THE UNIVERSITY *of* EDINBURGH

Edinburgh Research Explorer

The VADA Architecture for Cost-Effective Data Wrangling

OPEN ACCESS

The VADA Architecture for Cost-Effective Data Wrangling

Nikolaos Konstantinou[1], Martin Koehler[1], Edward Abel[1], Cristina Civili[2],
Bernd Neumayr[3], Emanuel Sallinger[3], Alvaro A.A. Fernandes[1], Georg Gottlob[3],
John A. Keane[1], Leonid Libkin[2], Norman W. Paton[1]

School of Computer Science[1]	School of Informatics[2]	Dept. of Computer Science[3]
University of Manchester	University of Edinburgh	University of Oxford
Manchester, M13 9PL, UK	Edinburgh, EH8 9AB, UK	Oxford, OX1 3QD, UK

ABSTRACT

Data wrangling, the multi-faceted process by which the data
required by an application is identified, extracted, cleaned
and integrated, is often cumbersome and labor intensive.
In this paper, we present an architecture that supports a
complete data wrangling lifecycle, orchestrates components
dynamically, builds on automation wherever possible, is in-
formed by whatever data is available, refines automatically
produced results in the light of feedback, takes into account
the user's priorities, and supports data scientists with di-
verse skill sets. The architecture is demonstrated in practice
for wrangling property sales and open government data.

1. INTRODUCTION

Although there are ever more potentially valuable data
sets available, as a result of both technical developments
such as web data extraction and policy developments such
as open government data, putting that data to effective use
is often a cumbersome and labor intensive process. Indeed,
it is quoted that data scientists may spend up to 80% of their
time on the process of extracting, collating and cleaning data
that is a precursor to its use for analysis[1]. If this process,
referred to as *data wrangling*, could be made more systematic
and cost effective, this would free up data scientists to devote
more of their efforts to analysing the data and interpreting
the results [7].

Although there is now tool support for aspects of the data
wrangling process, such as data format transformations [9],
in this paper we make a proposal for a data wrangling archi-
tecture that: (i) combines a comprehensive (and extensible)
collection of wrangling components that between them cover
the complete data wrangling lifecycle; (ii) builds on automa-
tion wherever possible, making use of whatever information
is available about the domain of application; (iii) refines the
results of automated processes in the light of user feedback;
and (iv) takes into account the user's priorities within the

---

[1]New York Times://http://nyti.ms/1Aqif2X

wrangling process, for example where it is necessary to trade
off some aspects of data quality with others.

In this paper we demonstrate an approach to data wran-
gling that has comparable scope to typical Extract-Transform-
Load (ETL) systems [12], but that through automation,
feedback and dynamic orchestration seeks to reduce the ex-
tent to which skilled application developers are required to
configure individual components and to specify the depen-
dencies between them. Our aim is to reduce the effort re-
quired for data wrangling, while accommodating diverse user
communities, in terms of the levels of expertise available for
data wrangling and the features that are required of the
resulting data sets.

2. ARCHITECTURE

The VADA (Value Added DAta systems) architecture is
illustrated in Figure 1. Here we briefly summarize the key
components and their relationships, before drilling down to
provide some more details of how they are applied in our
demonstration scenario.

The *Transducers* are components that represent the func-
tionalities within the wrangling process; a *transducer*[2] is a
software component with input and output dependencies de-
fined as Datalog queries over the knowledge base and/or the
state of the transducer. The input dependencies, for exam-
ple, may initiate the evaluation of a transducer when infor-
mation becomes available on which it can act. For exam-
ple, a *mapping generation* transducer may start to evaluate
when matches have been created between source and target
schemas, or a *data fusion* transducer may start to evaluate
when duplicates have been detected. The architecture is not
tied to a specific or fixed set of transducers.

The *Knowledge Base* is a repository for representing the
data of relevance to the data wrangling process; this includes
information about the requirements of the user (*user con-
text*), the application domain (*data context*), and metadata
created and used by the transducers that participate in the
wrangling process. The knowledge base also provides access
to extensional data, but for the most part this is actually
stored in external file systems or databases.

The *Vadalog Reasoner* supports reasoning over the knowl-
edge base using Vadalog, a member of the Datalog± family
of languages [2]; Vadalog plays several roles in the architec-
ture, including specifying transducer dependencies, coordi-

---

[2]The notion of transducer is inspired by earlier work on
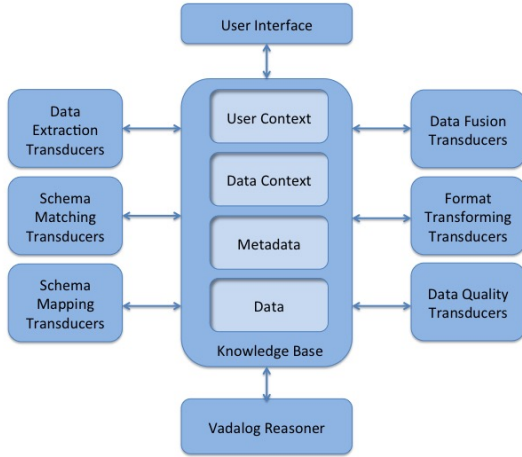*relational transducers* [1], although the languages used are
not formally equivalent.

Figure 1: VADA Architecture.

Rightmove					
price	street	postcode	bedrooms	type	description

Onthemarket					
price	street	postcode	bedrooms	type	description

Deprivation	
postcode	crime

(a) Sources.

Target						
type	description	street	postcode	bedrooms	price	crimerank

(b) Target Schema.

Address		
street	city	postcode

(c) Data Context.

```
completeness crimerank very strongly more important than
        accuracy property.type
consistency property strongly more important than
        completeness property.bedrooms
completeness property.street moderately more important than
        completeness property.postcode
```

(d) User Context.

Figure 2: Demonstration Scenario.

nating the orchestration of the transducers, and representing schema mappings.

Although a wide range of types of interaction with the architecture are possible, for the demonstration, the *User Interface* is being used by a data scientist who provides the information specifying the user context and the data context, as well as providing feedback on candidate results.

2.1 Real Estate Scenario

The demonstration will act on real estate data sets, bringing together: (i) data about properties that are for sale from web data extraction; and (ii) open government data that provides information about the areas in which the properties are located. The demonstration will use representative real-world data sets; for the paper we use a subset of these as a running example. In Figure 2(a), the source tables *Rightmove* and *Onthemarket* represent the results of web data extraction over deep web sources, as can be generated automatically by DIADEM [6]. The source table *Deprivation* is from open government data. For ease of comprehension, in Figure 2 attribute names are consistent across the different tables, whereas in practice attribute correspondences may need to be derived by schema matchers.

2.2 User and Data Context

We assume that the user is familiar with the domain of application (e.g., working as a data scientist to analyse property market behavior), and thus has an understanding of the priorities for the data wrangling activity. As such, we assume that the user can provide a target schema that describes the data that they need the data wrangling process to produce, such as that illustrated in Figure 2(b).

It is often the case, however, that additional information is available about a domain that can inform the data wrangling process; we refer to such information as the *data context*. For our example scenario, as illustrated in Figure 2(c), there are publicly available data sets that provide access to lists of addresses. Thus the user is able to associate the target schema with such data, which may be, for example, *reference data* (e.g., the complete list of postcodes or addresses), *master data* (e.g., the complete list of properties the user is interested in), or simply *example data* (e.g., a list of properties
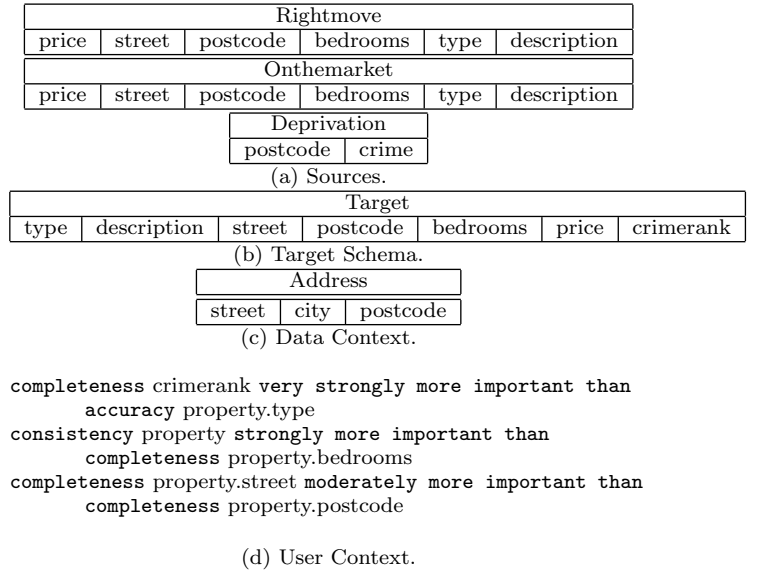
that a user has to hand, such as the properties that an estate agent has previously sold). The nature and coverage of such information will vary from user to user and application to application, but where it is available, we will demonstrate how it can be used to inform the wrangling process.

Furthermore, the user may be able to make explicit features of the resulting data set that are more or less important; there are likely to be trade-offs in any wrangling activity, such as between quality metrics. For example, if the user is going to analyse the relationship between property prices and crime levels, then the *completeness* of the *crimerank* attribute is likely to be more important to the user than the *accuracy* of the *type* attribute, as illustrated in Figure 2(d). We refer to such preference information as the *user context*, and will demonstrate the use of such a pairwise comparison approach, which has been shown to be effective in a range of multi-criteria decision analysis methodologies. In this approach, the user can make several pairwise comparison statements, for example based on quality metrics, that make explicit their requirements, and these are taken into account during data wrangling, to inform activities such as source or mapping selection. We note that different uses of the same data set may give rise to different user contexts; for example, if the user is now interested in analysing the relationship between property size and the *crimerank* attribute, then the *completeness* of the number of bedrooms becomes more important than in the user context definition from Figure 2(d).

2.3 Individual Transducers

The architecture is extensible, in that additional transducers can be added at any time; transducers can be implemented in Vadalog, or by wrapping external systems. There are typically several transducers associated with each functionality that makes up the wrangling process (data extraction, schema mapping, etc).

The different transducers associated with a functionality typically have different input dependencies, as illustrated for

Activity	Transducer	Input Dependencies
Matching	Schema Matching	Src/Target Schemas
Matching	Instance Matching	Src/Target Instances
Mapping	Mapping Generation	Src/Target Schemas
Mapping	Mapping Selection	Quality Metrics
Quality	CFD Learning	Data Examples

Table 1: Example transducer input dependencies

some transducers in Table 1. For example, in schema matching, an *Instance Matching* transducer requires instances to be available from both the schemas to be matched, and in data quality a *Conditional Functional Dependency (CFD) Learning* transducer requires that the data context for the target schema includes instances (e.g., from master or reference data)[4]. Each transducer knows what data it needs, and becomes available for execution when that data is available in the knowledge base; these dependencies define the data flow between transducers.

The declarative specification of dependencies is relevant to both the user and data contexts. For example, in the *user context* examples in Figure 2(d), although the `completeness` of the *crimerank* attribute can be estimated as the fraction of non-null values, determining the `consistency` of the *property* table needs additional information. If the user provides a reference data set of valid addresses from an open government data source as part of the data context, then this can be used to learn CFDs, against which the consistency of the address information within the property table can be established. Given the new information in the data context, a *Quality Metric* transducer becomes able to run, adding quality metrics on sources and mappings to the knowledge base. This in turn allows a *source selection* or a *mapping selection* transducer to run that selects sources or mappings, taking into account the user context.

It is a similar story for feedback. For example, assume that the user has been presented data in the target schema in Figure 2(b), in which for some tuples the number of *bedrooms* is clearly not correct (e.g., automatic web data extraction may be using the area of the master bedroom as the number of bedrooms). The user can annotate these values as incorrect through the *user interface*, leading to the feedback being asserted into the *knowledge base*. A *mapping evaluation* transducer, given information about the results of the mapping may identify a problem with a specific match used within the mapping, and revise the score of that match in the knowledge base. This may in run lead to the rerunning of the *mapping generation* transducer in the light of the new evidence, and thus to revised results for the user.

2.4 Controlling Transducers

As a consequence of the declarative approach to data dependencies, there may be several transducers available for execution at the same time; it is the responsibility of a *network transducer* to select between the executable transducers. A *network transducer* supplements the data dependencies with the additional decision making that determines the order in which transducers are executed. Network transducers are written by transducer developers or system administrators, and may be quite generic (e.g., by choosing transducers for one type of functionality before another, such as data extraction before mapping, and then using a pri-

ority scheme to make more local decisions) or may be quite specific (e.g., prefer instance level matchers to schema level matchers). In the demonstration, we will use a generic network transducer, and will show the dynamic execution of transducers as the user changes the data and user contexts.
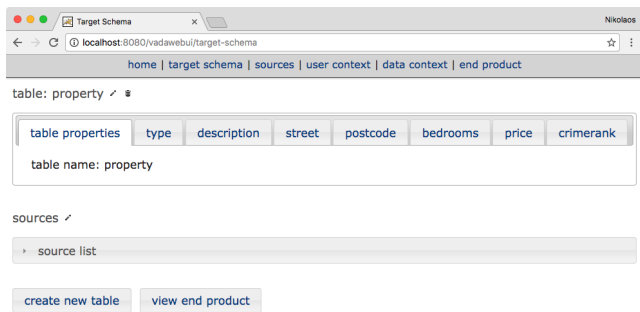
3. DEMONSTRATION

The functionality of VADA will be demonstrated through hands-on experience with a real estate scenario. Users interact with the system via a web interface; our goals are to demonstrate: (i) a pay-as-you-go approach to data wrangling, in which the more information is provided by the user, the better the outcome; (ii) the impact of different types of *data context*, *user context* and *feedback* on the result; and (iii) dynamic orchestration of components represented as transducers. Although the demonstration will support different ways of using the system, one approach would involve the following steps:

1. *Automatic Bootstrapping:* Interactively identify a collection of sources and define a target schema, as illustrated in Figure 3(a), and allow the system to automatically orchestrate a collection of transducers that together generate an initial result data set. The outcome can be expected to be of problematic quality.

2. *Data context:* Associate the target schema with some reference data, such as address information, as illustrated in Figure 3(b). The presence of such data allows various of the steps from bootstrapping to be revisited, including matching (to include the use of the instance data) and mapping generation (to use the revised matches). Furthermore, it is now also possible to learn CFDs from the reference data, and thereby to carry out repairs to the mapping results. The result data should now be of better quality, but is likely to contain some errors.

3. *Feedback:* The user views the result of the wrangling process from steps (1) and (2) as illustrated in Figure 3(c), and provides feedback to indicate that some of the results are correct or incorrect – such feedback can be at the tuple level or the attribute level. Depending on the feedback provided, this will enable some of the previous steps in the wrangling process to be revisited, giving rise to a revised result.

4. *User context:* The result is now hopefully of reasonable quality, but the data included in the result may not exhibit the features that make it especially well suited to the task at hand. The user specifies the *user context* by indicating the relative (pairwise) importance of different features in the result, as illustrated in Figure 3(d). The pairwise comparisons are used to derive weights that inform the selection of mappings based on multi-dimensional optimization.
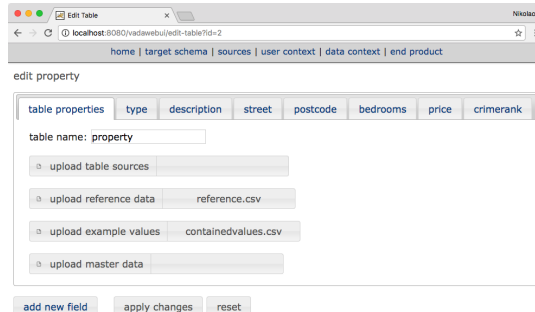
For all of steps (1) to (4), the system will provide browsable trace information that shows what transducers are being orchestrated, their inputs and results.

4. CONCLUSIONS

The importance of data wrangling for data analytics is motivating research into techniques and platforms that can
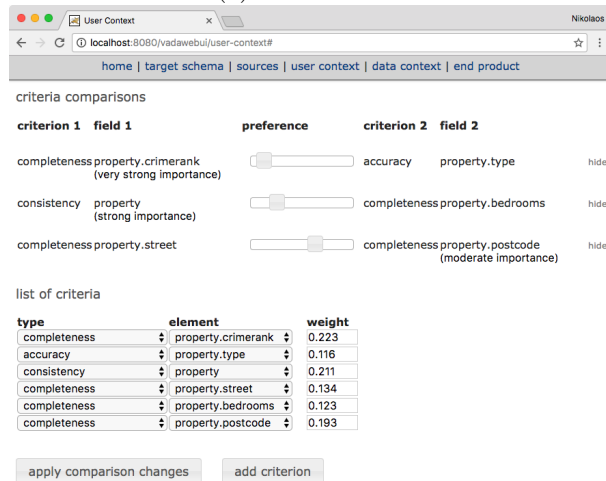
(a) Target schema.

(b) Data context.

(c) Results.

(d) User context.

Figure 3: Web interface for configuring data wrangling task.

support the wrangling process. Prominent results to date have supported the sharing of curation efforts across analytical tasks by way of data lakes (e.g., [8, 11]), built on linkage graphs to identify sources for integration [3], or sought to provide support for specific steps within the wrangling process (e.g., [5, 9, 10]). In this paper we complement such existing work by demonstrating an end-to-end data wrangling architecture that dynamically orchestrates new or existing components, automatically taking into account whatever information may be available to inform the wrangling process, in the form of the *data context*, *user context* or *user feedback*. By so doing, the effort directed at the wrangling process yields immediate results, in a pay-as-you-go manner, and data scientists can engage with the system in different ways. In the demonstration, the end-user tunes the wrangling process without writing code, but the architecture is flexible; developers can contribute to data wrangling by adding in new components as transducers, influencing the orchestration using control transducers, or by writing mappings or quality rules. As such the architecture provides mechanisms that support its deployment in both diverse application scenarios and development environments.

5. REFERENCES

[1] S. Abiteboul, V. Vianu, B. S. Fordham, and Y. Yesha. Relational transducers for electronic commerce. *J. Comput. Syst. Sci.*, 61(2):236–269, 2000.
[2] A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.
[3] D. Deng et al. The data civilizer system. In *CIDR*, 2017.
[4] W. Fan and F. Geerts. *Foundations of Data Quality Management*. Morgan & Claypool, 2012.
[5] M. H. Farid, A. Roatis, I. F. Ilyas, H.-F. Hoffmann, and X. Chu. CLAMS: bringing quality to data lakes. In *SIGMOD*, pages 2089–2092, 2016.
[6] T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, and C. Schallhart. The ontological key: automatically understanding and integrating forms to access the deep web. *VLDBJ*, 22(5):615–640, 2013.
[7] T. Furche, G. Gottlob, L. Libkin, G. Orsi, and N. W. Paton. Data wrangling for big data: Challenges and opportunities. In *EDBT*, pages 473–478, 2016.
[8] R. Hai, S. Geisler, and C. Quix. Constance: An intelligent data lake system. In *SIGMOD*, pages 2097–2100, 2016.
[9] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *CHI*, pages 3363–3372, 2011.
[10] J. Morcos, Z. Abedjan, I. Francis Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker. Dataxformer: An interactive data transformation tool. In *SIGMOD*, pages 883–888, 2015.
[11] I. Terrizzano, P. M. Schwarz, M. Roth, and J. E. Colino. Data wrangling: The challenging journey from the wild to the lake. In *CIDR*, 2015.
[12] P. Vassiliadis. A survey of extract-transform-load technology. *IJDWM*, 5(3):1–27, 2011.