# Predictive Interaction for Data Transformation

Jeffrey Heer
U. Washington & Trifacta Inc.
jheer@uw.edu

Joseph M. Hellerstein
UC Berkeley & Trifacta Inc.
hellerstein@berkeley.edu

Sean Kandel
Trifacta Inc.
skandel@trifacta.com

## ABSTRACT

The human work involved in data transformation represents a major bottleneck for today's data-driven organizations. In response, we present Predictive Interaction, a framework for interactive systems that shifts the burden of technical specification from users to algorithms, while preserving human guidance and expressive power.

## 1. INTRODUCTION

Many of today's data management challenges stem from the increasing variety of scenarios where data is being exploited. This variety has a number of dimensions: breadth of data *sources* with varying representation and quality, breadth of *use cases* for a range of both technical and popular products, and breadth of *people* with varying skill sets who work with data sources to build data products.

The human factors in this context present challenges and opportunities of increasing urgency for the technical community. Recent projections on labor markets predict dire shortfalls in the analytical talent available to realize the potential of "big data" [22]. Technologists can have significant impact here by developing techniques that dramatically simplify labor-intensive tasks in the data lifecycle.

In interviews with data professionals, we discovered that a majority of their time is spent in data transformation tasks, which they consider to be relatively unsophisticated and repetitive, yet persistently tricky and time-consuming [15]. The most-cited bottleneck arises in manually wrangling the various sources of data required for each distinct use case.

The technical challenges of data transformation come from the unbounded variety of inputs and outputs to the problem, which so often necessitate custom work. At heart, data transformation is a domain-specific programming problem, with a strong focus on the structure, semantics and statistical properties of data. If we make the specification of data transformation programs dramatically easier, we can remove drudgery for scarce technical workers, and engage a far wider labor pool in time-consuming, data-centric tasks.

Data transformation is not a new problem, though it is one of increasing urgency and complexity. It covers a range of tasks, including statistical manipulations (e.g., profiling, outlier handling, imputation), restructuring (e.g., extracting fields from text, (un)nesting, data-to-metadata transformations like Pivot/Unpivot), cleaning (e.g., standardization, entity resolution, dictionary management), enrichment (e.g., joins and references), and distillation (e.g., sampling, filtering, aggregation, windowing).

There are a number of recurring themes in previous work on data transformation. One is to develop new *user interfaces* for graphical specification of queries and transforms [11, 23, 26, 32], and visualization of outputs [3, 19]. A second theme is to innovate at the *data management* layer, with domain-specific languages that are well-suited to data transformation and can be executed in a scalable, high-performance manner [8, 18, 28]. And despite the wide-ranging, ad hoc nature of these tasks, there is a persistent theme in the literature on developing new *computational methods* for automation powered by AI, logic and crowdsourcing [5, 7, 29].

Building on this body of work, we take the position in this paper that the burden of specifying data transformation logic can be lifted via approaches that adopt a thoughtful "trifecta" of these three research streams of People, Data and Computation. Our goal here is not to prescribe a specific solution to data transformation. Rather, we present a general design framework that we call *Predictive Interaction*, which relieves users from the burden of technical specification. In Predictive Interaction, the user is not required to specify the details of their data transformations; they can instead highlight features of interest in data visualizations. These features help guide predictive methods to suggest a variety of possible next steps for their data. The user then decides on the best next step, and the interaction repeats. We call this the *guide/decide* loop of Predictive Interaction. In the field, users have reported order-of-magnitude productivity gains based on this technology, as well as the ability to let less-technical end-users wrangle their own data, avoiding time-consuming back-and-forth discussions with IT professionals [30]. In this paper we highlight the benefits of Predictive Interaction, as well as key design considerations for systems based on our approach.

### 1.1 Case Study: Pattern Extraction

We begin our discussion with a concrete example of a small subtask in data transformation — *text pattern extraction* — and a Predictive Interaction approach we developed to address it.

Text pattern specification is a basic step in many data wrangling tasks, including substring extraction, delimiter identification, and the specification of filters. Typically, domain-specific pattern languages like Regular Expressions (REs) are used for this purpose. But REs are hard for non-experts to specify, and can often be challenging for experts to read and debug.

During the design of the data transformation interface in Trifacta, we developed a Predictive Interaction approach to specifying text patterns for manipulating data. Figure 1 shows an example of this design in use on mobile advertising logs. Users are presented with

This article is published under a Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/), which permits distribution and reproduction in any medium as well allowing derivative works, provided that you attribute the original work to the author(s) and CIDR 2015.
*7th Biennial Conference on Innovative Data Systems Research (CIDR '15)* January 4-7, 2015, Asilomar, California, USA..

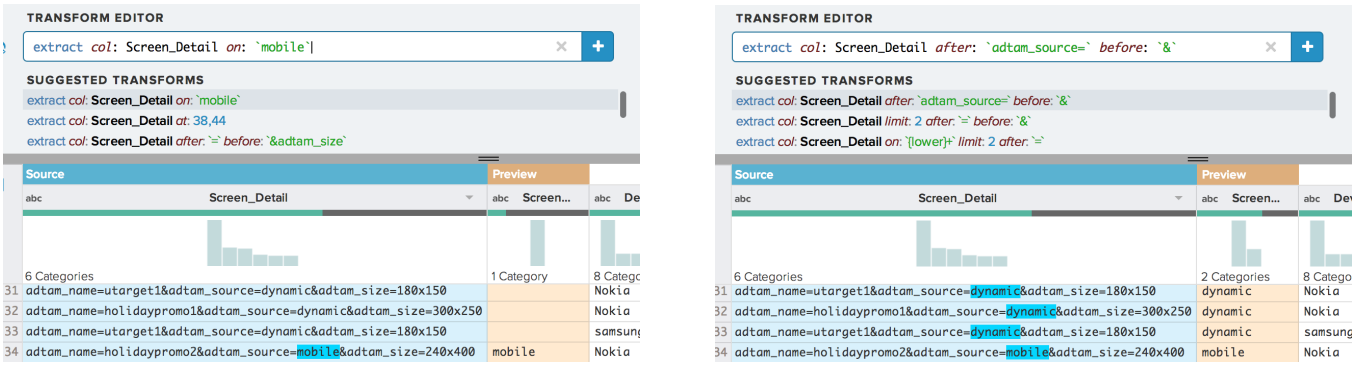


**Figure 1: Predictive Interaction for text pattern specification. The left image shows the interface after the user has highlighted the string `mobile` in line 34. The right shows the interface after one more gesture: highlighting the string `dynamic` in line 31. Note that the top-ranked suggested transform changes after the second highlight, and hence so do the Source and Preview contents.**

```
/(?<=adtam_source\=)[^\&]*(?=\&)/
/(?<=\=)[^\&]*(?=\&)/
/(?<=\=)[a-z]+/
```

**Figure 2: A ranked list of regular expressions.**

a visual rendering of their data in a familiar tabular grid. They can guide the system by highlighting substrings in the table, which are added to an example set. Based on this set, an inference algorithm produces a ranked list of suggested text patterns that model the set well. For the top-ranked pattern, the table renderer highlights any matches found, and shows how those matches will be used.

Figure 1 shows the states of the interface after the user makes each of two guiding interactions: first, highlighting the string `mobile` in row 34, and then highlighting the additional string `dynamic` in row 31. The user interface shows the highlighted patterns in the source (blue), and the outcome of a text extraction transform in a preview column (tan). The user can choose to view the outputs of other suggested transforms by clicking on them in the top panel; they can also edit the patterns directly in a Transform Editor. When the user decides on the best pattern, they can click the "plus" (+) to the right of the transform to add it to a DSL script.

In our initial prototype the suggested transforms looked different than what is shown in Figure 1. Originally, users would see a ranked list of REs in a traditional syntax, as shown in Figure 2 (corresponding to the ranked list of suggested transforms on the right of Figure 1). In user studies we found that even experienced programmers had difficulty deciding quickly and accurately among alternative REs. It seems that RE syntax is better suited to *writing* patterns than to *reading* them. Hence we changed our DSL to a new pattern language (compilable to REs) that is better suited to rapid disambiguation among options.

In essence, we evolved our DSL design to simplify the way that users can interact with automated predictions. Although simple, this example illustrates some of the subtleties involved in co-designing Predictive Interaction across the three streams of traditional research mentioned above. The visualization has to be informative and the affordances for user guidance clear; the predictive model has to receive information-rich guidance from the interactions, and do a good job of surfacing probable but diverse choices; the DSL has to be expressive yet sufficiently small for tractable inference and simple user interaction.

In the remainder of the paper, we provide a general framework for Predictive Interaction, putting it in context with previous approaches to visual languages for managing data, and highlighting research challenges and opportunities for the community.

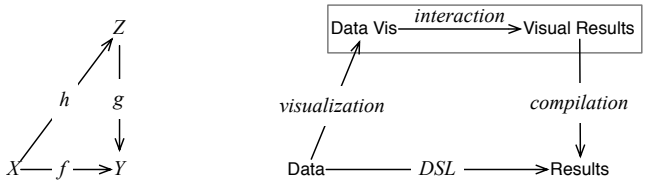


**Figure 3: Lifts. A traditional lift (a): given a map $f : X \to Y$, and a map $g : Z \to Y$, the lifting problem is to find a map $h : X \to Z$ such that $g \circ h = f$. Lifting in the context of visual specifications (b): rather than write expressions in a textual DSL, we define a lift to a domain of data visualization and interactions, such that the interactions in that domain lead to final outputs: $compilation \circ interaction \circ visualization = DSL$ programming.**

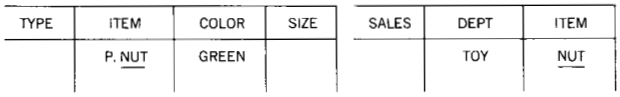


**Figure 4: Query By Example: qualified retrieval using links [32].**

## 2. LIFTING TO VISUAL LANGUAGES

To set the stage for our discussion, we re-examine the more traditional integration of two of our three themes: visualization and data-centric languages. There are a number of influential prior efforts along these lines, including Query-By-Example (QBE) [32], Microsoft Access, and Tableau. These interfaces take a textual data manipulation language (e.g., relational calculus) and "lift" it into an isomorphic higher-level visual language intended to be more natural for users. Given a visual specification of a query, a system can translate ("ground") to the domain of the textual language for processing. Lifting is a basic idea from category theory, sometimes used in the design of functional programming languages (Figure 3).

Lifting to a visual domain has proven to be useful for the specification of standard select-project-join-aggregate queries. As illustration, we review two influential systems: QBE and Tableau.

**Example 1: QBE.** The main idea in QBE is to lift the database

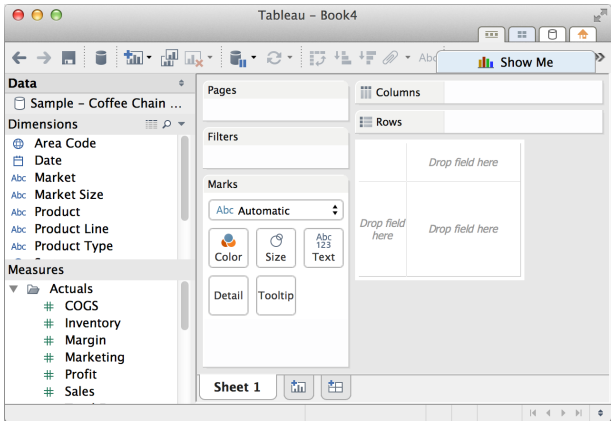


**Figure 5: The Tableau interface for query specification.**

into a visual representation of relational metadata: table grids with column headers. The user interacts by starting with an empty "skeleton" of a table grid, and types a table name at the upper left of the skeleton, which causes the associated column names to be populated. The user can then place variables and expressions into the empty cells in the rows beneath the header. The placement of the variables and expressions in these empty cells precisely determines a query, which can be compiled to SQL in a straightforward manner, and executed in a relational database. Figure 4 shows a query from Zloof's 1977 paper [32] that illustrates "qualified retrieval using links": a query with selection predicates (`TYPE.COLOR = 'GREEN', SALES.DEPT = 'TOY'`) and an equijoin (`TYPE.ITEM = SALES.ITEM`). By typing `P.`, the user has indicated that they wish to print the `TYPE.ITEM` field in the output, which is also shown in a tabular format.

**Example 2: Tableau.** Tableau (Figure 5) is a data visualization tool rooted in Stolte et al.'s Polaris system [28] for specifying and visualizing results of pivot table queries. Again, the visual metaphor focuses on the manipulation of metadata. Users are shown the schemas of tables, with attributes partitioned into the familiar OLAP notion of dimensions (categorical types) and measures (numeric types). Users can drag the names of attributes to "shelves" on a visual canvas. These actions indicate a desire to group or filter records, and to visualize them by spatial position (row/column), color, shape or size. Users can further tailor the results by selecting aggregation functions and visualized mark types (bars, plotting symbols, etc) using drop-down menus. Tableau's interface is a direct lifting of a DSL called VizQL, an intermediate high-level language that in turn compiles to both database queries (e.g., SQL or MDX) and visual encoding procedures.

## 2.1 Discussion

Tableau and QBE (as manifested in commercial variants like MS Access) have proven to be popular and approachable for both business and technical users. There are notable benefits to the visual lifting that they achieve. First, by grounding user interactions in an underlying DSL, these tools couple graphical specification with the ability to compile down to widely-deployed scalable execution engines. Contrast this approach to other interactive data tools that are not grounded in a DSL, such as OpenRefine: the standalone nature of those tools limits the data and contexts to which they can be applied, isolating them from third-party investments in improved engine infrastructure.

Next, these visual tools can be easier to learn than the corresponding DSLs, due in part to visual *affordances*: cues in the interface

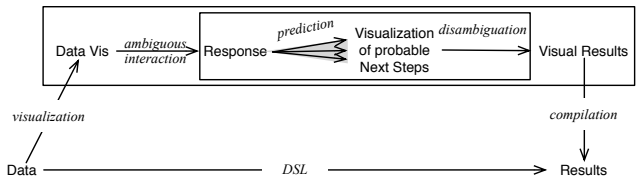


**Figure 6: Predictive Interaction: The Guide/Decide loop. We augment previous lifting schemes to *guide* the system via ambiguous interaction, visualize a predicted distribution of next steps, and *decide* on their desired results. More generally,** $compilation \circ disambiguation \circ prediction \circ interaction \circ visualization = DSL$**.**

that tell users what actions are possible (menus, icons, drag/drop targets, etc.). These affordances make the significant constructs in the language readily apparent.

Another advantage of these visual languages is a more flexible specification style. Textual language expressions are fundamentally sequential in format, written and read from start to finish. Visual interfaces can loosen the order in which users construct specifications — and, critically, can show intermediate results along the way, to help users assess progress and refine their goals as they go. In addition, a 2-dimensional layout can allow for better clustering of concepts and hence aid comprehensibility: for example, tables that are being joined might be shown close together on screen along with their associated join predicates.

Visual lifting can be helpful, but it has a significant built-in limitation: it presents a strict isomorphism to a textual language. This raises multiple problems. Textual languages are often quite rich, and most visual languages (including QBE and Tableau) provide visual metaphors for only a subset of their underlying DSLs. For example, in both these languages, arithmetic expressions and string manipulation of the sort we saw in the previous section require explicit textual specification. In essence, the user occasionally hits a "cliff" of complexity in the user interface, where they fall out of the visual domain and must work in an underlying DSL.

More importantly, the visual interaction model does not inherently remove the user's specification burden; *it merely changes the syntax*. In order to achieve a certain query, the user must make visual specifications that are precisely as descriptive as any textual specifications for that query. Unfortunately, in many instances a user's inability to make progress with an interface is not fundamentally a matter of syntax (e.g., typing vs. mousing), but rather an inability to specify a precise intent. This inability can stem from technical unfamiliarity (not knowing *how* to do something), or from an ambiguity of intent (uncertainty about *what* to do). Note that ambiguity is often desirable — it is at the heart of exploratory tasks where the computer is serving as an aid for the analyst's creative process.

Going beyond the inherent limitations of visual lifting requires a new approach, one in which computational methods assist the user in the task of specification.

# 3. PREDICTIVE INTERACTION

The key drawback of both DSLs and lifted visual interfaces is an intolerance of ambiguity — they ask users for too much technical specificity, or at best limit expressivity. By contrast, the key drawback of automated techniques is the absence of user guidance, which prevents automated techniques from adjusting to a wide variety of inputs and outputs. In this section we show how this tension can be resolved by incorporating aspects of all three of our technical

themes: People, Data and Computation.

Figure 6 shows the basic structure of Predictive Interaction as an augmentation of our earlier illustration of visual lifting. Rather than ask users to visually specify exact desires, they provide ambiguous input to *guide* predictive methods, and *decide* on a desired result. We explain the steps of the diagram with reference to our case study of text pattern specification in Section 1.1.

Once again, we use *visualization* to lift from the domain of data and code into a more intuitive visual domain. In Figure 1, the data is rendered in a familiar table visualization with each column augmented with univariate histograms and data quality bars [16].

The next step is user *interaction*. The first and most prominent distinction brought by Predictive Interaction is that the visual interactions can be highly ambiguous, merely guiding the system toward features of interest in the data. For example, in Figure 1, the user simply highlights substrings in the table visualization, indicating an interest in certain textual features. Many other aspects of the visualization are available for interaction as well, including row and column headers, histogram bars, and data quality summaries.

As user guidance is received, the system can provide a transient response to acknowledge the user's interaction (e.g., by highlighting text during mouse drag). Then, based on a variety of features — from the interaction, the data, historical information and more — Predictive Interaction computes a distribution of *probable next steps*. This space of next steps is visualized in a manner that allows the user to explore among them and *disambiguate* the uncertainty introduced by prediction. In Figure 1, the user can browse through a ranked list of DSL statements (Suggested Transforms), and by clicking on each suggestion they can see its results visually in the table preview. Once the user decides on the next step, it can be compiled down to the grounded syntax of the DSL.

Compare this model with the visual lifting systems described in Section 2. In both cases, the user benefits from visual affordances and graphical layout. However, with Predictive Interaction users are freed from the burden of fully specifying their intentions in using these affordances. Instead, algorithms can help with the specification, and users can focus on resolving any resulting ambiguity. Of course Predictive Interaction also raises technical challenges in translating user interactions into meaningful specifications.

In Figure 6 we cast Predictive Interaction in the context of visual lifts, but this viewpoint is not intended to be sacrosanct. In fact, in our case study of Section 1.1, you can see that we broke out of the pattern of Figure 6 somewhat. Next steps in Figure 1 are not purely in the visual domain: they are shown as both grounded DSL statements (Suggested Transforms) and higher-level visual previews. Users disambiguate by choosing among these grounded transforms. Note that the experience of the DSL in Figure 1 is quite different than the "cliff" we discussed in Section 2.1; thanks to Predictive Interaction, it provides a smooth learning curve for the DSL. Novice users can click on the suggested transforms in turn, and choose the one whose previewed outputs match their desires. As users gain experience, they can pay more attention to the DSL text and how it associates with the output they see previewed. Expert users understand the details of the suggested transforms, and can optionally edit them — with the added benefit of instantaneously visualizing the effects of their edits via previews.

Other variations on the model of Figure 6 are natural. For example, we need not insist on a purely visual domain of interaction; we could also allow speech, natural language, and other interface modalities. We may also want to "skip steps": for example, Proactive Wrangling [10] is an approach that goes straight from visualization to suggested next steps without waiting for any interaction.

Independent of specific variations, the core inner *guide/decide* loop of Predictive Interaction is the sequence of steps shown in Figure 6: Visualize, Interact, Predict, Disambiguate, and Compile. In practice, most transformation tasks require multiple iterations of this loop in order to get data into usable shape. As a result, it is helpful for users to be able to fluidly work through the loop with a relatively consistent user interaction.

In Section 1.1 we saw an example of the guide/decide loop providing lightweight and intuitive specification of REs. Obviously, data transformation in the large requires a richer language than REs. Trifacta is grounded in a DSL we designed called *Wrangle*, which builds on a lineage of previous languages in the research literature [16, 18, 26]. Wrangle includes text selection patterns (including REs) as a sub-language, in support of a wide range of tasks: the single-source transformations supported by Potter's Wheel and Data Wrangler, structural transformations for nested data like JSON and XML, multi-source transformations like joins and unions, data-to-metadata transformations like Pivot/Unpivot, and various data cleaning operations for numerical and textual data. Although we glossed over it in our earlier discussion, Figure 1 shows the Predictive Interaction approach generating complete Wrangle statements (and outputs): specifically, a variety of "Extract" statements with embedded text selection patterns.

## 3.1 Benefits

Predictive Interaction offers many of the benefits of other visual interaction metaphors, but the addition of predictive methods in the interface adds significant advantages.

By freeing the visual language from providing full coverage of the textual language, Predictive Interaction allows the interface designer to keep the affordances for specification lean. This enables designs where data visualizations can remain a consistent, predominant interface during all phases of the guide/decide loop — including interaction, previews of possible next steps, and disambiguation. It fits naturally with interfaces for directly manipulating data tables and visualizations, rather than more traditional interfaces that focus on schemas, workflows, menus and textual languages. The resulting experience promotes a fluid, agile working style in a unified visual environment.

In our experiences with novice users, we have seen both anecdotally and empirically that the tolerance of ambiguity inherent in Predictive Interaction helps users overcome conceptual hurdles: suggestions help them see what is possible [10]. We have also seen Predictive Interaction help experienced users save significant time: in most cases they can simply gesture a general intent, and have the computer synthesize detailed specifications. In controlled user studies [16], we found that even for very small data amenable to manual manipulation, a Predictive Interaction approach led to significant performance benefits (at least 2x faster on median) over traditional approaches. In our experience in the field, experienced data transformation users have attested to over 10x improvements in task completion upon adoption of a Predictive Interaction solution [30]. This directly addresses a common complaint regarding the drudgery involved in data transformation, by shifting the burden of specification from users to algorithms.

Perhaps most interestingly, users have commented that Predictive Interaction enables much more free-wheeling exploration: starting by highlighting interesting features of the data, they can explore the effects of a variety of transformations that relate to those features.

# 4. DESIGN CONSIDERATIONS

We believe that Predictive Interaction models can be used to alleviate a wide variety of data-centric problems where users face technical bottlenecks of programming or scripting. Addressing new

problems will require different design choices than the the ones we have made to date for data transformation. Predictive Interaction provides a framework for making these design choices.

The basic prescription for designing a new Predictive Interaction system is to co-evolve all three aspects of the technology. First, choose or design a target DSL for the domain, which is amenable to agile, stepwise specification. Second, choose an intuitive lifted domain for the data, with affordances for input to *guide* predictions, and to *decide* on next steps. Third, design and train prediction models that take inputs from the data and interface and produce a distribution of candidate steps in the DSL. In our experience designing systems like Data Wrangler and Trifacta, the co-design of these three aspects is best served by an iterative design style, as changes in one aspect often require changes in another.

At a finer level of detail, domain-appropriate Predictive Interaction systems raise a number of design challenges across human-computer interaction, database languages and systems, and machine learning methods. We outline a number of those challenges here.

## 4.1 Data: Grounding to a DSL

A well-designed DSL is key to reining in the complexity of the guide/decide process for both people and computation.

Domain-specific languages aid critically in the tractability of our prediction problems. In the context of Predictive Interaction, keeping the DSL small helps considerably with the development of inference algorithms for predicting user desires. A small language makes for a smaller search space in which to do inference; well-orthogonalized language constructs make it easier to train a model that effectively separates the constructs. And the simpler the language, the easier it is to compile to scalable data processing engines — preferably multiple such engines for portability.

From a human perspective, a good DSL helps with the conceptual challenge of disambiguation. In his introduction to DSLs, Hudak notes:

> A user immersed in a domain already knows the domain semantics. All the DSL designer needs to do is provide a notation to express that semantics [13].

Hudak's paper is focused on traditional textual specification with a DSL, but his comments apply even more to the Predictive Interaction context. If the DSL is a good match to the domain (data transformation, in our case), then the user gets the same benefits as the inference algorithm: a relatively small number of visual outcomes are possible, so they are easy to disambiguate. Clean interface design for disambiguation can often follow relatively naturally. For example, candidate statements in a small, well-orthogonalized language are easy to disambiguate in interfaces like N-best lists — whether they are represented literally (as in the Suggested Transforms of Figure 1), or via icons. Similarly, if the space of possible outcomes is small and well differentiated, then previews of those outcomes should be easy to distinguish visually — via exploration (as in Figure 1), or via small-multiple visualizations of many alternative outcomes.

## 4.2 People: Visualization and Interaction

The user interface for any Predictive Interaction system has to integrate design aspects along two dimensions. Functionally, it needs to enable users to perform both the *guide* and *decide* tasks — the former being an ambiguous specification, and the latter a unique specification. Visually, it needs to provide users with both *data visualization* and *data manipulation* features. Customizing for the cross-product of these dimensions (guide/decide $\times$ visualization/manipulation) leads to as many as four different interaction designs, one for each pairing. However, we might prefer to drive to a single user experience that supports agile movement through the guide/decide loop, with a unified "direct manipulation" experience for visualizing and manipulating data. Prior work has tended toward multi-modal interfaces targeting a mixture of these four design points. These design tradeoffs deserve further study.

As the first step in interaction, the input affordances for the *guide* phase deserve special attention — particularly because input ambiguity is a difficult design goal to scope. From a usability perspective, there are advantages to keeping the guidance interface minimal. Still, the user needs to feel that they can control the software in an intuitive and repeatable fashion. This problem is compounded by the potential feature space being quite large. To form sentences in our DSL, we might want users to highlight data features (the objects of sentences) or behaviors (the verbs), or both. Data highlighting can range from the small (e.g., example values or substrings) to the large (e.g., visualizations or textual representations of statistical patterns), potentially across both data and metadata. Depending on the user persona we design for, we need to balance a variety of concerns including what helps the user (ease of learning and memorization, efficiency of operation) and what's good for the predictive algorithm (ease of prediction, clear diversity of options for disambiguation.)

The visual language of the *decide* phase also merits discussion. There are many ways to visually surface a single suggested transform: by showing the transform itself (e.g., DSL syntax or iconography), by visualizing the data after the transform, or even visualizing the change to the data performed by the transform (e.g., visual diffs or animation). This design space becomes more complex when considering multiple suggested transforms that the user needs to disambiguate. In general, the goal is to help the user make an informed choice among the transforms, and adapt them as needed.

In our work we have often chosen to show both transforms and data for aiding in disambiguation. This coupling makes for a more information-rich user experience, at the expense of apparent complexity. We have considered hiding the textual transform specifications beneath an expert mode switch, but even moderately sophisticated users appreciate understanding some details of the transforms, and as noted in Section 3 there are learning-curve benefits for novice users in gradually observing transforms and their associations with outputs. In Section 1.1 we illustrated our work on the readability of Trifacta's syntax for text patterns, which was designed to break up complex patterns into composable clauses (on, before, after, etc.) When choosing to show transforms in any syntax — and especially when choosing to allow transform editing — it is useful to refer to the Cognitive Dimensions of Notations [9] as a broad-brush framework to both inform and evaluate alternative designs.

## 4.3 Computation: Predictive Methods

Domingos describes Machine Learning with the equation "Learning = Representation + Evaluation + Optimization" [6]. The learning problems in Predictive Interaction are rendered unique largely by their representation space: sentences in a DSL. As mentioned above, a critical component to the tractability of our learning task is the simplicity of this DSL.

As Domingos points out in the same paper, "feature engineering is the key" to practical Machine Learning [6]. In Predictive Interaction systems, we can acquire features immediately from user interactions over the data they are transforming. We can also leverage features from the past: previous data and user interactions, previous user selections for next steps, previous transformation scripts, etc. Finally, we can impact the features we will receive in the future by controlling the user interface: in an Active Learning style [27], we can drive the user to highlight features that will make learning more efficient. The design of the user interface can have significant impact on what features we gather, and in what order.

Evaluation is a particularly challenging problem in Predictive Interaction. Traditional optimization goals like error rates or precision/recall can be hard to pin down in this context. Given that many tasks span multiple statements in the DSL, it is not clear on a statement-by-statement basis whether the prediction model is guiding the user in the right direction. And in some cases the user's goals may be ambiguous to begin with. Other metrics like diversity of results are an important ingredient in the evaluation.

One design advantage of Predictive Interaction over certain other prediction problems (e.g., ranking results for web search queries) is the presence of the DSL as a fallback: in the worst case, advanced users can override bad predictions by typing in their intended specifications in detail. This fallback mechanism can be very helpful in allowing developers of Predictive Interaction systems work in a more agile fashion: while tuning their predictive models, they can continue refining the DSL and user experience. Co-design across components moves more quickly when the components can evolve side-by-side. As mentioned in Section 2.1, depending upon DSL specification too often in practice can lead to hazardous "cliffs" in the UI experience. This is why it is important to have an agile process and methods for learning from user experimentation. Horvitz's principles for mixed-initiative interfaces [12] are a useful source of design guidance here.

# 5. RELATED WORK

There is a long tradition of research and development in data transformation, visualization and interactive querying—some referenced above, most beyond the scope of this paper. Here we highlight some specific prior work that is related to Predictive Interaction.

Tableau's lifted interaction based on VizQL was one inspiration for this work. The current Tableau product offers a variety of non-predictive techniques to ease the difficulty of specifying visualizations. For example, it will automatically choose mark types and scale transforms based on features of data. These decisions are applied automatically based on deterministic rules, so there is no ambiguous visual "guide" step in Tableau's interface akin to that of Predictive Interaction, nor any probabilistic prediction to disambiguate. Tableau's "Show Me" [20] feature provides something of a visual "decide" interface, using its rules to generate multiple alternative visualizations for a set of dimensions and measures.

There is a wide variety of work on "wrappers" and "mediators" that addresses a specific form of data transformation focused on mappings, often from irregular tree-shaped web sources to structured databases. The text by Doan, Halevy and Ives covers a variety of this work [4], including sections on interactive and learning methods. This is useful background material for considering Predictive Interaction designs. This body of work is rather light in its treatment of interaction design; the focus is largely on the design of inference algorithms for identifying extraction rules from tree-structured sources like the DOM. One example that discusses a specific interaction proposal is the CLIDE system, which provides an augmented version of a Microsoft Access-like schema-level UI for specifying wrappers that guides users through the wrapping process [24]. A representative example of a more predictive, direct-manipulation approach is the work of Irmak and Suel [14].

Abouzied, et al. describe a visual, learning-based trial-and-error interface called DataPlay for specifying and "tweaking" complex quantified queries in SQL [1]. The interface provides the user with both schematic and direct manipulation interfaces. The schematic specification language in this work is non-ambiguous, but the idea is that user specifications are often effectively ambiguous by being incorrect guesses. As a result, DataPlay allows the user to "tweak" their specification via iteratively scoring candidate answers and "non-answers"; a learning algorithm drives this feedback in concert with the user [2]. This model of "ambiguously incorrect specifications" is an interesting twist on the idea of approach.

The lifting in Predictive Interaction need not be to a visual domain; any high-level interaction model with ambiguity fits. Natural language interfaces are one potential alternative [25], perhaps coupled with recent work on translating queries and data to natural language at the output [17]. Gesture-based interfaces are another recently-proposed interface modality for query specification where Predictive Interaction could potentially be relevant when coupled with visualization [21].

Programming By Example (PBE) is a traditional paradigm for simplifying programming tasks by having users specify input/output pairs. Unlike Predictive Interaction, it does not include any aspect of lifting into another domain like visualization. A vein of recent work has applied PBE to data transformation tasks; the most relevant to this paper is the recent STEPS work of Yessenov et al. [31]. In addition to traditional PBE, STEPS presents an interface to highlight features of the textual inputs and outputs as "clues", and choose among potential statements in a DSL.

# 6. CONCLUSION

As Moore's Law progresses, human bottlenecks become the overarching cost for the vast majority of organizations working with data. The technical community has an imperative to tackle problems of human efficiency: productivity for expert data professionals, and accessibility of data technology for a broader population. Breakthroughs will not come solely from new UIs; the Database community has much to offer and to learn from both HCI and AI on these fronts. A persistent challenge in this context is that so much work with data is bespoke: customized to specific data sets and a specific target usage. We see Predictive Interaction as a promising framework for making custom solutions easier to specify, assess, and productionalize in scalable infrastructure. Beyond data transformation, we also see potential for Predictive Interaction in statistical analysis, large-scale graph processing, data modeling and visualization, and processing of unstructured data types such as free text, photos and videos.

# References


[1] Azza Abouzied, Joseph M. Hellerstein, and Avi Silberschatz. "Dataplay: Interactive tweaking and example-driven correction of graphical database queries". In: *UIST*. ACM. 2012, pp. 207–218.

[2] Azza Abouzied et al. "Learning and verifying quantified boolean queries by example". In: *PODS*. ACM. 2013, pp. 49–60.

[3] Alexander Aiken et al. "Tioga-2: A direct manipulation database visualization environment". In: *ICDE*. 1996, pp. 208–208.

[4] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of data integration*. Elsevier, 2012.

[5] AnHai Doan and Alon Y Halevy. "Semantic integration research in the database community: A brief survey". In: *AI magazine* 26.1 (2005), p. 83.

[6] Pedro Domingos. "A few useful things to know about machine learning". In: *CACM* 55.10 (2012), pp. 78–87.

[7] Ahmed K Elmagarmid, Panagiotis G Ipeirotis, and Vassilios S Verykios. "Duplicate record detection: A survey". In: *TKDE* 19.1 (2007), pp. 1–16.

[8] Helena Galhardas et al. "AJAX: an extensible data cleaning tool". In: *ACM Sigmod Record* 29.2 (2000), p. 590.

[9] Thomas R. G. Green and Marian Petre. “Usability analysis of visual programming environments: a ‘cognitive dimensions’ framework”. In: *Journal of Visual Languages & Computing* 7.2 (1996), pp. 131–174.

[10] Philip J Guo et al. “Proactive wrangling: mixed-initiative end-user programming of data transformation scripts”. In: *UIST*. 2011, pp. 65–74.

[11] Mauricio A Hernández, Renée J Miller, and Laura M Haas. “Clio: A semi-automatic tool for schema mapping”. In: *ACM SIGMOD Record* 30.2 (2001), p. 607.

[12] Eric Horvitz. “Principles of Mixed-initiative User Interfaces”. In: *SIGCHI*. 1999, pp. 159–166.

[13] Paul Hudak. “Domain-Specific Languages”. In: *Handbook of Programming Languages* 3 (1997), pp. 39–60.

[14] Utku Irmak and Torsten Suel. “Interactive wrapper generation with minimal user effort”. In: *Proceedings of the 15th international conference on World Wide Web*. ACM. 2006, pp. 553–563.

[15] Sean Kandel et al. “Enterprise data analysis and visualization: An interview study”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2917–2926.

[16] Sean Kandel et al. “Wrangler: Interactive visual specification of data transformation scripts”. In: *SIGCHI*. 2011, pp. 3363–3372.

[17] Georgia Koutrika, Alkis Simitsis, and Yannis E Ioannidis. “Explaining structured queries in natural language”. In: *ICDE*. 2010, pp. 333–344.

[18] Laks VS Lakshmanan, Fereidoon Sadri, and Iyer N Subramanian. “SchemaSQL-a language for interoperability in relational multi-database systems”. In: *VLDB*. Vol. 96. 1996, pp. 239–250.

[19] Miron Livny et al. “DEVise: integrated querying and visual exploration of large datasets”. In: *ACM SIGMOD Record* 26.2 (1997), pp. 301–312.

[20] Jock Mackinlay, Pat Hanrahan, and Chris Stolte. “Show Me: Automatic Presentation for Visual Analysis”. In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1137–1144.

[21] Michael Mandel, Arnab Nandi, and Lilong Jiang. “Gestural Query Specification”. In: *PVLDB* 7.4 (2013).

[22] James Manyika et al. *Big data: The next frontier for innovation, competition, and productivity*. Tech. rep. McKinsey Global institute, June 2011.

[23] Chris Olston et al. “VIQING: Visual interactive querying”. In: *IEEE Symposium on Visual Languages*. 1998, pp. 162–169.

[24] Michalis Petropoulos, Alin Deutsch, and Yannis Papakonstantinou. “Interactive query formulation over web service-accessed sources”. In: *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM. 2006, pp. 253–264.

[25] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. “Towards a Theory of Natural Language Interfaces to Databases”. In: *IUI*. New York, NY, USA: ACM, 2003, pp. 149–157.

[26] Vijayshankar Raman and Joseph M. Hellerstein. “Potter’s wheel: An interactive data cleaning system”. In: *VLDB*. Vol. 1. 2001, pp. 381–390.

[27] Burr Settles. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison, 2009.

[28] Chris Stolte, Diane Tang, and Pat Hanrahan. “Polaris: A system for query, analysis, and visualization of multidimensional relational databases”. In: *IEEE Transactions on Visualization and Computer Graphics* 8.1 (2002), pp. 52–65.

[29] Michael Stonebraker et al. “Data Curation at Scale: The Data Tamer System.” In: *CIDR*. 2013.

[30] Trifacta Inc. *Data Transformation Platform Customers*. `http://www.trifacta.com/customers/`. Accessed November 17, 2014. 2014.

[31] Kuat Yessenov et al. “A colorful approach to text processing by example”. In: *UIST*. 2013, pp. 495–504.

[32] Moshe M. Zloof. “Query-by-example: A data base language”. In: *IBM systems Journal* 16.4 (1977), pp. 324–343.