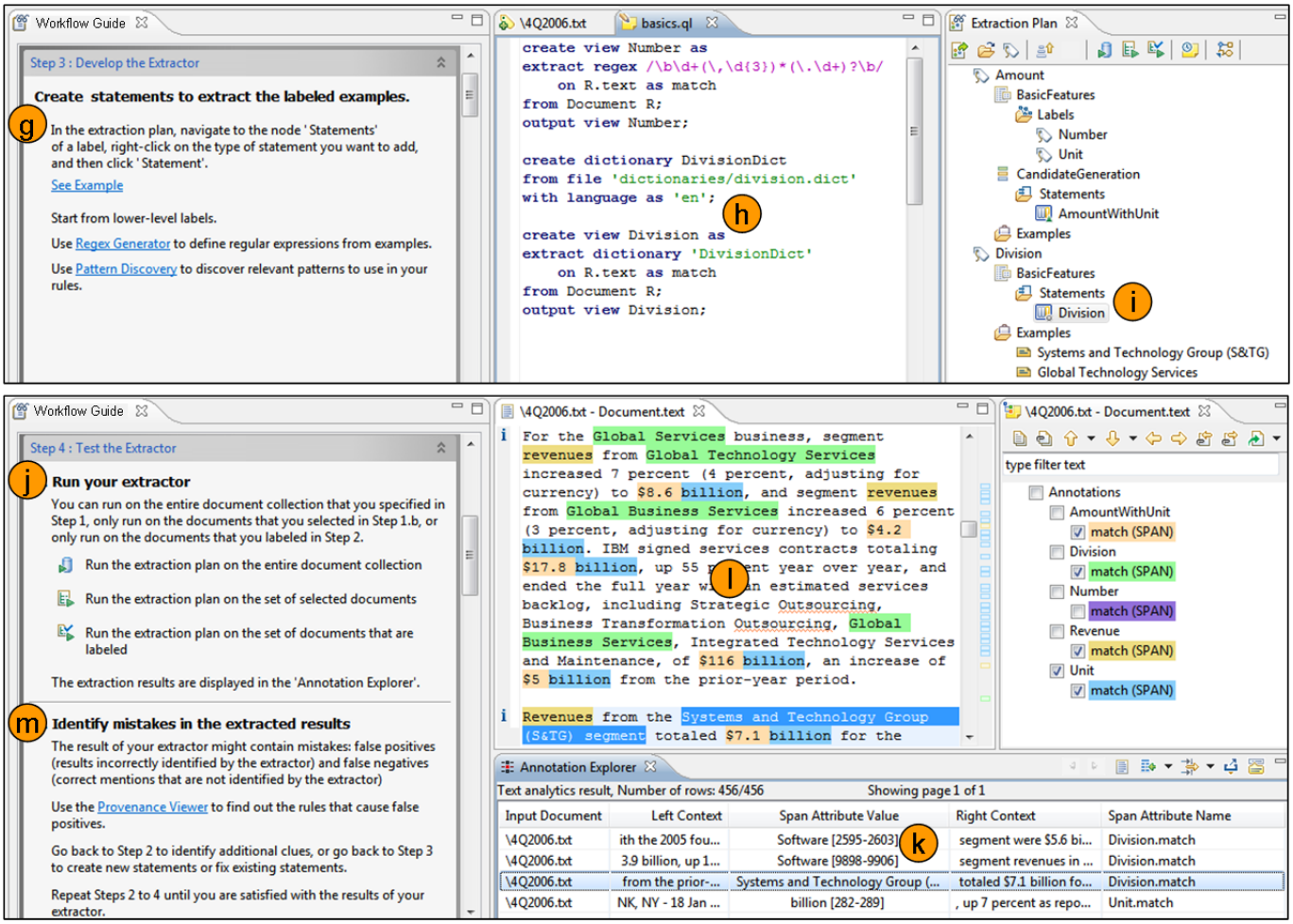


Figure 3. Extractor Development with WizIE: Rule development.

traction Plan also updates to show the Division basic feature AQL statement under the Division label (i).

Note that the New AQL Statement menus do not capture all possible AQL constructs. As the user gains more experience, she can edit the code directly and associate resulting statements with the Extraction Plan via drag-and-drop from the AQL editor. Furthermore, in addition to the Extraction Plan's connection with the existing tools such as the AQL and dictionary editors, the Workflow Guide panel (g) also has links to existing tools (e.g., Pattern Discovery, Regex Generator) that are helpful at this stage.

Testing the extractor. Once the user has completed one iteration of development, the first part of Step 4 of the Workflow Guide (j) allows her to execute the current extractor on the collection of source documents and to analyze the extraction results. The results are shown in a tabular form in the Annotation Explorer panel (k), as well as highlighted within each source document (l) for maximum context. The second part of Step 4 (m) informs the user of the types of incorrect results she might encounter (false positives and negatives), and the process to follow to fix them: understand false positives using an existing Provenance tool, and iterate through Steps 2 and 3 in order to add and/or refine the examples for clues and the rules.

Observations. While the first iteration of our design was close to the second iteration in terms of the general purpose and functionality of the Workflow Guide and Extraction Plan, the second iteration added a number of enhancements to the Extraction Plan to address concerns expressed by users in our former lab setting and real use studies. Specifically: (1) the ability to create regular expressions and dictionaries by selecting examples from the Extraction Plan, to address a previous shortcoming that examples could not be reused and had to be recreated in the dictionary editor or the Regex Generator from scratch; (2) enhancement to the Extraction Plan to not show nodes that do not have any descendants, as well as an alternative view for the plan that shows only AQL statements, without the examples, to address the concern that the Extraction Plan can become too large and difficult to scroll; and (3) enhanced flexibility to add labels inside a category of statement to address a concern that the hierarchy within a label may become disconnected from the implementation when the hierarchy is more than two levels deep.

EVALUATION

Table 2. Top 10 Songs titles from billboard.com, week of May 5, 2012

Somebody That I Used To Know	We Are Young
Payphone	Boyfriend
Glad You Came	Wild Ones
Call Me Maybe	What Makes You Beautiful
Starships	Stronger (What Doesn't Kill You)

For the current iteration of evaluation, we wanted to test the upper limit of the novice users' performance using our user interface design, i.e. given a non-trivial extraction task, how good can extractors developed by novices be? To do so, we used a competition format to motivate the participants. The winners of the competition were given monetary awards based on how well their extractors performed.

Procedure

The study consisted of the following components: (1) a 2 hour training session in which the participants learned about basic concepts of text analytics, the AQL language, the WizIE tooling, and the high-level task required for the contest; (2) a 10 day period in which the participants built extractors independently to accomplish the task, with access to the recording of the training session, and online reference materials about AQL; (3) a 1 day period in which the participants submitted their extractors along with a completed questionnaire; (4) an award session, during which each participant gave a brief presentation detailing their experience with the task, followed by the announcement of the contest winners; and (5) a hour long post-study interview.

Participants. The participants were graduate students doing summer internships in IBM. Six interns registered for the contest and filled out the questionnaire. All participants were computer science majors with various levels of Eclipse experience. None of them were interns for this research project, nor had any prior experience in text analytics. Four participants submitted an entry and agreed to be interviewed.

Tasks. We asked participants to identify mentions of the Top 10 Billboard songs in the week of May 5, 2012 from Twitter (see Table 2). We selected the task based on two criteria: the difficulty and the relevancy to the participants. The task needed be difficult enough to reflect the real-world complexity of text analytics. As illustrated by Table 1, identifying whether a match of the name of a song is a true mention of the song is a non-trivial task even for a human being. At the same time, we wanted a task that the participants would find relevant and fun, since we realized that participants had to squeeze time out of their own busy work schedule to make the time commitment to our competition [2].

We further restricted the scope of the task by limiting the identification of song names to a small set of tweets. Specifically, we first randomly collected 10,000 tweets from the week of May 5, 2012, each containing at least one of the names listed in Table 2. Among these, we selected a set of 259 distinct tweets (i.e., no re-tweets) such that each song name was mentioned in at least 10% of the tweets in the set (a tweet may contain multiple song names). We manually labeled the correct song mentions in the 259 tweets and split the set into a training set consisting of 159 tweets randomly selected from the set, and the test set, consisting of the remaining 100 tweets. The training set was provided to the participants for extractor development, and the test set was used to evaluate the quality of the participants' solutions. An additional set of 25,000 unlabeled tweets (randomly selected but with no overlap with the 259 mentioned above) was also available to the participants.

[2]These interns were expected to produce a publishable work in their short three month internship.

Table 3. Extraction quality for the extractors built by the participants.

	Participants			
Quality	1	2	3	4
Precision (%)	96.26	97.00	**97.96**	96
Recall (%)	**86.55**	81.51	80.67	60.50
F_1 (%)	**91.15**	88.58	88.48	74.23
Num of Rules Built	16	37	58	14

Measures. We measured an extractor's quality based on the following standard metrics: $precision = \frac{tp}{tp+fp}$ (also known as *accuracy*), $recall = \frac{tp}{tp+fn}$ (also known as *coverage*), and $F_1 = 2 \bullet \frac{precision \bullet recall}{precision+recall}$, where tp denotes the number of *true positives*, fp denotes the number of *false positives* and fn denotes the number of *false negatives*.

Hypothesis

Our hypothesis was that the participants would find the UI intuitive and easy to use. We expected them to perform well on the given task and to become reasonably proficient with using text analytics to address high-level tasks with AQL and WizIE. In addition, after a brief exposure to the tool, the participants should understand how to build extractors to accomplish a high-level task and how to evaluate the quality of an extractor.

Performance Results

The participants produced extractors consisting of 31 rules on average. Table 3 lists the quality of extractors on the test set. As can be seen, the precision scores of all the extractors are above 96%, their recall scores range from 60.5% to 86.66%, and the best F_1 score is above 90%.

We regard the quality of the extractors built by the participants as surprisingly good, especially considering the short amount of time spent on the task. Text analytics over Twitter has been widely regarded as extremely challenging. In fact, the extraction quality reported for similar tasks over tweets is usually around 80% for F_1 [28, 29, 40]. While the task used in our study is more restrictive than the previous ones, two expert AQL developers on our team note that the best extractor built by the participants is comparable to what they would build for the same task.

Subjective Results

Questionnaires. The participants reported that they spent a range of 1 to 15 hours completing the task, with an average of 6.8 hours. The overall ease of use ratings for the UI

Table 1. Example tweets containing matches of song names (highlighted in bold)

```
RT @ardanradio #NowPlaying FUN feat Janelle Monae - We Are Young | #RIAUW
RT @arieladriane:  @1DirectionIndo what makes you beautiful - one direction cover by glee.  http://t.co/t4BmvZbM
@Cimorelliband @LisaCim @LaurenCimorelli payphone was amazing can you guys please do we are young by fun!!  Thanks.
RT @Jadore1Dx:  Dear Mothers &amp;amp; fathers of 1D - as The Wanted would say, im glad you came.
RT @Melisaaa11:  My boyfriend knows hes jealous of my relationship with Justin Bieber
Now u just somebody that I used to know!
```

were on the low side (2.8 on a scale of 1 to 5, with 5 being very easy). The ease of use for creating extractors was rated slightly higher (3.33 on the same scale). Reasons for these ratings were probed for in the interviews.

Interviews. All participants said they did all of the work the day before the deadline. The winner of the competition only started working on it after work at 6pm, while the deadline was at 11:59pm. Participants mostly worked by using the standard lab training material as a reference and did not search for additional resources from the help section in the tool, nor in the AQL language online reference.

All participants stated that the UI provided a good framework for getting started with the task. Additionally, participants called out the ease of use for creating dictionaries (an enhancement made in the last iteration). Specific quotes include: *"Because the process is very clear, the wizard is very easy to follow"*; *"[The tool] is quite helpful to analyze the sample data and define basic concepts. I used it extensively to create my dictionaries"*, and *"I did not face any problems using the tool"*.

According to the interviews, the reason for the relatively low ease of use rating of the UI was due to UI elements that were not part of our design goals. Specifically, participant stated that the steps required to measure precision and recall were too tedious, as the contest participants needed to repeatedly access this functionality in order to monitor their extraction performance on the training data. One participant complained that 13 clicks were needed to accomplish the action! Another said: *"It's pretty easy to create a workable extractor. However, it's very painful to create extractors with high performance. You need to analyze your results and keep refining your extractors"*.

Two participants who used the Extraction Plan did so for both organizing their code and for organizing text snippets and clues. A third participant used the Extraction Plan while becoming familiar with the UI, for example to create his first dictionary. He then moved on to using cut and paste during subsequent iterations of refining his extractors. None of the participants who used the Extraction Plan rated the tooling or the task as difficult.

In summary, both the user task performance and the user subjective reports give encouraging results that support our hypotheses. However, our study may represent a "best-case" scenario, with the possibility of our participants being better than average learners, an experiment with better control and larger subject size would allow us to obtain a more conclusive understanding of the impact of our UI design on novice developers of text analytics.

DISCUSSION

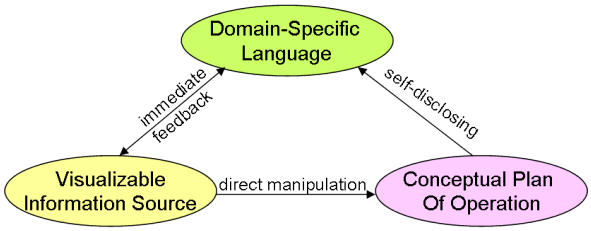


Figure 4. Lower Barrier to Learning Domain Specific Languages

Through a series of design and evaluation iterations, we have succeeded in building tools to bring a novice developer of text analytics to a level of proficiency where she can write high quality extractors. We took inspiration from a variety of user centered design principles and learning theories. The results were two unique tools, the Workflow Guide and the Extraction Plan, which effectively work together to lower the barriers to text analytics development.

It is our contribution to leverage such a model driven approach in helping novice users bridge the gap between high-level task requirements and concrete implementations. We believe this approach is not only applicable to text analytics, but also to other domain specific languages that operate on visualizable sources of information. Figure 4 illustrates this general idea. By bringing the primary source information to the forefront, we bridged the gulf of execution, so that users see the information pieces that they are working on and view the changes happening on them immediately. Users can then create and modify a conceptual plan of operations on those information pieces through direct manipulation. The conceptual plan self-discloses the underlying domain specific languages, so that users can harness the full power of the language gradually.

We are continuing to refine and iterate on this interface to make it even more usable. We are also working to make progress on the hard problem of bi-directional synchronization between the Extraction Plan and the application code. Finally, designing novel user interfaces for enabling *non-programmers* to develop complex text analytics solutions is our next challenge.

CONCLUSION

In this paper, we have demonstrated a new approach to lowering the barriers to acquire expertise in a complex area of software development. This approach integrates user centered design principles, learning theories, and model driven development methods. Working with the domain of text analytics, a field with high barriers to entry, we show that novice developers can quickly reach a high level of expertise using our

tools. We hope that the applications of this approach can be explored in other domains.

ACKNOWLEDGMENTS

We are grateful to Eser Kandogan for his contributions to the first version of the system. We also thank the participants in our user studies for their contribution, Brent Hailpern for sponsoring the programming competition, and the anonymous reviewers for their insightful comments.

REFERENCES

1. Brauer, F., Rieger, R., Mocan, A., and Barczynski, W. M. Enabling information extraction by inference of regular expressions from sample entities. In *CIKM* (2011).
2. Burnett, M., and Gottfried, H. Graphical definitions: Expanding spreadsheet languages through direct manipulation and gestures. *ACM Transa. on Comput.-Hum. Interact. (TOCHI) 5*, 1 (1998), 1–33.
3. Chiticariu, L., Krishnamurthy, R., Li, Y., Raghavan, S., Reiss, F., and Vaithyanathan, S. SystemT: an algebraic approach to declarative information extraction. In *ACL* (2010).
4. Crifasi, T. Hold your thoughts, SAP and NetBase will soon analyse your sentiments in the Cloud. Post in Bluefin Solutions Insights Blog, published on 12/15/2011, last accessed on 09/12/2012, http://bit.ly/vCpLgH.
5. Cunningham, H., Maynard, D., Bontcheva, K., and Tablan, V. GATE: an architecture for development of robust hlt applications. In *ACL* (2002).
6. Cypher, A., and Halbert, D. *Watch what I do: programming by demonstration*. 1993.
7. DiGiano, C., and Eisenberg, M. Self-disclosing design tools: a gentle introduction to end-user programming. In *DIS* (1995).
8. Doan, A., Gravano, L., Ramakrishnan, R., and Vaithyanathan, S. Special issue on managing information extraction. *SIGMOD Record 37*, 4 (2008).
9. Ennals, R., Brewer, E. A., Garofalakis, M. N., Shadle, M., and Gandhi, P. Intel Mash Maker: join the web. *SIGMOD Record 36*, 4 (2007), 27–33.
10. Gro-Klumanna, A., and Hautschb, N. When machines read the news: Using automated text analytics to quantify high frequency news-implied market reactions. *Journal of Empirical Finance 18*, 2 (2011), 321340.
11. Guo, P. J., Kandel, S., Hellerstein, J. M., and Heer, J. Proactive wrangling: mixed-initiative end-user programming of data transformation scripts. In *UIST* (2011).
12. Guzdial, M. Softwarerealized scaffolding to facilitate programming for science learning. *Interactive Learning Environments 4*, 1 (1994).
13. Hailpern, B., and Tarr, P. Model-driven development: The good, the bad, and the ugly. *IBM systems journal 45*, 3 (2006), 451–461.
14. Hundhausen, C. D., Farley, S. F., and Brown, J. L. Can direct manipulation lower the barriers to computer programming and promote transfer of training?: An experimental study. *ACM Trans. Comput.-Hum. Interact. 16*, 3 (Sept. 2009), 13:1–13:40.
15. Hutchins, E., Hollan, J., and Norman, D. Direct manipulation interfaces. *Human-computer interaction 1*, 4 (1985), 311–338.
16. IBM. Content Analytics with Enterprise Search Version 3.0. http://ibm.co/tun1ta, 2012.
17. IBM. InfoSphere BigInsights Version 2.0: https://ibm.biz/bdxmmt, 2012.
18. IBM. InfoSphere Streams Version 3.0: https://ibm.biz/bdxmmk, 2012.
19. Kahn, K. Toontalktm–an animated programming environment for children. *Journal of Visual Languages & Computing 7*, 2 (1996), 197–217.
20. Kandel, S., Paepcke, A., Hellerstein, J., and Heer, J. Wrangler: interactive visual specification of data transformation scripts. In *CHI* (2011).
21. Kelleher, C., and Pausch, R. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *CSUR 37*, 2 (2005), 83–137.
22. Ko, A., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., et al. The state of the art in end-user software engineering. *CSUR 43*, 3 (2011), 21.
23. Ko, A., Myers, B., and Aung, H. Six learning barriers in end-user programming systems. In *VL/HCC* (2004).
24. Leshed, G., Haber, E., Matthews, T., and Lau, T. Coscripter: automating & sharing how-to knowledge in the enterprise. In *CHI* (2008).
25. Li, Y., Chu, V., Blohm, S., Zhu, H., and Ho, H. Facilitating pattern discovery for relation extraction with semantic-signature-based clustering. In *CIKM* (2011).
26. Li, Y., Krishnamurthy, R., Raghavan, S., Vaithyanathan, S., and Jagadish, H. V. Regular expression learning for information extraction. In *EMNLP* (2008).
27. Liu, B., Chiticariu, L., Chu, V., Jagadish, H. V., and Reiss, F. R. Automatic Rule Refinement for Information Extraction. *PVLDB* (2010).
28. Liu, X., Zhang, S., Wei, F., and Zhou, M. Recognizing named entities in tweets. In *HLT* (2011).
29. Liu, X., Zhou, M., Zhou, X., Fu, Z., and Wei, F. Joint Inference of Named Entity Recognition and Normalization for Tweets. In *ACL* (2012).
30. Lowe, W. Software for Content Analysis: A Review. bit.ly/QMCC4M (accessed on September 12, 2012).
31. Mack, R. L., Mukherjea, S., Soffer, A., Uramoto, N., Brown, E. W., Coden, A., Cooper, J. W., Inokuchi, A., Iyer, B., Mass, Y., Matsuzawa, H., and Subramaniam, L. V. Text analytics for life science using the unstructured information management architecture. *IBM Systems Journal 43*, 3 (2004), 490–515.
32. Mayer, R. E. The psychology of how novices learn computer programming. *ACM Comput. Surv. 13*, 1 (1981), 121–141.
33. Norman, D. *The design of everyday things*. 2002.
34. Pane, J., Myers, B., and Miller, L. Using HCI techniques to design a more usable programming system. In *HCC* (2002).
35. Papert, S. *Mindstorms: Children, computers, and powerful ideas*. 1980.
36. Patel, K., Bancroft, N., Drucker, S., Fogarty, J., Ko, A., and Landay, J. Gestalt: Integrated support for implementation and analysis in machine learning processes. In *UIST* (2010).
37. Patel, K., Fogarty, J., Landay, J., and Harrison, B. Investigating statistical machine learning as a tool for software development. In *CHI* (2008).
38. Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., and Paterson, J. A survey of literature on the teaching of introductory programming. In *ACM SIGCSE Bulletin*, vol. 39 (2007), 204–223.
39. Quintana, C., Reiser, B. J., Davis, E. A., Krajcik, J., Fretz, E., Duncan, R. G., Kyza, E., Edelson, D., and Soloway, E. A scaffolding design framework for software to support science inquiry. *Journal of the Learning Sciences 13*, 3 (2004).
40. Ritter, A., Clark, S., Mausam, and Etzioni, O. Named entity recognition in tweets: An experimental study. In *EMNLP* (2011).
41. SAP. BusinessObjects Text Analysis: Text Services Platform Users Guide: http://bit.ly/pgjyml, 2012.
42. Soderland, S. Learning information extraction rules for semi-structured and free text. *Machine Learning 34*, 1-3 (1999), 233–272.
43. Text Analysis International Inc. Integrated development environments for natural language processing, 2012.
44. Wagstaff, K. Machine Learning that Matters. In *ICML* (2012).
45. Wong, J., and Hong, J. Making mashups with marmite: towards end-user programming for the web. In *CHI* (2007).
46. Wood, D., Bruner, J., and Ross, G. The role of tutoring and problem solving. *Journal of Child Psychology and Psychiatry 17* (1976), 89–100.