Pay-as-you-go Data Cleaning and Integration

Shawn Jeffery

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-169
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-169.html

December 18, 2008

Copyright 2008, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Pay-as-you-go Data Cleaning and Integration

by

Shawn R. Jeffery

B.S. (University of Wisconsin, Madison) 2002
M.S. (University of California, Berkeley) 2005

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Michael J. Franklin, Chair
Professor Joseph M. Hellerstein
Professor Dara O'Rourke

Fall 2008

The dissertation of Shawn R. Jeffery is approved.

Chair	Date
	Date
	Date

University of California, Berkeley

Fall 2008

Pay-as-you-go Data Cleaning and Integration

Copyright © 2008

by

Shawn R. Jeffery

Abstract

Pay-as-you-go Data Cleaning and Integration

by

Shawn R. Jeffery

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Michael J. Franklin, Chair

Many emerging applications such as Web mash-ups and large-scale sensor deployments seek to make use of large collections of heterogeneous data sources to enable powerful new services. These sources range from traditional sources such as relational databases to emerging sources such as structured data on the Web and streaming sensor data.

In order to realize the potential of these applications, however, the data from these disparate sources must be cleaned and integrated. In emerging data sources such as the Web and sensors, traditional cleaning and integration techniques are necessary, but not sufficient to deal with the unique challenges presented by this data. I argue that new techniques, based on the concept of *pay-as-you-go* are crucial for incorporating such data sources into applications. This concept provides a framework for building cleaning and integration solutions that are easy to deploy and maintain, efficiently leverage human feedback where possible, and automatically adapt their processing to the underlying data.

In this thesis, I contribute key building blocks designed to provide pay-as-you-

go data cleaning and integration. Specifically, I develop the following techniques: *Roomba*, a technique for effectively involving user feedback to augment data cleaning mechanisms; *Metaphysical Data Independence* (MDI), a means of hiding all details of sensor data cleaning and integration under a single interface; *SMURF* an adaptive cleaning tool for providing MDI for RFID data; and *ESP*, a declarative-query based cleaning framework for sensor data streams. These techniques all embody key principles that underly the pay-as-you-go philosophy: ease of setup and deployment, adaptability, and incremental integration.

Additionally, I show that a focus on the pay-as-you-go philosophy does not preclude effective data cleaning and integration mechanisms. Indeed, in many cases the techniques developed in this thesis are capable of producing higher-quality data than current cleaning and integration techniques. For instance, effective use of human feedback is able to integrate data in a large-scale data integration scenario with half the human cost of current approaches. Similarly, an adaptive approach to cleaning RFID data is able to produce a three-fold reduction in data error rate in certain scenarios compared to the state-of-the-art RFID middleware solutions.

In summary, this thesis makes two broad contributions. First, it demonstrates that a pay-as-you-go approach to data cleaning and integration enables an emerging class of applications dependent on data derived from many heterogeneous data sources. Second, it proposes a suite of pay-as-you-go based data cleaning and integration techniques that provide a solid foundation on which to build the systems to support these applications.

Professor Michael J. Franklin
Dissertation Committee Chair

Acknowledgements

While his official title is "research advisor", Michael Franklin has been many things throughout my years in graduate school: mentor, friend, guide, spokesman, and editor. From Mike, I have learned the importance of looking at the big picture, finding the important real-world problems to solve, and always asking "What's the point?" He has shown me that even when focusing on the big picture, the little stuff matters (e.g., correct punctuation for "e.g.,"). Mike gave me direction to focus my energy, but also knew when to let me find my own way. I owe much of my success to Mike's guidance and support.

Many years ago, when I was a somewhat directionless undergraduate, David DeWitt told me that I was going to get a Ph.D. at Berkeley. I am deeply grateful to him for that initial push, as well as for the advice, healthy encouragement, and swimming motivation he has given me over the years.

The database group at Berkeley has been a continual source of inspiration and friendship. Joe Hellerstein is a great complement to Mike with a great eye for details that has always kept me honest (especially with statistics). I am honored to have worked with many students in various research projects, especially Sailesh Krishnamurthy, Shariq Rizvi, Ryan Huebsch, Fred Reiss, and Eugene Wu (The Wu). The entire group, including Alexandra Meliou, Russell Sears, David Liu, David Chu, Tyson Condie, Boon Tau Loo, Mehul Shah, Amol Desphande, Daisy Wang, and Eirinaios Michelakis have provided many helpful comments, friendship, spirited debate, and of course, plenty of fun. While never seeming so at the time, the long stressful nights prior to demo, paper, or project deadlines were some of the best times in my years in Berkeley.

I have been fortunate to collaborate with many exceptional researchers during my graduate studies. I want to thank Gustavo Alonso, Jennifer Widom, Wei Hong,

Minos Garofalakis, and Alon Halevy for their guidance, insights, and willingness to put up with the various team names I used during our collaboration (e.g., *Smurfettes*, *Hifighters*, and *Roombers*). I am very happy to consider them both colleagues and friends.

I want to express my appreciation to Dara O'Rourke, Ryan Aipperspach, Graham Bullock, and everyone at Taoit for being a small group of dedicate people out to change the world. I am lucky to have had such a direct and empowering outlet for my research.

Julia Owen has been supportive and encouraging throughout my writing, patient through all the revisions, and helpful as a sounding board or just telling me to take a deep breath. Her unwavering belief in me and what I can accomplish has given me the strength I needed to finish this work.

I am indebted to my family for all of their encouragement, the value they placed on education, and their patience with me when I brought my laptop home for holidays. They have been infinitely supportive, but I also knew they'd be proud of me no matter where I ended up as long as I was happy.

Similar to most graduate students, my graduate career was filled with many ups and downs. The one constant throughout these times has been Fuego. I want to express my deepest gratitude to Johan, Evan, Steven, and all the Fuegans for being some of the most caring, supportive people I have ever met. From the first week I was in Berkeley, Fuego was there as both an escape from the strain of work as well as a grounding in the reality of friendship. No matter how foggy (literally or metaphorically) it was in Berkeley, it was always sunny in Spieker pool. I dedicate this dissertation to all the Fuegans who have been such a positive influence in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The quantity and complexity of structured data is rapidly increasing in a variety of new environments: large-scale sensor deployments [TPS+05] provide unprecedented insight into the physical world, an increasingly structured Web [MHC+06] is promising to make the bulk of human knowledge accessible in one large networked repository, large enterprises are becoming more reliant on data-driven IT infrastructures, and large-scale scientific collaborations (e.g., the LHC project [LHC08]) are linking vast amounts of data. As these data sources continue to grow in both scale and complexity, new applications are leveraging these disparate data sources to provide new services, such as Web mash-ups [HM08], cross-enterprise SOA systems [SF08], and world-wide sensing infrastructures [KNLZ07].

These emerging applications and environments, however, pose many new data management challenges. In particular, data in these environments are dirty, inconsistent, rapidly changing, and highly heterogeneous. Thus, before data can be used by applications, they must be *cleaned* and *integrated*. Data cleaning refers to the process of resolving errors such as missing or duplicate data, while data integration is the means by which multiple autonomous heterogeneous data sources are merged

into a single coherent picture of the underlying data. These processes are interrelated as many errors that need cleaning arise due to integrating heterogeneous sources. Traditional data cleaning and integration techniques are necessary but not sufficient to deal with these new sources of data. Therefore, there is a need for techniques that are capable of cleaning and integrating data from these data sources.

This thesis sets out to address this challenge. The key principle used in my research for enabling data cleaning and integration techniques for these new data sources is the concept of *pay-as-you-go* [FHM05]. When applied to data cleaning and integration, this principle states that these processes are on-going, and as such, techniques must be designed to handle continuous, incremental cleaning. More specifically, pay-as-you-go techniques should have three important properties. First, pay-as-you-go techniques should make judicious use of human effort (i.e., the "pay" component of pay-as-you-go) to incrementally improve data quality over time (i.e., the "as-you-go" component). Second, these techniques should automatically adapt to the characteristics of the underlying data sources when the use of human effort is infeasible. Finally, there should be little up-front investment for pay-as-you-go solutions: such techniques should be easy to initially deploy.

In this thesis, I develop a set of pay-as-you-go techniques to integrate and clean data from a wide range of sources. Due to the interrelated nature of data cleaning and integration, I address both processes with pay-as-you-go techniques. These techniques embody different aspects of the pay-as-you-go principle to directly address the unique challenges presented by emerging data sources.

In this initial chapter, I first ground my research by describing a motivating application that utilizes data from multiple heterogeneous data sources, many of which are ill-handled by current data integration technology. I then discuss the shortcomings of current data management systems in providing data for this type of application.

Next, I describe the principle of pay-as-you-go integration and cleaning. Finally, I outline the pay-as-you-go techniques for next-generation data integration platforms developed as part of this thesis.

1.1 Total Consumer Awareness

While there are many emerging applications that utilize data derived from a wide variety of heterogeneous data sources, one particularly compelling example is *Total Consumer Awareness*. Rather than using only price, brand, and sometimes quality information to decide what products to buy, in a TCA application consumers are given the complete spectrum of information about a particular product when making a purchasing decision: e.g., the product's origin and transportation mechanism, the actual chemical significance of the each ingredient, comparable prices for the product in nearby stores or online, and recommendations from friends and family. Given this data at the point of purchase, accessed either through the Web or mobile devices, consumers can make much more informed decisions about the products they buy.

A concrete example of TCA is illustrated in Figure 1.1. In this figure, a user at a store is interested in information about a particular product. He enters or scans identifying information for a product (e.g., UPC [GS108], EPC ID [EPC05b], or product name) into a mobile device (e.g., [iPh08, NFC08]). Using this information, the device connects with the TCA services to access a wealth of information about that product, including health and environmental impacts, friends' recommendations, and prices at nearby stores. This information is then tailored to the user's preferences and presented in an easy to understand manner. To produce this information, TCA integrates data from a range of data sources falling into two broad categories: Web data and sensor data.

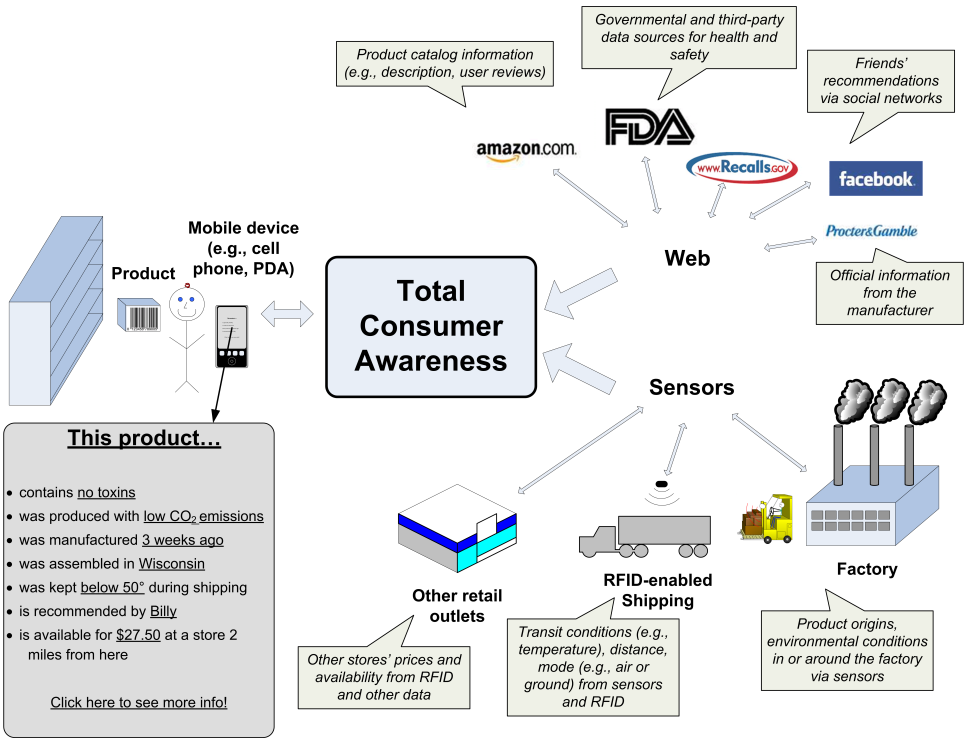


Figure 1.1. An example of a Total Consumer Awareness (TCA) application.

Web Data

There is a large amount of product information residing on the Web, ranging from traditional HTML pages to streaming RSS feeds. The TCA application extracts general product information from the manufacturer's website using information extraction techniques (e.g., [CDYR08]). This data is correlated with product catalog information and user reviews using sites such as Amazon [AZN08] or Shopping.com [Sho08]. Such sites typically offer APIs that allow access to semi-structured data feeds (e.g., RSS).

Deeper product information such as product recalls or the product's health risks can be extracted from both governmental and non-governmental sources such as the FDA [FDA08], recalls.gov [Rec08], or WebMD [WMD08]. These sites can either be

crawled and integrated using information extraction techniques as noted above, or can be accessed using form-based interfaces. For instance, the U.S. Consumer Product Safety Commission provides a form for accessing its database of recalled consumer products [CPS08]. Such data sources, where databases are accessible behind Web forms, are referred to as the *deep web* [Ber01].

Finally, product reviews and recommendations from friends and family can be gleaned from online social networks. For instance, the social-networking site Facebook allows users to recommend products to their friends [FBA08]. Through APIs exposed by Facebook, the TCA application can access and flow this data to its users.

Sensor Data

The other broad class of information utilized in this application is data produced by sensors: devices placed in the physical world that monitor and report data about some attribute of reality.

Radio Frequency Identification (RFID) provides fine-grained insights into a product's life-cycle. Through data derived from RFID tagging, the TCA application can determine, for example, the factory in which the product was manufactured. By combining this information with data collected from sensors monitoring environmental conditions in or near the factory (e.g., air and water quality), the application can derive an estimated environmental impact of producing that product.

RFID can provide additional detail regarding the transport of the product through its supply chain. RFID readers installed at critical points in the supply chain such as shipping docks and distribution centers provide shipping route and timing information. This data can be used to determine the freshness of the product as well additional environmental impact based on distance traveled (e.g., *food miles* [Pax94]). Sensors placed on or near the product during shipping can be used to determine the transport conditions, such as temperature and acceleration. For instance, an

acceleration-sensing device (e.g., a WISP [SSP+06]) on a product could report that the item was handled carefully during distribution.

With RFID-enabled distribution centers and retail stores (e.g., smart shelves [DKB03]), TCA can access the real-time availability of a product in its entire distribution network. Correlating these data with the user's location information (through GPS or cell phone positioning such as [LCC+05]) and product pricing information available online (e.g., [TF08]) allows TCA to alert the user of equivalent products at nearby stores.

Such an application has potential to change the way consumers interact with products. By collecting, processing, cleaning, and correlating this information and presenting it to consumers at the time of purchase, TCA empowers consumers to be significantly more informed about the products they buy. Thus, TCA has the power to resolve many of the information asymmetries present in today's markets [Sti61, Sti02]; consumers can choose products based on more complete information, not just price, brand, and limited quality information. Thus, manufacturers will be driven to produce products more in line with what consumers care about, hopefully promoting higher quality, healthier, and more sustainable products and practices.

To realize this vision of TCA, however, there are significant data management challenges to overcome. TCA accesses a wide array of heterogeneous data sources, each with its own semantics and format. These sources contain data that are missing, inconsistent, and ambiguous. Additionally, the data is rapidly changing and dynamic. As I explain next, current data integration and cleaning techniques are ill-suited to deal with such challenges at the scale required by TCA.

1.2 Current Challenges

Traditional data management systems are based on several assumptions that prevent them from handling data in environments such as TCA.

First of all, most data management systems assume that the data in the database is completely valid: that is, they ignore any errors, inconsistencies, or other problems with the data. When dealing with data on the Web entered by humans or with sensor data produced by faulty devices, however, dirty data is not just an exceptional condition, it is the norm.

Traditional systems also assume that the data in the database is complete: if a datum is not in the database, it does not exist. Again, with fallible humans or sensors producing data, data will be missing, and the data management system must account for this missing data.

Another shortcoming of current data management solutions is that they assume that data unambiguously maps to real-world entities and that there are no duplicate entities in the system. When dealing with domains outside those traditionally handled by relational databases (e.g., back-office processing), semantic mappings between data are hard to define and duplicate entities are prevalent.

Currently, database systems assume that time and space are simply attributes just like any other attribute in the system. Time and space, however, have special significance in reality, and can be utilized to help address shortcomings of the data (e.g., the techniques presented in Chapter 6).

Finally, current systems assume that any data that does not fit with these expectations can be cleaned and integrated once, after which they will adhere to all of the above assumptions. That is, traditional data integration typically works in a two-phased approach: during the first phase the data is cleaned and/or the appropriate

cleaning pipelines are created, a time-intensive task that can require extensive human involvement. Once this phase is complete, the system begins its operational phase, where it is assumed that the data is valid, complete, and unambiguous.

When dealing with large amounts of continuously changing data in the real world, cleaning and integration are never complete. As such, these processes must continually adapt to the data. Furthermore, applications that utilize large collections of heterogeneous data (e.g., TCA) typically do not need perfectly clean and integrated data (which is likely impossible), but rather "close enough" cleaning and integration; instead, the primary needs of such applications typically focus on ease of initial setup, evolvability and adaptability, and minimizing human effort.

1.3 The Pay-As-You-Go Principle

Traditional cleaning and integration mechanisms are necessary to deal with the challenges described above, but are not sufficient given the nature of these large-scale heterogeneous data sources. As described in the preceding sections, it is typically impossible to tightly integrate all relevant data sources: the data is of a scale and heterogeneity such that full, one-time integration is not feasible and if it were, the continually changing nature of the data precludes such an approach.

In this thesis, I address these issues using an incremental, *pay-as-you-go* approach as the basis for designing data cleaning and integration tools. These tools are easy to deploy and provide basic cleaning and integration out of the box. Over time, through the combination of semi-automated techniques and judicious use of human effort, pay-as-you-go tools gradually improve the quality of the underlying data. Where the use of human effort is infeasible, such as with streaming sensor data, pay-as-you-go tools automatically adapt to the characteristics of the underlying data and environment.

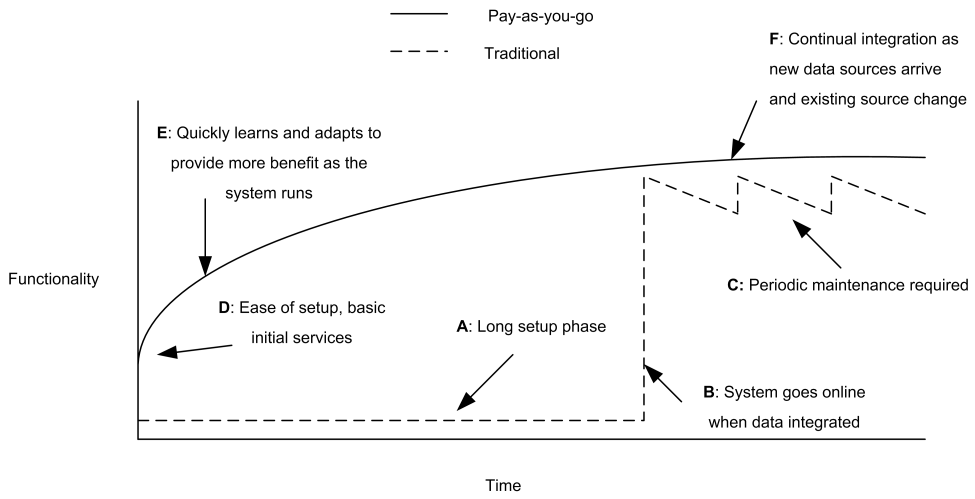


Figure 1.2. Pay-as-you-go integration compared to traditional integration.

To contrast the pay-as-you-go approach with that of traditional data cleaning and integration, Figure 1.2 compares the functionality over time provided by a traditional data management system compared to a system using tools based on the pay-as-you-go principle. Traditional data integration, shown as a dashed line, requires a long setup phase during which structure is imposed on unstructured data, schemas are matched, and the data are cleaned (phase **A**). The length of this phase is determined by the amount of data to be integrated and its heterogeneity. Only once this process is complete does the system go live and begin providing functionality (**B**). Since data in these new application environments are always changing, periodic upkeep is necessary in order to structure, match, and clean the data (**C**). There are two major drawbacks to this approach. First, the data are inaccessible for the entire setup phase. As the amount and complexity of the data grow, so does the period of time during which the data are inaccessible. Second, the maintenance cost in this approach is high because the data must be periodically re-integrated as the data fluctuate.

A system utilizing pay-as-you-go cleaning and integration tools, on the other hand, provides immediate benefit, as such tools are designed to be easy to set-up and deploy, and provide basic services such as keyword search (**D**). Pay-as-you-go tools are

designed to progressively process the data by adding structure, matching schemas and objects, and cleaning the data incrementally and iteratively over time (**E**). Thus, the data in such a system are continually improving. Finally, during normal operation, pay-as-you-go-based tools continue their ongoing integration to accommodate the continual fluctuations of the data (**F**).

Pay-as-you-go techniques are crucial for enabling data cleaning and integration to cope with challenges presented by large collections of heterogeneous data. In these environments, there are many data sources with complex interactions that are rapidly changing, and thus any data cleaning or integration solution must be easy to deploy and flexible to maintain thereafter. Additionally, the data are of such scale and heterogeneity that they cannot be cleaned and integrated all at once; they must be integrated incrementally using pay-as-you-go techniques. Furthermore, in some cases, such as with streaming sensor data, there is no time for human intervention, and so the cleaning techniques must be designed to automatically adapt over time.

Dataspaces

Dataspaces [FHM05, HFM06, MHF06] provide a context for the pay-as-you-go techniques presented in this thesis. Dataspaces are a vision for next-generation data integration environments and systems that describes a set of principles for extending traditional technology to new data sources and application environments. These principles include encompassing large collections of heterogeneous data, unified access through both simple (e.g., keyword search) and complex (e.g., structured query) interfaces, and best-effort behavior.

For the purposes of this thesis, I define a *dataspace* as a collection of multiple heterogeneous data sources, both traditional and emerging, including sensor data, Web data, and traditional databases. A *dataspace system* (also referred to as a *dataspace*

support platform, or *DSSP*) is a software system designed to manage such data. The goal of a dataspace system is to incorporate data from these disparate data sources and to process, clean, merge, and integrate these data for use in applications via structured queries, visualizations, and data exploration. I term the class of applications that utilize dataspace data (e.g., TCA) as *dataspace applications*.

1.4 Pay-as-you-go Techniques

Having introduced the pay-as-you-go principle, its benefits, and the broader dataspace context, I now briefly outline the pay-as-you-go data cleaning and integration techniques I develop in this thesis.

Roomba: Pay-as-you-go Guidance of Human Input

The first technique I present is *Roomba* (Chapter 3), a means of efficiently utilizing human feedback to continually improve large-scale structured and semi-structured data repositories, such as Google Base [GB07] and similar sites found on the Web.

A primary challenge to large-scale data integration in dataspaces is creating semantic equivalences between elements from different data sources that correspond to the same real-world entity or concept. These correspondences are necessary to provide quality data to applications utilizing dataspace data. As part of the pay-as-you-go approach, dataspaces use automated mechanisms such as schema matching and entity resolution to provide initial correspondences, termed *candidate matches*. For instance: *"the schema elements 'school' and 'university' may be the same."* One practical way to resolve such a match is to ask a human to confirm or reject the match. Since the number of such questions that could benefit from human feedback typically far exceed the resources available, only a limited number of tasks can be presented to humans. Thus, the key challenge in involving humans in data integration tasks is to

determine an order in which to solicit user feedback for confirming candidate matches such that the most useful matches are presented to humans first.

To address this challenge, I develop a decision-theoretic framework for ordering candidate matches for user confirmation using the concept of the *value of perfect information* (*VPI*) [RN03]. At the core of this concept is a *utility function* that quantifies the desirability of a given state; thus, I devise a utility function for a dataspace based on query result quality. While an exact calculation of VPI is computationally infeasible, I show in practice how to efficiently apply an approximation of VPI in concert with this utility function to order user confirmations.

One of the key advantages of this method is that it considers candidate matches produced from *multiple* types of mechanisms (e.g., schema matching, entity resolution) in a uniform fashion. Hence, the system can weigh, for example, the benefit of asking to confirm a schema match versus that of confirming the identity of references to two objects in the domain.

A detailed experimental evaluation on both real and synthetic datasets shows that the ordering of user feedback produced by this VPI-based approach yields a dataspace with a significantly higher utility, and thus better query result quality, than a wide range of other ordering strategies.

To utilize this decision-theoretic framework in a dataspace, I develop Roomba, a system for guiding user feedback in a dataspace system.

Metaphysical Data Independence

Roomba provides pay-as-you-go support for data integration problems (e.g., schema matching, entity resolution) that arise when dealing with traditional relational data or data on the Web; many dataspace applications, however, depend on data collected from the physical world through sensor devices as well. Unfortunately,

there is a wide gulf between the raw data produced by sensors and the high-level needs of dataspace applications. I term this gulf the *physical-digital divide*. Specifically, sensor data is unreliable, semantically low-level, and rapidly changing. The first step in integrating sensor data into a dataspace application is to provide a means by which applications can interact with sensor data at the application's semantic level, without dealing with the challenges associated with the physical-digital divide.

To address this issue, in the next part of this thesis (Chapter 4) I present *Metaphysical Data Independence* (*MDI*), an interface for exposing sensor data to dataspace applications. MDI proposes a layer of independence that shields applications from the challenges that arise when dealing with the physical-digital divide. The key philosophy behind MDI is that applications should not deal with any aspect of the physical devices, but rather should interface with a high-level reconstruction of the physical world created by a dataspace system.

SMURF: Adaptive Cleaning of RFID Data Streams

To instantiate the MDI interface, a dataspace system must deal with the unreliable and rapidly changing nature of sensor data. While Roomba provides such services in environments where humans can be put in the loop, when dealing with sensor data that is streaming in real-time, these processes must be automated. To this end, I develop *SMURF* (*Statistical sMoothing of Unreliable RFid data*), a cleaning mechanism that alleviates issues associated with RFID data through adaptive techniques based on a novel statistical framework (Chapter 5). SMURF models the unreliability of RFID readings by viewing RFID streams as a statistical sample of tags in the physical world, and exploits techniques grounded in sampling theory to drive its cleaning processes. Through the use of statistical tools such as binomial sampling and π-estimators [SSW92], SMURF continuously adapts its cleaning techniques in a principled manner to provide accurate RFID data to applications.

Through a detailed experimental study in multiple scenarios, I show that SMURF's techniques can eliminate many of the errors associated with RFID data to effectively provide MDI to dataspace applications.

ESP: A Declarative Framework for Sensor Data Processing

MDI and SMURF provide an interface and mechanism for pay-as-you-go cleaning and integration of sensor data. To provide a framework for incorporating these techniques into a dataspace system, I develop *Extensible Sensor stream Processing* (*ESP*)(Chapter 6). ESP is modular framework for building data cleaning pipelines focused on streaming data produced by sensor devices. ESP is built using declarative query processing as found in relational database query languages to enable rapid deployment of cleaning solutions with little up-front cost. Over time, ESP can be incrementally upgraded due to its modular nature: individual modules can be enhanced through declarative means or by embedding more complex techniques such as those used in SMURF.

I demonstrate ESP's effectiveness and ease of use through three real-world scenarios. First, I show a data cleaning pipeline built for RFID data that successively refines dirty RFID data streams. Second, I show ESP-based pipelines built to correct for errors commonly found in wireless sensor network data streams. Finally, I demonstrate the use of ESP to build a pipeline for integrating multiple sensor streams (e.g., RFID, wireless sensors, motion detectors) to produce a "person detector" in a digital home scenario. In all of these scenarios, ESP-based pipelines produce clean data with minimal deployment overhead.

1.5 Contributions

To summarize, the contributions of this thesis are as follows:

- **Pay-as-you-go data cleaning and integration**: I identify the importance of applying the pay-as-you-go principle to data cleaning and integration to enable emerging applications such as Total Consumer Awareness.

- **User feedback to integrate dataspace data**: To enable dataspaces to clean and integrate data in a pay-as-you-go manner, I propose a model and a mechanism, *Roomba*, for efficiently soliciting user feedback in data integration tasks. Roomba enables a dataspace to incrementally integrate data by determining the most important tasks in which to involve humans (Chapter 3).

- **Identification of the physical-digital divide**: There exists a disconnect between the raw data produced by sensor devices and the data expected by high-level applications that I call the physical-digital divide. I outline the reasons for this divide and the challenges it presents for applications, illustrated through multiple real-world studies (Chapter 4).

- **Metaphysical Data Independence**: To shield dataspace applications from the challenges associated with the physical-digital divide, I propose *Metaphysical Data Independence* (MDI), a philosophy for building dataspace system interfaces that hides all details of the underlying devices (Chapter 4).

- **Adaptive cleaning for cleaning RFID data**: To clean RFID data in an adaptive, pay-as-you-go manner, I develop SMURF, an adaptive mechanism for providing MDI for RFID-based applications (Chapter 5).

- **A declarative framework for sensor data processing**: To enable easy to deploy sensor data cleaning infrastructures for providing MDI, I present *ESP*, a framework of declarative cleaning stages designed to clean sensor data streams. Additionally, I describe how ESP tracks data quality estimates through its pipeline (Chapter 6).

1.6 Summary

The next generation of data-intensive applications will be built upon heterogeneous data derived from a wide range of sources, including the Web and sensors. To enable these applications, next-generation data integration platforms such as dataspace systems must integrate data from a variety of data sources and provide a unified interface to this data. I argue that due to the large-scale, heterogeneous, and dynamic nature of these new application environments, data integration and cleaning techniques must be built around the concept of "pay-as-you-go". In this chapter, I have outlined a suite of such techniques that enable next-generation data integration platforms to handle data from sources such as large-scale heterogeneous repositories (e.g., the Web) and sensor deployments.

The remainder of this dissertation is organized as follows. Chapter 2 provides a background on relevant concepts. Chapter 3 discusses incremental cleaning involving user feedback. Chapter 4 presents metaphysical data independence, a philosophy for cleaning and integrating sensor data streams that hides all details of the underlying devices. SMURF, an adaptive mechanism for cleaning RFID data, is discussed in Chapter 5. In Chapter 6, I outline an general framework for cleaning sensor data streams to provide MDI using declarative query processing. Finally, Chapter 7 provides concluding remarks.

Chapter 2

Background

In this chapter, I present background material to place the ideas and techniques developed in this thesis in the context of prior research. I first discuss dataspace data with a focus on two of the emerging data sources that are gaining importance to dataspace applications: data on the Web and data produced by sensor technologies. I highlight the challenges associated with utilizing these data sources using traditional data processing techniques. Then, I outline current data integration technologies and how they attempt to address these challenges. Note that additional background and related work specific to each topic are covered in the relevant chapter.

2.1 Dataspace Data

As discussed in the previous chapter, dataspace data consist of a wide range of data sources, encompassing traditional sources as well as emerging data sources. Here I detail two of the more important types of emerging data sources, data on the Web and sensor data streams.

2.1.1 Data on the Web

While the Web has traditionally been dominated by unstructured content, there has recently been an increase in the Web's volume and diversity of data. This diversity of data can be roughly grouped into three categories, differing in their degree of organization: structured, semi-structured, and unstructured content. Structured data follows a strict format, the most common of which is a tabular layout as used in relational databases. Unstructured data, on the other hand, has very little organization; they are typically free-text. Semi-structured data [MAG+97] falls between structured and unstructured data: it has some organization, but the organization is not necessarily strictly followed. I discuss each class in detail below.

Structured Data

Historically, structured data repositories have been network-accessible through distributed and federated database systems (e.g., [LHM+84, Sto86, SAL+96]), but the Web has enabled increased access to structured data repositories through multiple mechanisms. Recently, *Deep Web* searching tools [Ber01] and Web APIs (Application Programming Interfaces), are opening up even more such repositories. Conceptually similar, both mechanisms provide simple Web-accessible interfaces to potentially complex structured-data backends.

The deep web refers to online content, typically stored in a relational database, accessible through HTML forms. On a deep web site, a form exposes some set of attributes on which to search. Upon form submission, these attributes are used to construct a query that is posed to the backend database. The site then renders the results of this query, usually in HTML or XML. For instance, the site Cars.com [Car08] exposes a large database of for-sale car listings via a simple form-based interface, as shown in Figure 2.1. This figure shows the form page where users can enter car

attributes and the result page returned by the site after submitting such a query. Recent estimates put the number of pages in the deep web in the tens of millions of forms [MJC+07], each of which may lead to hundreds of thousands of pages (e.g., [Car08]).

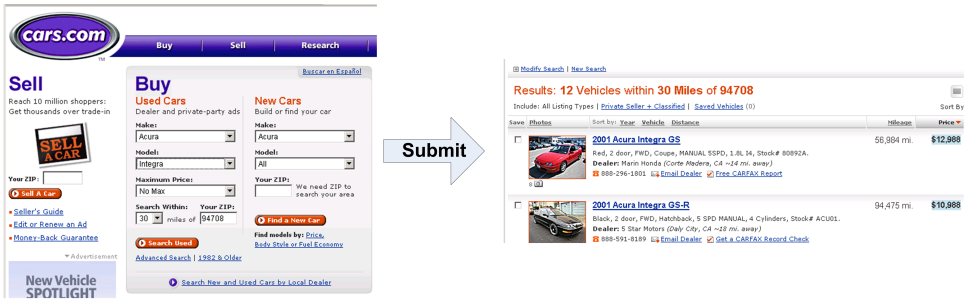


Figure 2.1. An example deep web site, Cars.com.

Web-accessible APIs are programming interfaces, usually exposed through standards-based mechanisms such as HTTP [FGM+99, Fie00] or SOAP [GHM+06], to applications that return data in a standard format such as XML. Web APIs have grown in popularity recently, primarily due to the explosion of *mash-ups*, websites that integrate data from two or more Web APIs to produce a new service. For instance, Housingmaps.com [HM08] combines map data from the Google Maps API [GMA08] and Craigslist.com [CL08] to display apartment listings on a map.

Semi-Structured Data

XML data [BPSM98] is arguably the most ubiquitous form of semi-structured data on the Web. It is generally rich in structure and often has a schema. There are many XML repositories available on the Web, including protein databases [SP08, PIR08] and bibliography data [DBL08].

Another form of semi-structured data on the Web can be found in Google Base [GB07]. Google Base in an online semi-structured repository of user-uploaded

content. The primary unit of data in Google Base is an *item*, that consists of one or more attribute-value pairs (e.g., ⟨"make"= "Toyota", "model" = "Prius"⟩). Items are organized into *item types*, which are essentially domains or verticals, such as vehicles or recipes. For each item type, Google Base provides a set of recommended attributes (i.e., schema) and popular values for each attribute. Users can, however, make up their own attributes and values for their items. Google Base also provides form-based interfaces and Web APIs for structured querying of this data. See Chapter 3 for an in-depth study involving Google Base.

The loosest form of semi-structured data on the Web is annotated or tagged data, that have emerged as a light-weight means of adding structure to Web data. In these schemes, key-words or terms (usually referred to as *tags*) are associated with some piece of information, such as a picture [Fli07, vAD04], Web page [GC07, Del07], or email message [GM08]. Tags provide a simple structure that enables classification, navigation, and search. One of the more popular examples of such a scheme is Flickr [Fli07], where users add descriptive nouns, verbs, or adjectives to pictures.

Unstructured Data

While much of data on the Web are unstructured, there are techniques for accessing and incorporating such data into structured data processing infrastructures such as a dataspace system.

Information extraction (IE) is a set of natural language processing techniques that convert unstructured text into structured data (see [CL96, CM03, AS06, DRV06] for recent overviews of such techniques). As a simple IE example, the text *"Chimpanzees like to eat figs and orangutans like to eat mangoes"* can be structured, for example, as the key-value pairs ⟨"animal"= "Chimpanzee", "eats" = "figs"⟩, ⟨"animal"= "Orangutan", "eats" = "mangoes"⟩. An example of a structured repository created

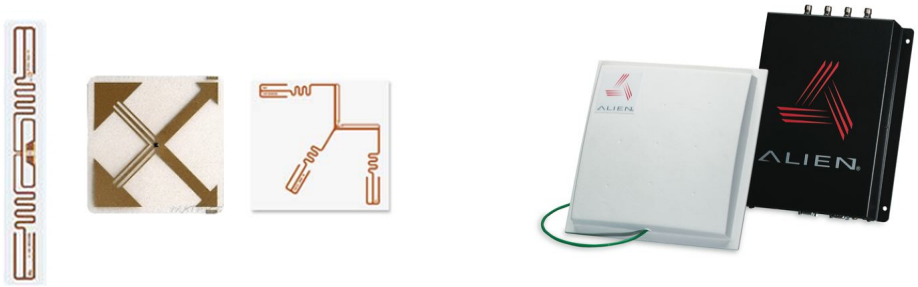
via information extraction is DBLife [DS+07], which stores structured information about the data management research community extracted from pages on the Web.

2.1.2 Sensor Data

Another emerging source of data is streams yielded by sensing devices. Sensors are physical devices designed to monitor some real-world phenomena and report these data, usually at regular intervals. There are a wide variety of sensor types, each with its own characteristics; here, I briefly outline the two main types that are addressed in this thesis.

Radio Frequency Identification (RFID)

RFID is an electronic tagging and tracking technology designed to provide non-line-of-sight identification [Wan04]. For the purposes of this dissertation, a typical RFID installation consists of three components: readers, antennae, and tags (see Figure 2.2). A *reader* uses *antennae* to communicate with *tags* using RF signals to produce lists of tag IDs in its detection field.



(a) Example RFID Tags [Ali05a] (b) RFID Reader and Antenna [ALR05]

Figure 2.2. RFID Components (not to scale).

There are multiple types of RFID technology for different frequencies, the two

most prevalent being 13.56 MHz and 915 MHz. 13.56 MHz RFID has a range of roughly 6 cm and is used primarily for contact-based tagging. 915 MHz technology has a long detection range (roughly 10-20 feet) and is typical of supply chain management-type applications. In this thesis I focus on 915 MHz technology as it is both the most promising as well as the most unreliable RFID technology.

RFID tags may either be passive (no on-board battery) or active (battery-powered). Passive tags are energized by the RF signals from the readers. Both types of tag store a unique identifier code (e.g., a 64- or 96-bit ID for EPCGlobal tags [EPC05a]). Additionally, both active and passive tags may collect other types of information: for example, passive tags have been augmented with accelerometers and light sensors [SSP+06]. I focus primarily on passive tags, as they are the most widespread type of RFID tag.

Readers *interrogate* nearby tags by sending out an RF signal. Tags in the area respond to these signals with their unique identifier code. Since there may be many tags in a reader's vicinity all trying to respond at once, readers typically employ an anti-collision protocol to identify individual tags. As part of this process, readers internally use multiple *interrogation cycles* to determine all tags in the reader's vicinity. The details of these protocols are specific to each type of reader and are typically proprietary.

Externally, the results of multiple reader interrogation cycles are exposed as a group through what I term an *epoch*.[1] For each epoch, the reader keeps track of all the tags it has identified, as well as additional information such as the number of interrogation responses for each tag during that epoch and the time at which the tag was last read. A reader store this information internally in a *tag list* that is periodically transferred to its clients. An example tag list is shown in Table 2.1:

[1] In ALE terms, an epoch is a *read cycle* [ALE05].

each row contains information for a single tag in an epoch, including the tag ID, the number of interrogation responses in that epoch, and a timestamp for the epoch. The epoch size, which may be specified as a number of interrogation cycles or as a unit of time, is configured as part of the initial reader hardware setup. A typical epoch on the order of 0.25 seconds [Ali05b, Sen04].

Tag ID	Responses	Timestamp
8576 2387 2345 8678	9	11:07:06
8576 4577 3467 2357	1	11:07:06
8576 3246 3267 5685	7	11:07:07

Table 2.1. Example RFID reader tag list.

RFID technology is employed in a variety of applications, including inventory management, ID badges and access control, and supply-chain monitoring.

Wireless Sensor Networks

Wireless sensor networks (WSNs) represent another sensing technology for monitoring the physical world. WSNs provide fine-grained information through ad-hoc networking and inexpensive sensing. WSNs consist of many (sometimes hundreds or thousands) of sensor *motes* (see Figure 2.3), each of which consists of a microprocessor, a sensor board containing some number of sensors to collect real-world data (e.g., temperature, light, sound), a wireless radio, and an on-board battery.

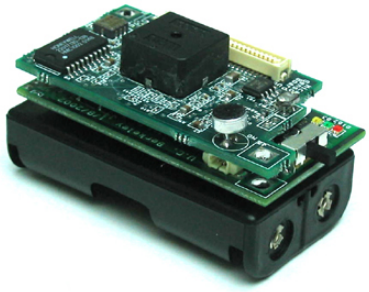


Figure 2.3. An Intel Berkeley Mote.

Upon deployment, a WSN's motes self-organize into an ad-hoc wireless network. Periodically or in response to some stimulus, an individual mote uses its sensor board to collect data. These data are then sent over the wireless network to other motes, where the data can be processed or aggregated and then passed on to other motes via wireless radio. Eventually, the data will reach a *gateway* mote, a mote that is connected to both the wireless network as well as a wired network, where it can be stored in a database or sent over a traditional wired network to other services. Analogous to epochs in RFID readers, this sensing and communication can also be grouped into epochs [MFHH02], where motes are globally scheduled to sense and transmit their data at regular intervals. Under this scheme, the output of a WSN can be thought of as a stream of tuples with a format similar to that shown in Table 2.2. Note that not all motes report during all epochs: messages may be dropped. Also note that the WSN may perform in-network processing such as aggregations (e.g., the average temperature across a set of motes).

Epoch	Mote ID	Temperature	Light	Sound
1	1	22.5	5.6	1.3
1	2	21.3	7.1	2.4
2	2	20.9	4.9	2.4
3	1	21.2	5.0	1.1
3	2	20.7	7.1	2.5

Table 2.2. Example sensor data stream.

There has been a large body of work on the specific operations of WSNs, including different networking [PHL+05] and data processing techniques [MFHH02, BGS01, DGM+04]. Wireless sensors networks have been deployed in a wide range of scenarios such as environmental and habitat monitoring [MCP+02, Son06], home automation [DR05], and traffic monitoring [HBZ+06].

2.1.3 Data Management Challenges

The new sources of data discussed above (i.e., the Web, sensors) present many challenges to applications that seek to make use of this data. I briefly outline some of the challenges here; additional details are provided in subsequent chapters.

With Web data, the primary challenge arises from scale and heterogeneity. First, there are typically many ways of describing real-world concepts and entities. For instance, the two different strings "University of California at Berkeley" and "Cal Berkeley" refer to the same university in reality. In order to pose queries over such data, these two different references must be reconciled to point to the same entity. In large scale repositories such as the Web, solving this problem is compounded by the fact that there is no well-defined domain. To illustrate the challenge in such an environment, consider the string "Berkeley". In the single domain of universities, it maps to the same entity as the two stings above, but when there is no such single domain, this string may map to the university, the city, or even the eighteenth-century British Empiricist George Berkeley.

Sensor data, on the other hand, suffer from their own set of challenges. Most sensing technology is designed for large-scale deployments, such as tagging all products in a retail store or monitoring an entire forest, where cost is the driving factor. As such, these devices are typically built with inexpensive, and thus error-prone, hardware that produces dirty data. These dirty data manifests itself in many forms. First, there is a large amount of missing data: devices are not sensitive enough to detect all phenomena in the real world, they fail outright and do not report any readings, or messages from sensors are dropped when using wireless communication. Another problem with sensor data is that they are often incorrect: devices often fail and produce erroneous readings. Finally, the data can be ambiguous: multiple devices may report conflicting readings. Such challenges have been widely recognized in prior work. Work from

ETH Zurich recognizes the poor behavior of RFID technology [FL04], while the Intel Research Lab in Seattle has characterized the performance and errors in RFID technology in order to better guide ubiquitous applications [FJPR04, HBF+04]. Other sensor-based applications have encountered similar issues in regard to dirty sensor data [BGH+05, DR05]. In Chapters 4, 5, and 6, I present additional data from experimental studies revealing the extent of this problem.

2.2 Data Integration and Cleaning

As mentioned above, there are many challenges posed by these new sources of data. Here, I summarize research to date in both data integration and data cleaning. While historically separate topics, I address both data cleaning and integration as they are both relevant in the process of collecting data from multiple heterogeneous sources into a single dataspace. For more background about data cleaning and integration, see [RD00, DH05] for recent surveys on each.

I first discuss general systems and architectures for data cleaning and integration, including data warehouses, distributed and Web data integration systems, and dataspace systems. I then describe specific data integration and cleaning techniques for both traditional data sources as well as for sensor data.

2.2.1 Data Integration and Cleaning in Data Warehouses

Data warehouses collect large amounts of data from multiple databases to store in one repository. As such, warehouses have extensive support for both data integration and cleaning to deal with errors or inconsistencies in data that typically arise due to input errors, missing data, or heterogeneous data. Techniques employed by data ware-

houses tend to focus on a small set of well-defined tasks, including transformations, matchings, and duplicate elimination.

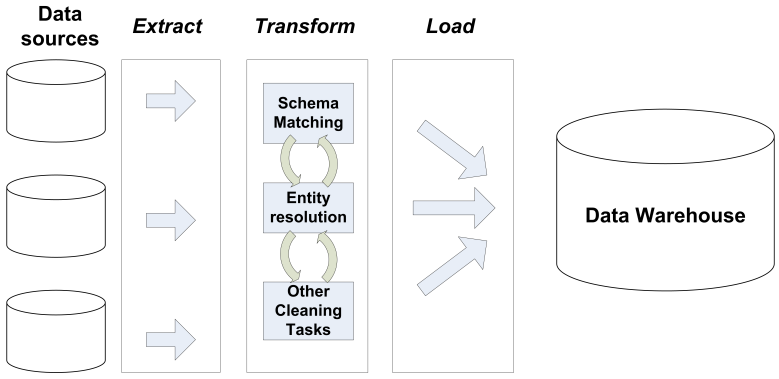


Figure 2.4. The Extract-Transform-Load (ETL) paradigm for integrating multiple data sources into a data warehouse.

Much of this process is captured in the Extract-Transform-Load (ETL) process as illustrated in Figure 2.4. In this process, data from source databases are first extracted from each data source. Then, data are typically processed through a pipeline of stages, each responsible for some aspect of cleaning and integration. Typically, these stages include schema matching and entity resolution [RD00] (I discuss these specific techniques in more detail below). Additionally, these stages handle data corrections such as misspellings and other data entry errors. Finally, data is bulk-loaded into the warehouse. For an overview of data warehousing technology, see [CD97].

2.2.2 Cleaning and Integration Infrastructures

There are a wide variety of tools for assisting in data integration and data cleaning, primarily originating from commercial vendors (e.g., [II08, Ora08, Inf08]). These frameworks typically consist of a suite of mechanisms designed for data analysis and transformations along with tools for organizing such mechanisms in workflows.

In the research community, various projects have built infrastructures for data cleaning and integration (e.g., [VVSK00, HS98, CGGM03, WSF95]).

AJAX [GFSS00] is a tool for developing data cleaning pipelines based on a declarative paradigm. In AJAX, an administrator specifies the high-level operations such as mappings and transformations at the logical level, which are separated from the actual implementations for each operation at the physical level. Potter's Wheel [Vij01] provides an interactive means of cleaning data through a spreadsheet-like interface on which users specify transformations and other cleaning operations. The system also helps to identify dirty data for correction by the user.

2.2.3 Distributed and Web-based Data Integration

Federated and Web-based databases provide another means by which structured data from multiple sources have been integrated. Such systems provide unified access to multiple networked, autonomous data sources through a single interface. Typically, such a system follows a *wrapper-mediator* architecture [Wie92]. In this architecture, a mediator accesses multiple data sources, each of which has its data exposed in a common format through a *wrapper*. Examples of federated data integration systems are the Information Manifold [LRO96], TSIMMIS [CGMH+94], and Garlic [CHS+95].

2.2.4 Dataspace Systems

As mentioned in Chapter 1, dataspace systems are next-generation data integration platforms designed to deal with large collections of heterogeneous data. While the characteristics of external applications and internal composition of each dataspace system vary, such systems share a few common traits. First, they seek to encompass *all* types of data sources (not just structured data in a single domain, for example), and thus typically manage highly heterogeneous data. Second, they unify these sources through a single interface designed such that both existing and emerging applications can leverage dataspace data using traditional data access and exploration techniques.

As such, they support simple keyword searches, complex structured queries, or a mixture of both. Third, dataspace systems are designed to be best-effort: that is, the system returns the best answer it can at the time of query, which may be approximate given the data available. As such, dataspace systems typically support some notion of uncertainty tracking. Finally, since data integration and cleaning in dataspaces is a continuous process, dataspace systems are designed to evolve and adapt to provide tighter integration and better data over time. Many projects are building dataspace systems targeted at different environments; here I outline a few of these efforts.

HiFi is a data integration platform for real-world data designed for processing queries over distributed sensor data streams [FJK+05]. The goal of HiFi is to provide access for traditional applications (e.g., SQL-based) by cleaning, processing, and aggregating disparate data streams derived from both sensors and static data sources. Underlying HiFi is the concept of a *Uniform Declarative Framework*: all levels of the system run on a common declarative interface. This framework enables rapid system deployment, modularity, ease of reasoning, and simple configuration.

PAYGO (an acronym for Pay-As-You-GO) is an architecture for managing and accessing large collections of heterogeneous data such as those found on the Web, designed with the pay-as-you-go principle underlying all aspects of the architecture [MJC+07]. While the application area is very different from HiFi, the goals are similar: data derived from diverse sources need to be cleaned and integrated before they can be used by applications. PAYGO recognizes that as the heterogeneity and scale of the data increases in data integration environments, new techniques are necessary in order to make sense of this data. First, data integration in PAYGO is an ongoing process, and as such, all aspects of the architecture needs to be designed using the pay-as-you-go principle. Second, PAYGO is designed to understand data uncertainty at all levels; it models uncertainty in each of its components and tracks the data's uncertainty as they flow through the system.

Similar to PAYGO, the CIMPLE project [DRC+06] is developing a software platform for creating online communities. CIMPLE focuses on extracting and managing structure from data-rich online communities, as well as leveraging community involvement to assist in cleaning and integration. DBLife [DS+07], as mentioned earlier, is an output of this project.

The iMeMex system [BDG+07] is a dataspace system for managing personal information, such as the data found on a personal computer. This system defines a unified data model for personal data information and then proposes a set of pay-as-you-go techniques for integrating such data (e.g., [SDK+07]).

2.2.5 Data Cleaning and Integration Mechanisms

Having discussed general frameworks and architectures for data cleaning and integration, I now discuss in more detail specific mechanisms used in such systems. In particular, I discuss schema matching, entity resolution, and sensor data cleaning.

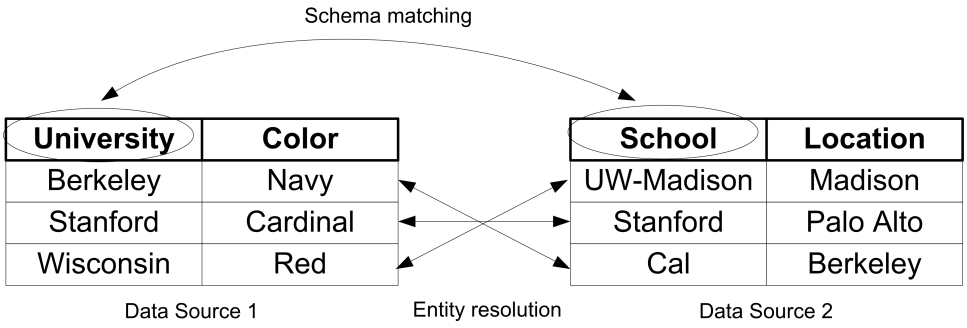


Figure 2.5. Schema matching and entity resolution between two data sources describing universities.

Schema Matching

One common problem with integrating multiple data sources is that the structure in which the data is stored in each source may differ. For instance, one data

source describing universities may use the term "University" where another may use "School", as shown in Figure 2.5. In order to integrate these sources, the schemata of each data source must be mapped to one another; this process is called *schema matching* [DH05].

Many techniques have been developed for schema matching, generally falling under two categories. In rule-based matching [DH05], a set of hand-built rules guide matching decisions. For instance, a rule might state that two schema elements can be deemed equal if they have the same name, data type, and domain. Learning-based matching, on the other hand, determines correspondences between schema elements using machine-learning techniques such as Naive Bayes [Jen96]. As an example of a learning-based mechanism, LSD (Learning Source Descriptions) combines the output of multiple Naive Bayes learners to identify semantic mappings between data sources [DDH01]. See [DH05, RB01] for a broad overview of schema matching techniques.

Entity Resolution

Another challenge with integrating multiple data sources is that the sources may contain multiple, different references to the same real-world entity. For instance, the two databases in Figure 2.5 refer to the same three universities, but with different representations and attributes. When integrating these data sources, these references must be *reconciled* such that there is only one reference to each university.

There are a variety of techniques used for entity resolution[2], again falling into the two basic categories of rule-based and learning-based mechanisms. As an example of rule-based mechanisms, SecondString is a library of a entity-resolution mechanisms that uses string similarity techniques such as edit distance and TF-IDF to determine

[2]This process goes by a variety of terms, including entity resolution, reference reconciliation, object de-duplication, and merge-purge.

if two references point to the same entity [CRF03]. An example of a learning-based mechanism is ALIAS, which uses an active-learning based approach. ALIAS trains its classifiers for determining duplicate entities by selecting training instances for human labeling that are both representative of the larger dataset as well as challenging for automated techniques to resolve [SB02].

Sensor Data Cleaning

Recently, many mechanisms have been developed specifically for cleaning sensor data. Some work has focused on techniques for low-level cleaning, typically involving statistical or machine learning approaches [EN03, MPD04]. The BBQ system uses models of sensor data to accurately and efficiently answer wireless sensor network queries with defined confidence intervals [DGM+04]. Similarly, the work in [PGM05] uses regression applied to sensor networks for inference purposes. These techniques are typically expensive to deploy as they require building and training statistical models.

Other work, in contrast, has addressed high-level cleaning using global information derived from data mining techniques or based on integrity constraints [RDTC06, KBS06]. These approaches suffer from the same heavyweight-deployment challenges as the statistical approaches.

In contrast, several projects in research and industry have explored simple techniques to clean sensor data, typically based on fixed-window smoothing [BGH+05, BLHS04, GS04, iAn06, WL05](see Chapter 5 for a detailed discussion on such techniques). In many cases, such approaches fail to adequately clean the data. All together, these techniques form a suite of tools that can be used to clean sensor data but must be evolved to provide both easy to deploy and effective cleaning.

Note that since sensor data is typically streaming, many of these techniques

can be applied within streaming data processing infrastructures such as TelegraphCQ [CCD+03], STREAM [BBD+02], and Aurora [ACC+03].

2.3 Chapter Summary

In this chapter, I presented background knowledge for the techniques presented in this thesis. In particular, I described two emerging data sources: structured data on the Web and streaming sensor data, which are the focus of this thesis. I then described the challenges associated with processing data from these sources, and the previous and current approaches for doing so.

In the following chapters, I develop specific pay-as-you-go techniques for cleaning and integrating these emerging data sources. The goal of these techniques is to directly address the challenges associated with these new data sources to provide clean and integrated data to dataspace applications such as Total Consumer Awareness.

Chapter 3

Guiding User Feedback to Clean Dataspace Data

In this chapter, I present the first major research contribution of this dissertation, a mechanism for utilizing human feedback for data integration tasks in dataspaces.[1] I develop a technique based on decision theory, specifically the value of perfect information [RN03], that estimates the benefit to the dataspace of possible data integration tasks. Given these estimates, tasks that provide the most benefit are presented to humans for confirmation. I present a performance study on multiple real world and synthetic datasets that investigates the benefits of this approach. Finally, I outline the architecture of *Roomba*, a utility for use within a dataspace system designed to employ this VPI-based integration technique.

[1] Much of the work presented in this chapter has been published in [JFH08].

3.1 Introduction

Applications such as Total Consumer Awareness (as described in Section 1.1) utilize dataspace data from a wide range of heterogeneous sources. One of the primary challenges in supporting these types of applications is large-scale semantic data integration [DH05]. Heterogeneous data originating from disparate sources often contain different representations of the same real-world entity or concept. For example, two employee records in two different enterprise databases may refer to the same person. Similarly, on the Web there are multiple ways of referring to the same product or person, for instance. Typically, a dataspace system employs a set of mechanisms for semantic integration, such as schema matching [RB01] and entity resolution [CRF03], to determine semantic equivalences between elements in the dataspace. The output of these mechanisms are a set of *candidate matches* that state with some confidence that two elements in the dataspace refer to the same real-world entity or concept.

Since these matches, when confirmed, define the semantic equivalences in a dataspace, accurate query processing in a dataspace system depends on the accuracy of these matches being confirmed. Thus, a way to provide better query results in a dataspace system is to confirm candidate matches by soliciting user feedback. For instance, a human could be asked to confirm the equivalence of two schema elements or asked if two objects are duplicates. In a large-scale dataspace (e.g., a dataspace for Web data) there are far too many candidate matches that could benefit from user feedback; the dataspace system may not be able to involve users in all of them. To address this problem, I apply the pay-as-you-go principle: with limited human attention and resources, the dataspace system should ask users to confirm only the most important matches. Thus, the key to soliciting user feedback in such a system is to *order* candidate matches by importance such that the most useful matches can be confirmed first. In fact, this is a common challenge in a set of recent scenarios where

the goal is to leverage mass collaboration, or the so-called *wisdom of crowds* [Sur04], in order to better understand sets of data [vAD04, MDV+03, Fli07].

In this chapter, I consider the problem of determining the order in which to confirm candidate matches to provide the *most benefit* to a dataspace. To this end, I apply decision theory to the context of data integration to reason about such tasks in a principled manner.

I begin by developing a method for ordering candidate matches for user confirmation using the decision-theoretic concept of the *value of perfect information* (*VPI*) [RN03]. VPI provides a means of estimating the benefit to the dataspace of determining the correctness of a candidate match through soliciting user feedback. One of the key advantages of this method is that it considers candidate matches produced from *multiple* mechanisms in a uniform fashion. Hence, the system can weigh, for example, the benefit of asking to confirm a schema match versus confirming the identity of references to two entities in the domain. In this regard, this approach is distinguished from previous research in soliciting user feedback for data integration tasks (e.g., [SB02, WYDM04, DDH01]) that are tightly integrated with a particular mechanism such as schema matching or entity resolution.

At the core of VPI is a *utility function* that quantifies the desirability of a given state of the world; thus to apply VPI to dataspaces, I devise a utility function for dataspaces based on query result quality. Since the correctness of all unconfirmed candidate matches is unknown, the exact utility of a dataspace at a given instant cannot be exactly computed. Thus, I develop a set of approximations that allow the system to efficiently estimate the utility of a dataspace.

I describe a detailed experimental evaluation on both real and synthetic datasets showing that the ordering of user feedback produced by this VPI-based approach yields a dataspace with a significantly higher utility than a variety of other ordering

strategies. I also illustrate experimentally the benefit of considering the output of multiple data integration mechanisms in a single framework: I show that an ordering approach that treats each class of mechanism separately for user-feedback purposes yields poor overall utility. Furthermore, the experiments explore various characteristics of data integration environments to investigate the effect of environmental properties on the efficacy of user feedback.

Finally, I outline the design of Roomba, a system that incorporates this decision-theoretic framework to guide a dataspace system in soliciting user feedback in a pay-as-you-go manner.

This section is organized as follows. Section 3.2 describes the terminology and general problem setting of this chapter. Section 3.3 discusses the decision-theoretic framework I develop for ordering match confirmations. Section 3.4 presents a detailed empirical evaluation of this match ordering strategy. In Section 3.5 I show how to relax the query answering model to consider unconfirmed matches. I describe Roomba in Section 3.6. Section 3.7 presents related work.

3.2 Preliminaries

I begin by describing the problem setting and defining the terms used throughout this chapter.

3.2.1 Dataspace Triples

I model a dataspace D as a collection of *triples* of the form $\langle object, attribute, value \rangle$ (see Table 3.2.1 for an example). Objects and attributes are represented as strings. Values can be strings or can come from other domains (e.g., numbers or dates). Note that the sets of strings used for objects, attributes, and values are *not* necessarily

disjoint. Intuitively, an object refers to some real-world entity and a triple describes the value of some attribute of that entity. I use the term *element* to refer to anything that is either an object, attribute, or value in the dataspace.

I do not assume that the data in the dataspace is actually stored as triples. Rather, the triples may be a logical view over multiple sets of data residing in independent systems.

	⟨**object**, **attribute**, **value**⟩
$t_0 =$	⟨Wisconsin, SchoolColor, Cardinal⟩
$t_1 =$	⟨Cal, SchoolColor, Blue⟩
$t_2 =$	⟨Washington, SchoolColor, Purple⟩
$t_3 =$	⟨Berkeley, Color, Navy⟩
$t_4 =$	⟨UW-Madison, Color, Red⟩
$t_5 =$	⟨Stanford, Color, Cardinal⟩

Example 3.2.1 *An example dataspace with six triples, describing properties of universities.* □

3.2.2 Dataspace Heterogeneity and Candidate Matches

Since the data in a dataspace come from multiple disparate sources, they can display a high degree of heterogeneity. For example, the triples in the dataspace could be collected from a set of databases in an enterprise or from a large collection of tables on the Web. As a result, non-equal strings in a dataspace do not necessarily denote different entities or attributes in the real world. For instance, Example 3.2.1 shows a dataspace describing properties of universities. In this example, the objects "Wisconsin" and "UW-Madison" refer to the same university, the attributes "Color" and "SchoolColor" describe the same property of universities, and the values "cardinal" and "red" are the same in reality.

In a dataspace system, there is a set of *mechanisms* that try to identify such equivalences between elements. In particular, there are techniques for schema matching that predict whether two attributes are the same (see [RB01, DH05] for overviews of such techniques), and there are techniques for entity resolution that will predict whether two object or value references are about the same real-world entity (see [CRF03, Win99] for overviews of these techniques).

The output of these mechanisms are modeled as a set of *candidate matches*, each of the form (e_1, e_2, c), where e_1 and e_2 are elements in the dataspace and c is a number between 0 and 1 denoting the confidence the mechanism has in its prediction. The techniques presented in this chapter are agnostic to the details of the mechanisms.

While in some cases the mechanisms can predict equivalence between elements with complete confidence, most of the time they cannot. Since query processing in a dataspace system depends on the quality of these matches, query results will be better when the matches are certain. Thus, the goal is to solicit feedback from users to *confirm* the matches produced by the mechanisms. Confirming a candidate match involves posing a question about the match to the user that can be answered with a "yes" or a "no". I assume that there exists a separate component that can translate a given candidate match into a such a question.

As discussed above, the number of candidate matches potentially exceeds the amount of human attention available to confirm them; hence, the approach is to confirm matches in a *pay-as-you-go* manner, where confirmations are requested incrementally. This approach takes advantage of the fact that some matches provide more benefit to the dataspace when confirmed than others: they are involved in more queries with greater importance or are associated with more data. Similarly, some matches may never be of interest, and therefore spending any human effort on them is unnecessary.

Since only a fraction of the candidate matches can be confirmed, the challenge is to determine which matches provide the most benefit when confirmed. Hence, the problem I consider here is ordering candidate matches for confirmation to provide the most benefit to the dataspace.

Clearly, the means by which the system asks for confirmation is important. There needs to be some way to formulate a natural language question given a candidate match. Also, the system will likely have to ask multiple users the same question in order to form a consensus in the spirit of the ESP Game [vAD04]. Furthermore, there may be subjective cases where two elements may be the same to some users, but not the same to others (e.g., red and cardinal are the same color to most people, but are different to artist, graphic designers, or college football fans). In this work, I focus on providing the foundation for building solutions to these challenges.

3.2.3 Perfect and Known Dataspace States

To model the benefit of confirming individual candidate matches, I define the *perfect* dataspace D^P corresponding to D. In D^P, all the correct matches have been reconciled and two different strings necessarily represent distinct objects, attributes, or values in the real world. Here, I assume that the real world can be partitioned as such using only strings; in some cases, however, it may be necessary to contextualize the strings with additional metadata to avoid linking unrelated concepts through a common string (e.g., the strings "Cal" and "City of Berkeley" may be incorrectly deemed equivalent if the matches ("Cal","Berkeley", c_1) and ("Berkeley", "City of Berkeley", c_2) are confirmed). Once the equivalence between strings in D is known, D^P can be produced by replacing all the strings belonging to the same equivalence class by one representative element of that class. Of course, keep in mind that D^P is not actually known.

The *known* dataspace for D, on the other hand, consists of the triples in D and the set of equivalences between elements determined by confirmed matches. Whenever the system receives a confirmation of a candidate match, it notes the equivalence of the two elements in the match in a concordance table.

At any given point, query processing is performed on the current known dataspace state. A confirmed match (e_1, e_2, c) causes the system to treat the elements e_1 and e_2 as equivalent for query processing purposes (through the use of a concordance table as discussed above). Initially, I assume that *only* confirmed matches are used in query processing; this requirement is relaxed in Section 3.5 to accommodate other query answering models.

3.2.4 Queries and Workloads

This discussion considers three classes of queries: atomic queries, keyword queries and conjunctive queries.

An *atomic query* is of the form $(object = d)$, $(attribute = d)$, or $(value = d)$ where d is some constant. The answer to an atomic query Q over a dataspace D, denoted by $Q(D)$, is the set of triples that satisfy the equality.

A *keyword query* is of the form k, where k is a string. A keyword query is shorthand for the following: $((object = k) \vee (attribute = k) \vee (value = k))$; i.e., a query that requests all the triples that contain k *anywhere* in the triple.

Finally, a *conjunctive query*, a conjunction of atomic and keyword queries, is of the form $(a_1 \wedge \ldots \wedge a_n)$ where a_i is either an atomic query or a keyword query. The answer returned by a conjunctive query is the intersection of the triples returned by each of its conjuncts.

Recall that when querying the known dataspace D, the query processor utilizes all

the confirmed candidate matches, treating the elements in each match equivalently. On the other hand, the query Q over D^P, the perfect dataspace corresponding to D, takes into consideration all matches that are correct in reality. I denote the result set of Q over D^P by $Q(D^P)$.

When determining which candidate match to confirm, the dataspace system takes into consideration how such a confirmation would affect the quality of query answering on a *query workload*. A query workload is a set of pairs of the form (Q, w), where Q is a query and w is a weight attributed to the query denoting its relative importance. Typically, the weight assigned to a query is proportional to its frequency in the workload, but it can also be proportional to other measures of importance, such as the monetary value associated with answering it, or in relation to a particular set of queries for which the system is to be optimized.

3.2.5 Dataspace Statistics

I assume that the dataspace system contains basic statistics on occurrences of elements in triples in the dataspace. In particular, the dataspace system maintains statistics on the cardinality of the result set of atomic queries over the dataspace D. These queries may be derived from the dataspace workload or may be seeded with important queries. For example, for the atomic query $Q : (object = d)$, the dataspace stores the number of triples in D that have d in their first position, denoted $|D^1_d|$. For conjunctive queries, the DSSP may either maintain multi-column statistics to determine the result sizes of conjunctive queries or use standard techniques in the literature (e.g., [Ioa03, CDS04]) to estimate such cardinalities.

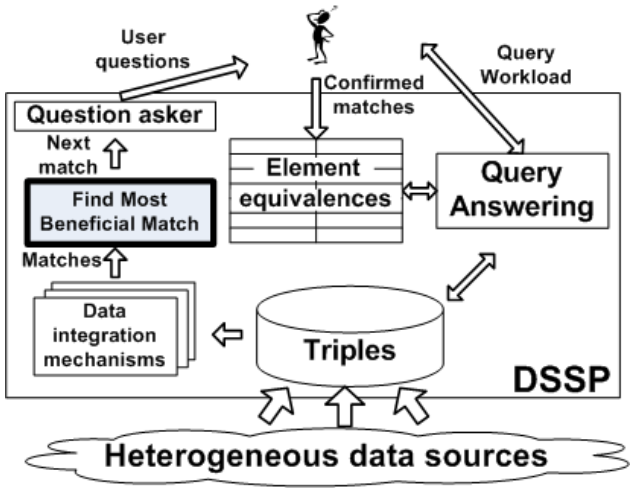


Figure 3.1. An architecture for incorporating user feedback in a dataspace system.

3.2.6 Overall architecture

To summarize, the overall architecture for incorporating user feedback in a dataspace system is shown in Figure 3.1. In this figure, a dataspace system (DSSP) manages multiple heterogeneous data sources. The data from these sources are represented as triples internally in the dataspace system. As noted above, triples may not actually be stored in a dataspace, but could be a logical view of the data in the dataspace. Triples are analyzed by multiple data integration mechanisms (e.g., schema matching, entity resolution) to produce candidate matches. Matches flow into a module that determines the most beneficial match to confirm; this component is the focus of this chapter. A question-asking module converts the most beneficial match into a question suitable for presentation to a user (e.g., "Are the schema elements 'college' and 'university' the same?", possibly with additional context if necessary). When a user confirms a match, it is entered into an equivalence table that notes the semantic equivalence between two dataspace elements. Elements in these correspondences are employed during query processing as discussed above.

3.3 Ordering Match Confirmations

Having laid out the problem setting, this section introduces a decision-theoretic approach to ordering candidate matches for user confirmation in a dataspace. The key concept from decision theory I use is the *value of perfect information* (*VPI*) [RN03]. The value of perfect information is a means of quantifying the potential benefit of determining the true value for some unknown. In what follows, I explain how the concept of VPI can be applied to the context of obtaining information about the correctness of a given candidate match, denoted by m_j.

Suppose we are given a dataspace D and a set of candidate matches $M = \{m_1, \ldots, m_l\}$. Let us assume that there is some means of measuring the *utility* of the dataspace w.r.t. the candidate matches, denoted by $U(D, M)$, which I explain shortly. Given a candidate match m_j, if the system asks the user to confirm m_j, there are two possible outcomes, each with their respective dataspace: either m_j is *confirmed* as correct or it is *rejected* as false. I denote the two possible resulting dataspaces by $D^+_{m_j}$ and $D^-_{m_j}$.

Furthermore, let us assume that the probability of m_j being correct is p_j, and therefore the expected utility of confirming m_j can be expressed as the weighted sum of the two possible outcomes: $U(D^+_{m_j}, M \setminus \{m_j\}) \cdot p_j + U(D^-_{m_j}, M \setminus \{m_j\}) \cdot (1 - p_j)$. Note that these terms do not include m_j in the set of candidate matches because it has either been confirmed or rejected.

Hence, the benefit of confirming m_j can be expressed as the following difference:

$$\begin{aligned} Benefit(m_j) = & U(D^+_{m_j}, M \setminus \{m_j\}) \cdot p_j + \\ & U(D^-_{m_j}, M \setminus \{m_j\}) \cdot (1 - p_j) - \\ & U(D, M). \end{aligned} \tag{3.1}$$

Broadly speaking, the utility of a dataspace D is measured by the quality of the results obtained for the queries in the workload W on D compared to what the dataspace system would have obtained if it knew the perfect dataspace D^P. To define $U(D, M)$, I first need to define the result quality of the query Q over a dataspace D, which I denote by $r(Q, D, M)$.

Recall that Q is evaluated over the dataspace D with the current *known* set of confirmed matches. Since our queries do not involve negation, all the results the dataspace system returns will be correct w.r.t. D^P, but there may be missing results because some correct matches are not confirmed. Hence, I define

$$r(Q, D, M) = \frac{|Q(D)|}{|Q(D^P)|}$$

and the utility of the dataspace is defined as the weighted sum of the qualities for each the queries in the workload:

$$U(D, M) = \sum_{(Q_i, w_i) \in W} r(Q_i, D, M) \cdot w_i. \tag{3.2}$$

The goal is to order the matches to confirm by the benefit outlined in Equation 3.1: the matches that are expected to produce the most benefit when confirmed are presented to the user first. However, in order to put this formula to use, we still face two challenges. First, we do not know the probability p_j that candidate match m_j is correct. Second, since we do not know the perfect dataspace D^P, we cannot actually compute the utility of a dataspace as defined in Equation 3.2.

I address the first challenge by approximating the probability p_j by c_j, the confidence measure associated with candidate match m_j. In practice, the confidence numbers associated with the candidate matches are *not* probabilities, but for our purposes it is a reasonable approximation to interpret them as such. In Section 3.4,

I show how to handle cases where such an approximation does not hold. The second challenge is the topic of the next subsection.

3.3.1 Estimated Utility of a Dataspace

In what follows, I show how to estimate the utility of a dataspace using *expected utility*. Note that the set M represents the uncertainty about how D differs from the perfect dataspace D^P; any subset of the candidate matches in M may be correct. I denote the expected utility of D w.r.t. the matches M by $EU(D, M)$ and the expected quality for a query Q w.r.t. D and M as $Er(Q, D, M)$.

Once we have $EU(D, M)$, the value of perfect information w.r.t. a particular match m_j is expressed by the following equation, obtained by reformulating Equation 3.1 to use c_j instead of p_j and to refer to expected utility rather than utility:

$$\begin{aligned} VPI(m_j) = & EU(D^+_{m_j}, M \setminus \{m_j\}) \cdot c_j + \\ & EU(D^-_{m_j}, M \setminus \{m_j\}) \cdot (1 - c_j) - \\ & EU(D, M). \end{aligned} \tag{3.3}$$

The key to computing $EU(D, M)$ is to estimate the size of the result of a query Q over the perfect dataspace D^P. I illustrate the method for computing $Er(Q, D, M)$ for atomic queries of the form $Q : (object = d)$, where d is some constant. The reasoning for other atomic, keyword, and conjunctive queries is similar. Once we have $Er(Q, D, M)$, the formula for $EU(D, M)$ can be obtained by applying Equation 3.2.

Let us assume that the confidences of the matches in M being correct are independent of each other and that M is a *complete* set of candidates; i.e., if e_1 and e_2 are two elements in D and are the same in D^P, then there will be a candidate match

(e_1, e_2, c) in M for some value of c. Given the dataspace D, there are multiple possible perfect dataspaces that are consistent with D and M. Each such dataspace is obtained by selecting a subset $M_1 \subseteq M$ as *correct* matches, and $M \setminus M_1$ as *incorrect* matches. I denote the perfect dataspace obtained from D and M_1 by D^{M_1}.

$Er(Q, D, M)$ is computed by the weighted result quality of Q on each of these candidate perfect dataspaces. Since it is assumed that the confidences of matches in M are independent of each other, $Er(Q, D, M)$ is computed as follows:

$$Er(Q, D, M) = \sum_{M_1 \subseteq M} \frac{|Q(D)|}{|Q(D^{M_1})|} Pr(D^{M_1}) \tag{3.4}$$

where

$$Pr(D^{M_1}) = \prod_{m_i \in M_1} c_i \cdot \prod_{m_i \notin M_1} (1 - c_i).$$

Finally, to compute Equation 3.4 we need to evaluate $|Q(D^{M_1})|$, the estimated size of Q on one of the possible candidate perfect dataspaces.

Recall that the size of Q over D, $|Q(D)|$, is the number of triples in D where d occurs in the first position of the triple. Hence, $|Q(D)| = |D^1_d|$, which can be found using the statistics available on the dataspace. In D^{M_1}, the constant d is deemed equal to a set of other constants in its equivalence class, $d_1, \ldots, d_m$. Hence, the result of Q over D^{M_1} also includes the triples with $d_1, \ldots, d_m$ in their first position and therefore $|Q(D^{M_1})| = |D^1_d| + |D^1_{d_1}| + \ldots + |D^1_{d_m}|$, which can also be computed using the dataspace statistics.

3.3.2 Approximating Expected Utility

In practice, we do not want to compute Equation 3.4 exactly as written because it requires iterating over all possible candidate perfect dataspaces, the number of which is exponential in $|M|$, the size of the set of candidate matches. Hence, in this subsection I show how to approximate $EU(D, M)$ using several simplifying assumptions. The experimental evaluation in Section 3.4 shows that despite these approximations, this approach produces a good ordering of candidate matches.

Two approximations are already built into the development of Equation 3.4. First, the confidences of the matches in M are not necessarily independent of each other. There may, for instance, be one-to-one mappings between two data sources such that a rejected match between two elements would increase the confidence of all other matches in which those elements participate; such considerations can be layered on top of the techniques presented here. Second, the set M may not include *all* possible correct matches, though we can always assume there is a candidate match for every pair of elements in D.

The main approximation made when computing the VPI w.r.t. a candidate match m_j of the form (e_1, e_2, c_j) is to assume that $M = \{m_j\}$. That is, it is assumed that M includes *only* the candidate match for which we are computing the VPI. The effect of this assumption is that we consider only two candidate perfect dataspaces, one in which m_j holds and the other in which m_j does not hold. These two perfect dataspaces are denoted by $D_{m_j}^{e_1=e_2}$ and $D_{m_j}^{e_1\neq e_2}$, respectively.

Given this approximation, we can rewrite Equation 3.4 where $\{m_j\}$ is substituted for M:

$$Er'(Q, D, \{m_j\}) = \frac{|Q(D)|}{|Q(D^{e_1=e_2}_{m_j})|}c_j + \frac{|Q(D)|}{|Q(D^{e_1\neq e_2}_{m_j})|}(1 - c_j) \tag{3.5}$$

and therefore the expected utility of D w.r.t. $M = \{m_j\}$ can be written as

$$\begin{aligned} EU(D, \{m_j\}) &= \sum_{(Q_i,w_i)\in W} w_i \cdot (\frac{|Q_i(D)|}{|Q_i(D^{e_1=e_2}_{m_j})|}c_j + \frac{|Q_i(D)|}{|Q_i(D^{e_1\neq e_2}_{m_j})|}(1 - c_j)) \\ &= \sum_{(Q_i,w_i)\in W} w_i \cdot \frac{|Q_i(D)|}{|Q_i(D^{e_1=e_2}_{m_j})|}c_j + \\ &\quad \sum_{(Q_i,w_i)\in W} w_i \cdot \frac{|Q_i(D)|}{|Q_i(D^{e_1\neq e_2}_{m_j})|}(1 - c_j). \end{aligned} \tag{3.6}$$

3.3.3 The Value of Perfect Information

Now let us return to Equation 3.3. By substituting $\{m_j\}$ for M, we obtain the following:

$$\begin{aligned} VPI(m_j) = &EU(D^+_{m_j}, \{\}) \cdot c_j + \\ &EU(D^-_{m_j}, \{\}) \cdot (1 - c_j) - \\ &EU(D, \{m_j\}). \end{aligned} \tag{3.7}$$

Now note that once we employ the assumption that $M = \{m_j\}$, $Er'(Q, D^+_{m_j}, \{\})$ and $Er'(Q, D^-_{m_j}, \{\})$ are both 1 because they evaluate the utility of a dataspace that is the same as its corresponding perfect dataspace. Thus, using these values with Equation 3.6, $EU(D^+_{m_j}, \{\}) \cdot c_j$ and $EU(D^-_{m_j}, \{\}) \cdot (1 - c_j)$ become $\sum_{(Q_i,w_i)\in W} w_i \cdot c_j$

and $\sum_{(Q_i,w_i)\in W} w_i \cdot (1 - c_j)$, respectively. Furthermore, note that the last term at the end of Equation 3.6 also evaluates the utility of a dataspace that is the same as its corresponding perfect dataspace and thus simplifies to $\sum_{(Q_i,w_i)\in W} w_i \cdot (1 - c_j)$. Therefore, this term cancels with the second term of Equation 3.7. Hence we are left with the following:

$$\begin{aligned} VPI(m_j) = & \sum_{(Q_i,w_i)\in W} w_i \cdot c_j - \\ & \sum_{(Q_i,w_i)\in W} w_i \cdot c_j \frac{|Q_i(D)|}{|Q_i(D_{m_j}^{e_1=e_2})|} \\ = & \sum_{(Q_i,w_i)\in W} w_i \cdot c_j \left(1 - \frac{|Q_i(D)|}{|Q_i(D_{m_j}^{e_1=e_2})|}\right). \end{aligned}$$

Finally, observe that only queries in W that refer to either e_1 or e_2 can contribute to the above sum; otherwise, the numerator and denominator are the same. Hence, if W_{m_j} denotes the set of queries that refer to either e_1 or e_2, then the above formula can be restricted to yield the following. This equation forms the basis for my VPI-based user feedback ordering approach.

$$VPI(m_j) = \sum_{(Q_i,w_i)\in W_{m_j}} w_i \cdot c_j \left(1 - \frac{|Q_i(D)|}{|Q_i(D_{m_j}^{e_1=e_2})|}\right). \tag{3.8}$$

By calculating the VPI value for each candidate match using this equation, a dataspace system can produce a list of matches ordered by the potential benefit of confirming the match.

Example 3.3.1 *Consider an example unconfirmed candidate match* m_j = (*"red"*, *"cardinal"*, 0.8) *in the dataspace from Example 3.2.1. The value of perfect information for* m_j *is computed as follows.*

Assume that the dataspace workload W contains two queries relevant to m_j, Q_1 : (value = "red") with a weight $w_1 = 0.9$ and Q_2 : (value = "cardinal") with a weight $w_1 = 0.5$, and thus $W_{m_j} = \{(Q_1, 0.9), (Q_2, 0.5)\}$. In the example dataspace D, the cardinalities of the two relevant values in the third position is $|D^3{}_{\text{"red"}}| = 1$ and $|D^3{}_{\text{"cardinal"}}| = 2$. Therefore, the query cardinalities in the known dataspace D are $|Q_1(D)| = |D^3{}_{\text{"red"}}| = 1$ and $|Q_2(D)| = |D^3{}_{\text{"cardinal"}}| = 2$. In the perfect dataspace where the values "cardinal" and "red" refer to the same color, the cardinality for both queries is $|Q(D_{m_j}^{\text{"red"}=\text{"cardinal"}})| = |D^3{}_{\text{"red"}}| + |D^3{}_{\text{"cardinal"}}| = 3$.

Applying Equation 3.8 to compute the VPI for m_j, we have:

$$\begin{aligned} VPI(m_j) =& w_1 \cdot c_j \left(1 - \frac{|Q_1(D)|}{|Q_1(D_{m_j}^{\text{"red"}=\text{"cardinal"}})|}\right) + \\ & w_2 \cdot c_j \left(1 - \frac{|Q_2(D)|}{|Q_2(D_{m_j}^{\text{"red"}=\text{"cardinal"}})|}\right) \\ =& 0.9 \cdot 0.8 \left(1 - \frac{1}{3}\right) + 0.5 \cdot 0.8 \left(1 - \frac{2}{3}\right) = 0.61. \end{aligned}$$

This value represents the expected increase in utility of the dataspace after confirming candidate match m_j. □

3.4 Experimental Evaluation

In this section I present a detailed experimental evaluation of the VPI-based approach presented in the previous section, using both real-world and synthetic datasets .

3.4.1 Google Base Experiments

The first set of experiments I present uses real-world datasets derived from Google Base [GB07].

Experimental Setup

Google Base. Google Base is an online repository of semi-structured, user-provided data. Recall from Chapter 2 that the primary unit of data in Google Base is an *item*, each of which consists of an item ID and one or more attribute/value pairs. An item can be converted to one or more triples, each containing the item ID and one of the attribute/value pairs. Items are organized into *item types* such as vehicles or recipes. While there are recommended attributes and values for each item, users can make up their own attributes and values when uploading items. Since each user is free to use their own terminology in their items, there are many cases where different strings for attributes or values in two different items refer to the same concept in reality. Examples of such correspondences are: ("address" $\leftrightarrow$ "location"), ("door_count" $\leftrightarrow$ "doors"), and ("Tan" $\leftrightarrow$ "beige").

Google Base datasets. To collect the data used for our experiments, I sampled Google Base to get 1000 elements from each of the 16 standard item types recommended by Google Base. These item types encompass most of the data in Google Base across a wide range of domains. The item types sampled from are as follows: business locations, course schedules, events and activities, housing, jobs, mobile, news and articles, personals, products, recipes, reference articles, reviews, services, travel packages, vehicles, wanted ads.

I partitioned this sample into two datasets (characteristics for these datasets are shown in Table 3.1):

• *Full*: This dataset contains all 16 item types, and thus represents the full range of semantic heterogeneity that exists in Google Base. Due to this heterogeneity, correct correspondences are hard for the mechanisms (described below) to determine and

Characteristic	*Restricted*	*Full*
Total items	8000	16000
Total triples	60716	148050
Unique triples	52773	113489
Total attributes	333	708
Total values	6900	16924
Total elements	7233	17632
Candidate matches	839	4853
Percent correct matches	0.33	0.23
Avg matches per element	2.8	4.9

Table 3.1. Statistics for the Google Base datasets. *Total items* denotes the total number of Google Base items in each dataset. Each of these items may contain multiple triples and thus *total triples* is the total number of triples contained in those items. Many of these triples are identical: *unique triples* is the the number of unique triples in each dataset. In these datasets, the elements of interest are attributes and values; this table shows the counts for each of these types of elements and the total number of elements. Additionally, I show statistics for the matches: the total number of matches produced by the mechanisms (*Candidate matches*), the fraction of correct matches (*Percent correct matches*), and the average number of matches in which each element participates (*Avg matches per element*).

thus many of the matches are incorrect; only 23% of the matches in this dataset are correct matches.

• *Restricted*: This dataset contains a relatively small amount of semantic heterogeneity: there are fewer cases where two strings refer to the same real-world entity. Thus, the mechanisms produce a smaller number of matches and a higher percentage of correct matches (see Table 3.1). This dataset was created by selecting items from item types that contained less heterogeneity. These item types are as follows: business locations, housing, news and articles, products, recipes, reference articles, reviews, wanted ads.

Ground truth. In order to evaluate the VPI-based techniques against ground-truth,

I manually annotated the dataset with correspondences between attributes and values in each dataset.

Mechanisms. I employ two different types of mechanisms, each applied to both attribute and value matching.

For the first mechanism, I use Google Base's internal attribute and value matching, termed *adjustments*, designed to convert attributes and values to canonical strings. For example, the attribute "Address" is converted to "location" and the value "F" is converted to "female". The Google Base data includes for each adjustment both the original string and the adjusted string. Each original and adjusted string pair represents a candidate match. Note that matches produced by this mechanism are particularly challenging to use in VPI calculations as the matches *do not* have an associated confidence. I discuss how to address this challenge below.

The second mechanism I use is the SecondString library [SS07] for approximate string matching. This mechanism also operates on both attributes and values. From this library, I use the *SoftTFIDF* approach [CRF03], a hybrid approach combining the Jaro-Winkler metric [Win99] (a measure based on how many characters the strings have in common and their ordering) with TF-IDF similarity. This mechanism was shown to have the best string matching accuracy in a variety of experiments [CRF03]. For a match's confidence, this mechanism produces a "score", a number between 0 and 1, with 1 indicating the highest degree of similarity. While this score is loosely correlated with the probability the match is correct, it is inaccurate. To limit the number of candidate matches, only matches with a score greater than 0.5 are considered for confirmation.

Candidate matches. Using these two mechanisms, I produce a set of candidate

matches for each dataset. These matches are annotated as correct or incorrect as determined by the manual annotation.

Converting confidences to probabilities. Since the VPI-based approach uses the confidence numbers produced by mechanisms as probabilities to drive its VPI calculations, a major challenge presented by these candidate matches is to convert the match confidences into probabilities.

To address this challenge, I use a histogram-based learning approach. As matches are confirmed, the dataspace system compiles how many times each mechanism produces correct or incorrect matches and the corresponding input confidence (or lack thereof) for each match. Using these observed probabilities, the system maintains a histogram for each mechanism that maps the confidence value for matches produced by that mechanism to a probability: each bucket corresponds to an input confidence range and the bucket contains the computed probability for that input confidence. Upon each confirmed match (or after a batch of confirmed matches), the system recomputes the probabilities for each mechanism and then applies the computed probabilities to the remaining candidate matches for that mechanism based on their confidences. Note that for mechanisms without any confidence value (e.g., the Google Base adjustments in this scenario), this approach assigns a single confidence value based on the percentage of correct matches produced (and confirmed as correct) by that mechanism (i.e., the histogram only has one bucket).

Queries. I use a query generator to generate a set of queries. Each generated query refers to a single element and is representative of the set of queries that refer that element. For simplicity, the generator only produces keyword queries. The generator assigns to each query a weight w using a distribution to represent the frequency of queries on this element. Since the distribution of query-term frequencies on Web search engines typically follows a long-tailed distribution [SHMM98], for w

in these experiments values are selected from a Pareto distribution [CB02]. Other query workloads below are evaluated below.

Match ordering strategies. I compare a variety of candidate match ordering strategies. Each strategy implements a $score(m_j)$ function, which returns a numerical score for a given candidate match $m_j = (e_1, e_2, c_j)$. A higher score indicates that a candidate match should be confirmed sooner.

• VPI: $score(m_j) = VPI(m_j)$. Each candidate match is scored with the value of perfect information as defined in Equation 3.8.

• $QueryWeight$: $score(m_j) = \sum_{(Q_i,w_i)\in W_{m_j}} w_i$. Each candidate match is scored with the sum of query weights for that match's relevant queries. The intuition behind this strategy is that important queries should have their relevant matches confirmed earlier.

• $NumTriples$: $score(m_j) = |D_{e_1}| + |D_{e_2}|$. This strategy scores each candidate match by number of triples in which the two elements in the match appear. The rationale behind this strategy is that matches containing elements appearing in many triples are more important since queries involving the elements in these matches will miss more data if the match is correct but not confirmed.

• *GreedyOracle*: For each unconfirmed candidate match, this strategy runs the entire query workload W and measures the actual increase in utility resulting from confirming that match. To calculate the actual utility, this strategy uses the manual annotation for all matches to determine if they are correct or not in reality. The match with the highest resulting utility is chosen as the next confirmation. Note that this strategy is not a realistic ordering approach as it relies on knowing the correctness of all candidate matches as well as running the entire workload for all unconfirmed matches for each match confirmation. It represents an upper-bound on any myopic strategy.

• *Random*: Finally, the naive strategy for ordering confirmations is to treat each candidate match as equally important. Thus, the next match to confirm in this strategy is chosen randomly. This strategy provides a baseline to which the above strategies can be compared

Confirming candidate matches. All candidate matches are scored using each of the above strategies and the match with the highest score is chosen to confirm using the correct answer as determined by the manual annotation. After each confirmed match, the set of equivalence classes are updated and the process repeats.

Measurement. After confirming some percentage of candidate matches using each of the orderings produced by each strategy, I run the query workload W over the dataspace and measure the utility using the utility function defined in Equation 3.2. I report the percent of improvement in utility over a dataspace with no confirmed matches. Conceptually, a higher value for this metric means that query results have a higher recall, especially for important queries (as defined by the query weights).

Basic Tests

To study the basic efficacy of the VPI-based approach, the first experiments investigate the performance of different ordering strategies on both of the Google Base datasets. I use the basic setup as described above with each dataset.

The resulting utility produced by confirming matches ordered by each strategy for the *Full* dataset is shown in Figure 3.2. The results in this graph can be interpreted as follows. Since only a small fraction of candidate matches can be confirmed in a large-scale dataspace, the goal is to provide the highest utility with as few confirmations as possible. Thus, the slope of the curve at lower percentages of confirmations is the key component to the curve: the steeper the slope, the better the ordering.

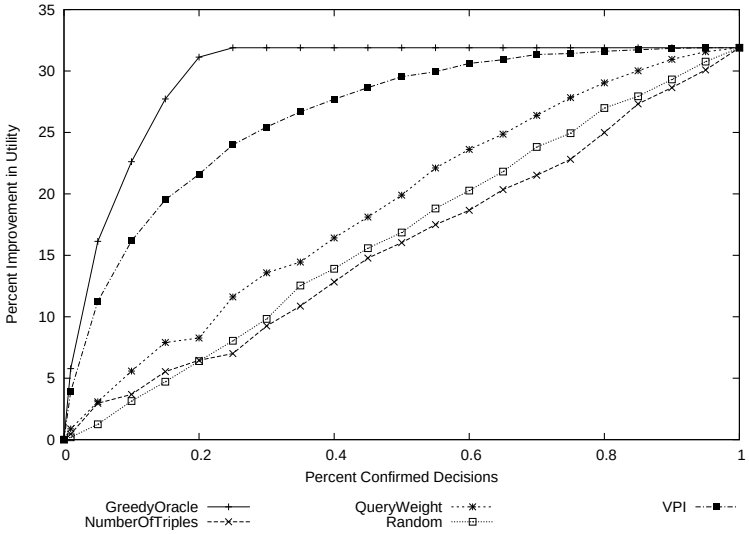


Figure 3.2. Basic test comparing a VPI-based approach for ordering user feedback to other approaches run over the *Full* dataset.

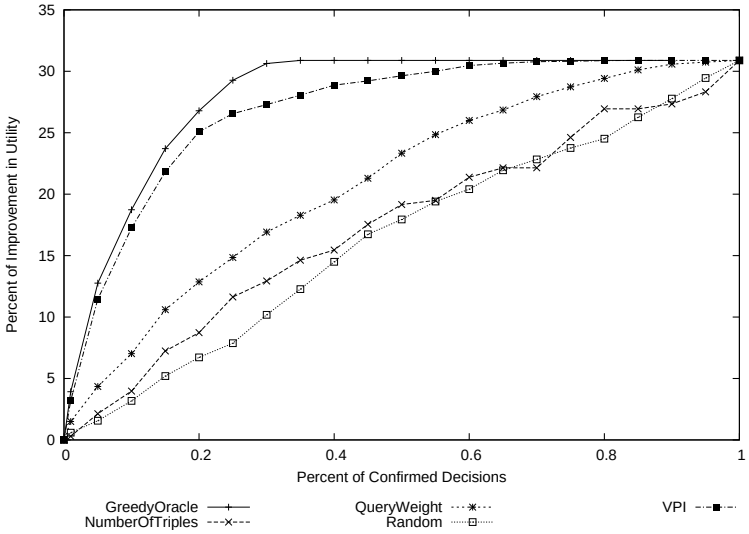


Figure 3.3. Basic test comparing a VPI-based approach for ordering user feedback to other approaches run over the *Restricted* dataset.

First observe the curve for the *GreedyOracle* strategy. This approach selects the most beneficial candidate matches to confirm first and thus the curve is very steep for the early confirmations. As it confirms more matches, the curve flattens out as these matches provide less benefit to the dataspace. Finally, it converges to the utility of the perfect dataspace, i.e., the dataspace where all element equivalences are known.

As can be seen, the VPI strategy performs comparatively well. The VPI strategy performs well despite the fact that this dataset is particularly challenging for any non-oracle strategy as there is a large degree of semantic heterogeneity. First, this dataset contains many incorrect matches, which provide no benefit to the dataspace's utility when rejected. More challenging, however, is that many of the incorrect matches are given similar confidences. As an example of the challenges in this dataset, consider the following matches created by the SecondString mechanism in the vehicles item type: the candidate match (*"AM/FM Stereo Cassette/Cd"*, *"AM/FM Stereo Cassette & CD Player"*) is correct in reality with a confidence of 0.91 while the match (*"AM/FM Stereo Cassette/Cd"*, *"AM/FM Stereo Cassette"*) is incorrect in reality but is given a similar confidence of 0.92. The VPI strategy is unable to discern between these two matches and thus occasionally incorrectly orders them. Despite these challenges, however, the VPI strategy substantially outperforms all other non-oracle strategies.

In contrast, the slopes of the curves for the other strategies are much shallower; it takes many more confirmations to produce a dataspace with a high utility. The $NumTriples$ strategy does particularly poorly. These results emphasize the importance of considering the query workload when selecting candidate matches for confirmation: $NumTriples$ performs poorly because it fails to consider the workload. The utility of the dataspace increases roughly linearly as the percent of confirmations increase for the *Random* curve since it treats each candidate match as equally important.

Strategy	10% matches confirmed	0.95 of perfect dataspace
VPI	17.2	0.20
$QueryWeight$	7.0	0.55
$NumTriples$	3.9	0.75
$Random$	3.2	0.80
$GreedyOracle$	18.7	0.20

Table 3.2. Two measures of candidate match ordering effectiveness (shown for the *Restricted* dataset). The first column shows the resulting percent of improvement after confirming 10% percent of the matches. The second column shows the fraction of confirmed matches required to reach a dataspace whose utility is 0.95 of the utility of the perfect dataspace.

The results for the basic tests on the *Restricted* dataset are shown in Figure 3.3. Here, the VPI-strategy does particularly well; it tracks the $GreedyOracle$ curve closely. Since this dataset contains less semantic heterogeneity than the *Full* dataset, the VPI strategy is able to easily discern the best matches to confirm.

While these graphs provide a holistic view of how each strategy performs, I present in Table 3.2 two alternative views of this data (for the *Restricted* dataset) to better illustrate the effect of match ordering strategy on dataspace utility. First, since a DSSP for a large-scale dataspace can only request feedback for a very small number of candidate matches, I report the improvement after a small fraction of confirmed matches (here, 10%). Second, since the goal of user feedback is to move the known dataspace state towards the perfect dataspace, I report how many confirmations are required from each strategy until the utility of the dataspace reaches some fraction of the utility of the perfect dataspace (here, 0.95).

These numbers further emphasize the effectiveness of a VPI-based ordering approach. With only a small percentage of confirmations in using the VPI strategy, the utility of the dataspace closely approaches the utility of the perfect dataspace. For instance, with only 20% of the confirmations, the VPI strategy is able to produce a

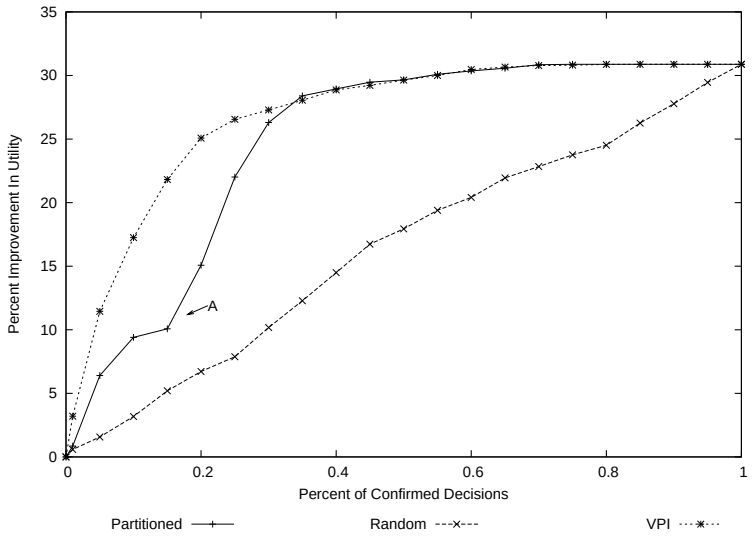


Figure 3.4. Experiment comparing a strategy that separately orders candidate matches from disparate mechanisms to the VPI strategy, run on the *Restricted* dataset.

dataspace that is 95% of the perfect dataspace, equivalent to the oracle strategy and over twice as fast as the next-best strategy.

Partitioned Ordering

To study the need for a single unifying means of reasoning about user feedback in a dataspace, I compare the VPI approach to an ordering algorithm that treats candidate matches produced by different mechanisms separately. The general idea with this strategy is that the output of each mechanism is ordered separately in a *partition* for each mechanism's matches, and then these partitions are ordered. I term this ordering strategy *Partitioned*, where $score(m_j)$ is calculated as follows. The algorithm separates its matches into partitions corresponding to the output of the two mechanisms. These partitions are roughly ordered based on the relative performance of each mechanism. For this experiment, *Partitioned* orders Google Base adjustments first, followed by SecondString matches. To provide a fair comparison, within each partition the matches are ordered by their VPI score. This strategy represents the

case where individual mechanisms each perform their own ordering; there is no global ordering beyond deciding how to the order partitions.

For this experiment, I compare the *Partitioned* strategy to the VPI strategy using the experimental setup as in the basic tests using the *Restricted* dataset. I also include the curve for the *Random* strategy for reference. The results are shown in Figure 3.4.

Here we can see the distinct phases of match confirmations in the *Partitioned* curve: in the early confirmations (prior to point A), the algorithm confirms matches from Google Base adjustments, after which (after point A) it confirms SecondString matches.

Of note in this figure is the tail end of the Google Base confirmation phase (point A) and the start of the SecondString confirmation phase (point B). The highly-ranked SecondString confirmations provide more benefit than the lower-ranked Google Base confirmations, but since the two types of candidate matches are ordered separately, these non-beneficial Google Base matches are confirmed first and thus the overall utility of the dataspace suffers: at 10% confirmations, the percent improvement for *Partitioned* is 9.4%, whereas with the VPI strategy it is 17.2%.

This problem is not just a result of this particular setup or due to the details of the *Partitioned* strategy. Rather, any strategy that treats the output from different mechanisms separately will suffer from the same issues we see here. The problem is a result of multiple independent mechanisms using different, incomparable means of ordering candidate matches and thus there is no way to balance between the output of multiple mechanisms.

The key to solving this problem is that candidate matches from different mechanisms need to be globally ordered based on the overall benefit to the dataspace and not based on their ranking relative to matches produced within each mechanism. The

VPI strategy scores matches from different mechanisms in a uniform manner based on the expected increase in utility of the dataspace on confirmation; thus it is able to interleave match confirmations from multiple mechanisms in a principled manner to produce a dataspace with a higher utility using less user feedback.

Other Query Workloads

The above experiments used a query workload representative of a workload found on a Web search engine; here, I explore the effectiveness of the VPI approach in other environments by experimenting with different query workloads on the same dataset. I generate workloads for the *Full* dataset containing query weights using different types of distributions: *Pareto* (modified to produce values between 0 and 1) as was used above, a Normal distribution with a mean of 0.5 and a standard deviation of 0.25 ($Normal(0.5, 0.25)$), a Uniform distribution between 0 to 1 ($Uniform(0, 1)$), and a Uniform distribution between 0.5 to 1 ($Uniform(0.5, 1)$). I generate a curve for each query weight distribution using the VPI ordering strategy. Additionally, I show the curve for the *Random* strategy to provide a baseline. Changing the query weights used to generate the workload affects the overall utility of the dataspace; thus, in order to present all curves on the same graph, I report the percent of potential improvement in utility: i.e., the ratio of improvement between a dataspace with no confirmations and the perfect dataspace. The results are shown in Figure 3.5.

Observe that while the VPI strategy performs best with the highly-skewed *Pareto* query workload, its effectiveness on all other workloads is close to that of the *Pareto* workload. Thus, across a range of query workloads in this scenario, the VPI strategy effectively orders candidate matches for confirmation.

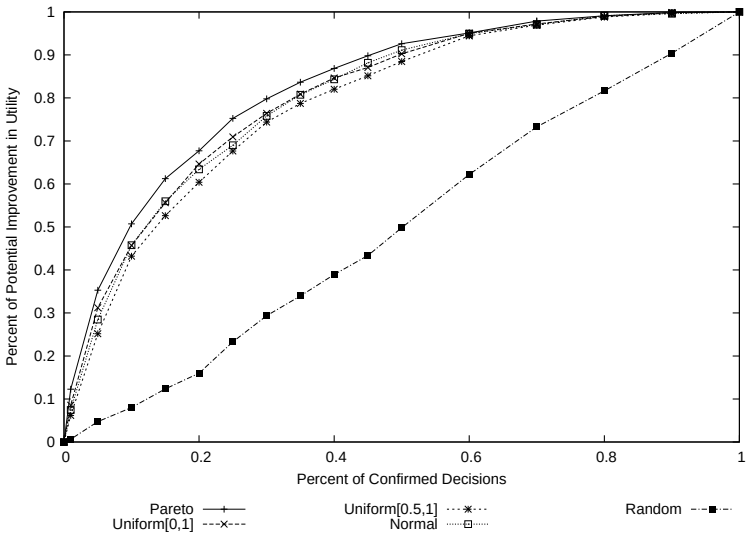


Figure 3.5. The effect of different query workloads on the performance of the VPI strategy run over the *Full* dataset.

Different Means of Assigning Confidence

Recall that Google Base does not assign confidences to its adjustments and the confidences for the matches produced by SecondString are not accurate as probabilities. In the experiments above, I used the histogram-based approach for converting confidences to probabilities as discussed in Section 3.4.1. Here, I investigate the effectiveness of this approach on the VPI-based strategy's performance.

I compare the histogram-based technique as used in the previous experiments (labeled here as *VPI with full histogram*) to three other VPI-based approaches and the *Random* strategy (shown in Figure 3.6). In *VPI with 0.5 for GB*, the confidences for the matches produced by Google Base are set to 0.5, while the confidences for the SecondString matches are left as is. This is the baseline strategy. In *VPI with GB histogram*, the Google Base matches are converted to probabilities using the histogram technique while the SecondString matches are left as is. In *VPI with SStr histogram*, the SecondString matches are converted using the histogram technique while the confidence for the Google Base matches are left at 0.5. Note that I also

experimented with different bucket sizes for the histogram technique. As expected, the effectiveness is higher at smaller bucket sizes, although there is not a marked difference. For all experiments with the histogram approach, the bucket size is set to 0.01.

The results of this experiment are presented in Figure 3.6. First, observe that the curve for *VPI with 0.5 for GB* provides the least benefit for ordering confirmations. This performance is due to the inaccurate confidences produced by SecondString and the lack of confidences for Google Base matches. A slight improvement occurs when using the histogram approach to convert the confidence for the Google Base matches (*VPI with GB histogram*). The improvement is small because the conversion is very coarse-grained: all Google Base matches have no input confidence and thus get assigned to the same histogram bucket resulting in all Google Base matches getting the same output probability. When the SecondString match confidences are converted to probabilities, we see a large jump in the VPI strategy's effectiveness (*VPI with SStr histogram*). Here, the histogram approach is able to map input confidences to probabilities at a very fine resolution. Finally, when confidences for both SecondString and Google Base matches are converted (*VPI with full histogram*), the VPI strategy has the best performance, although only slightly more than just converting the SecondString matches due to the issues with the Google Base matches mentioned above. This experiment shows that the histogram-based technique for converting confidences to probabilities can provide a substantial increase in effectiveness for the VPI strategy, but in order to realize its full potential it is better to have initial confidences assigned per match.

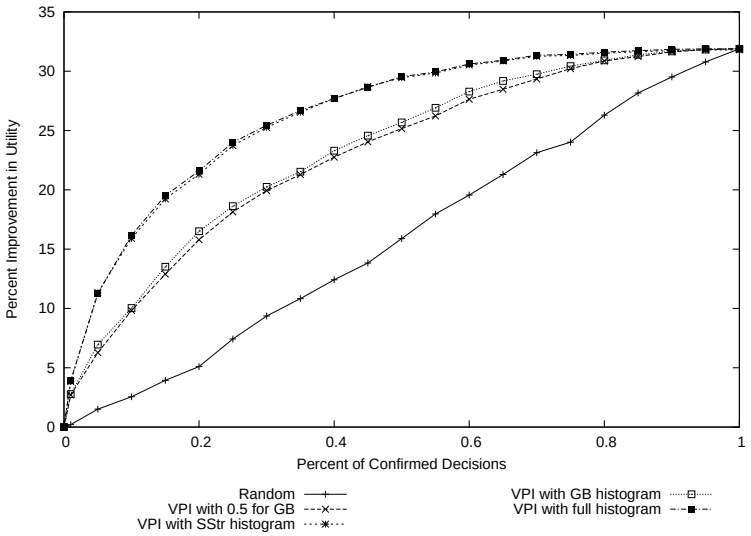


Figure 3.6. The performance of the VPI strategy using different means of assigning probabilities to matches, run over the *Full* dataset.

3.4.2 Synthetic Data Experiments

The experiments above show that the VPI-based strategy is effective in multiple real-world dataspace scenarios derived from Google Base data. In order to investigate the VPI-based strategy in an even wider range of scenarios, I built a dataspace generator capable of creating dataspaces with different characteristics.

Results on Synthetic Datasets

Using the dataspace generator, I ran a series of tests that evaluated the VPI strategy under a variety of dataspace environments. Here I summarize the findings in these experiments.

Basic tests. I generated a dataspace that recreates a realistic large scale data integration environment using realistic values or distributions for different characteristics: e.g., 100000 elements and a zipfian distribution for the query weights and items per

element. In this dataspace, the VPI strategy is able to produce a dataspace whose utility remains within 5% of the utilty produced by the oracle strategy.

Robustness tests. The VPI strategy is robust to variations in the dataspace characteristics: manipulating one characteristic of the dataspace (e.g., matches per element, items per element) while leaving the others at the realistic values used in the basic tests above has very little effect on the efficacy of the VPI strategy. For instance, to test the effect of differing degrees of heterogeneity, I generate dataspaces with different distributions for the number of matches in which an element participates. Regardless of the distribution used in this experiment, the utilities produced by the VPI strategy was within 10% of the percent of potential improvement in utility (as described in the query workload experiment in the previous section). I varied other parameters of the dataspace and ran similar experiments; all experiments produced results similar to the first experiment. In particular, I varied the distributions for the number items in which each element appears, I introduced errors into element cardinality statistics, used different mechanism accuracies, and used different distributions for the query weights (as investigated above with the Google Base datasets).

Parameter exploration tests. By setting all parameters but one to trivial constants (e.g., 1 item per element, 0.5 query weight), we can study the effect that one parameter has on the VPI strategy's effectiveness. We can then determine in what environments it is particularly important to employ an intelligent ordering mechanism for user feedback.

These experiments reveal that in cases where there is a wide range or high skew in the values for a particular parameter, the benefit provided by the VPI strategy is greater: it is able to effectively determine the matches that provide the most benefit and confirm them first.

For instance, when I manipulate the query weight distribution (and set all other

parameters to trivial constants), the percent of potential improvement in the dataspace by the VPI strategy after 10% confirmation in a dataspace with a zipfian query weight distribution is 0.35, whereas with a uniform distribution between 0 and 1 for the query weights, the percent of potential improvement is only 0.19. With smaller ranges in the query weights, the improvement is even less. Note that these results illustrate the potential benefit of employing an intelligent ordering strategy in environments such as Web search where some queries are orders of magnitude more frequent than others. On the other hand, in environments with more homogeneous queries, the selection method is less important.

3.5 Query Answering Using Thresholding

Up to this point in the presentation, I have assumed that a dataspace system employs a query answering model that uses only confirmed matches. Since the goal of a dataspace system is to to provide query access to its underlying data sources even when they have not been fully integrated, it is likely that the system will need to also provide results that are based on matches that are not confirmed, but whose confidence is above a threshold. In this approach, the elements e_1 and e_2 in match $m = (e_1, e_2, c)$ are considered equivalent if the confidence c is greater than a threshold T. The actual value of T depends on systems's tolerance of false negatives versus false positives: the lower the threshold, the more data included in queries, but the higher the potential for erroneous query results.

Here, I analyze the impact of such a query answering model on the VPI-based match scoring mechanism and show that the decision-theoretic framework can be applied with only minor changes to the utility function. We follow a similar process as in Section 3.3 to derive an equation for the value of perfect information for confirming match m_j when the query answering module uses thresholding.

I first need to redefine result quality when thresholding is used for query answering. Here, the query answering module may use an incorrect match if its confidence is above the threshold; thus, some answers in $Q(D)$ may not be correct w.r.t. $Q(D^P)$. To account for these incorrect results as well as the missed results due to correct but unused matches as before, the equation for result quality is altered to consider both precision and recall using F-measure [VR79][2], defined as

$$\frac{2 \cdot precision \cdot recall}{precision + recall}.$$

Precision and recall are defined in this context as follows:

$$precision(Q, D, M) = \frac{|Q(D) \cap Q(D^P)|}{|Q(D)|}$$

$$recall(Q, D, M) = \frac{|Q(D) \cap Q(D^P)|}{|Q(D^P)|}.$$

The result quality of query Q is redefined using F-measure as follows:

$$\begin{aligned} r(Q, D, M) =& \frac{2\frac{|Q(D)\cap Q(D^P)|}{|Q(D)|} \cdot \frac{|Q(D)\cap Q(D^P)|}{|Q(D^P)|}}{\frac{|Q(D)\cap Q(D^P)|}{|Q(D)|} + \frac{|Q(D)\cap Q(D^P)|}{|Q(D^P)|}} \\ =& \frac{2(|Q(D) \cap Q(D^P)|)}{|Q(D)| + |Q(D^P)|}. \end{aligned}$$

Substituting this formula into Equation 3.4, the expected result quality, $Er(Q, D, M)$, when using thresholding becomes:

$$Er(Q, D, M) = \sum_{M_1 \subseteq M} \frac{2(|Q(D) \cap Q(D^{M_1})|)}{|Q(D)| + |Q(D^{M_1})|} Pr(D^{M_1}) \tag{3.9}$$

[2]Here, I use F-measure with precision and recall as equally important, sometimes referred to as F1-measure.

From Section 3.3.1, we already know how to compute $|Q(D)|$ and $|Q(D^{M_1})|$ in this equation. To compute $|Q(D) \cap Q(D^{M_1})|$, first recall that in D^{M_1}, the constant d is deemed equal by the matches in M_1 to a set of other constants in its equivalence class, $\{d_1, \ldots, d_m\}$, which I denote here as $E_d^{M_1}$. Similarly, the matches in M that are above the threshold T determine a set of constants in D that are assumed to be equal to d when computing $Q(D)$, denoted as E_d^M. The set $Q(D) \cap Q(D^{M_1})$ includes the triples that have an element from the intersection of these two equivalence classes in the first position. Therefore, $|Q(D) \cap Q(D^{M_1})| = |D_d^1| + \sum_{d_i \in (E_d^M \cap E_d^{M_1})} |D_{d_i}^1|$.

Since computing Equation 3.9 is prohibitively expensive, $Er(Q, D, M)$ is approximated by employing the same assumption made in Section 3.3.2 where $M = \{m_j\}$. Thus, we rewrite Equation 3.9 with $\{m_j\}$ substituted for M:

$$\begin{aligned} Er'(Q, D, \{m_j\}) = & \frac{2(|Q(D) \cap Q(D_{m_j}^{e_1=e_2})|)}{|Q(D)| + |Q(D_{m_j}^{e_1=e_2})|} c_j + \\ & \frac{2(|Q(D) \cap Q(D_{m_j}^{e_1 \neq e_2})|)}{|Q(D)| + |Q(D_{m_j}^{e_1 \neq e_2})|} (1 - c_j). \end{aligned} \tag{3.10}$$

Finally, following the same logic used to derive Equation 3.8, we have:

$$\begin{aligned} & VPI(m_j) = \\ & \sum_{(Q_i, w_i) \in W_{m_j}} c_j \cdot w_i \left(1 - \frac{2(|Q_i(D) \cap Q_i(D_{m_j}^{e_1=e_2})|)}{|Q_i(D)| + |Q_i(D_{m_j}^{e_1=e_2})|} \right). \end{aligned} \tag{3.11}$$

I implemented this VPI scoring method and experimentally evaluated the ordering it produced when the query answering module uses thresholding. These experiments yielded results very similar to those presented in the previous section, verifying the fact that this framework can be applied to query answering using thresholding. I omit the details of these experiments due to their similarity with the previous section's results.

3.6 Roomba

Having developmed the VPI-based mechanism for ordering candidate matches and shown its effectiveness, I now outline the architecture of Roomba[3], a component of a dataspace system that incorporates the decision-theoretic framework presented in the previous sections to provide efficient, pay-as-you-go match ordering using VPI.

To facilitate a pay-as-you-go mode of interaction, a dataspace system contains a user interaction module that determines the appropriate time to interrupt the user with confirmation request, such as in [HKPH03]. At such a time, this module calls the method $getNextMatch()$ exposed by Roomba that returns the next best match to confirm. The naive approach to supporting such a method call is to order all matches once and then return the next best match from the list on each call to $getNextMatch()$.

In a pay-as-you-go dataspace system, however, $getNextMatch()$ is called over time as the system runs; thus, a particular ordering of matches derived at one point of time using one state of the dataspace may become invalid as the characteristics of the dataspace change. A match's VPI score depends on the queries for which it is relevant, the cardinalities of the elements involved in the match, and the confidence of the match. Furthermore, over time confirmed matches may be fed back to the data integration mechanisms which, as a result, may alter some match predictions.

The key to efficiently supporting $getNextMatch()$ under changing conditions is to limit the number of VPI scores that need to be recomputed when some aspect of the dataspace changes. Here I briefly outline the techniques employed by Roomba to efficiently compute the next best match as the characteristics of the dataspace change.

[3]The name "Roomba" alludes to the vacuuming robot of the same name [iRo07]. Just as the robot discovers what needs to be cleaned your room, the Roomba system aids a dataspace system in determining what needs to be cleaned in the dataspace.

Roomba operates in three phases: initialization, update monitoring, and $getNextMatch()$.

Initialization: The initialization phase creates all the data structures used by Roomba and produces an initial ordering of matches. At this point, Roomba also calculates the element cardinality statistics over the dataspace. In order to facilitate efficient VPI recomputation, Roomba builds indexes that map from each aspect of the data that factors into the VPI calculation to matches that would be affected by a change in that data. Finally Roomba calculates the initial VPI score for each match as defined in Equation 3.3 and stores them in an ordered list. Note that these VPI computations can be done in parallel.

Update Monitoring: While the system runs, the dataspace's conditions will continuously change, potentially causing an invalidation of the ordering derived during initialization. When such a change occurs, a $ChangeMonitor$ notes the type of change (i.e., element cardinality, query workload, or match confidence) and utilizes the indexes built during the initialization phase to find the matches that are affected by the particular change. Only these matches are flagged for recomputation in a recalculation list to be processed on the next call to $getNextMatch()$.

$getNextMatch()$**:** On a call to $getNextMatch()$, Roomba sends the recalculation list to the VPI calculator to recompute and reorder any matches whose VPI score may have changed. Note that here, too, the VPI calculations can be done in parallel. Roomba then returns the top match off the list for user confirmation.

The overall architecture of Roomba is shown in Figure 3.7. The storage engine of Roomba stores and indexes elements, matches produced by different mechanisms, and the query workload. The ChangeMonitor watches elements, matches, and queries for any changes, at which point it sends affected matched to the recalculation list as

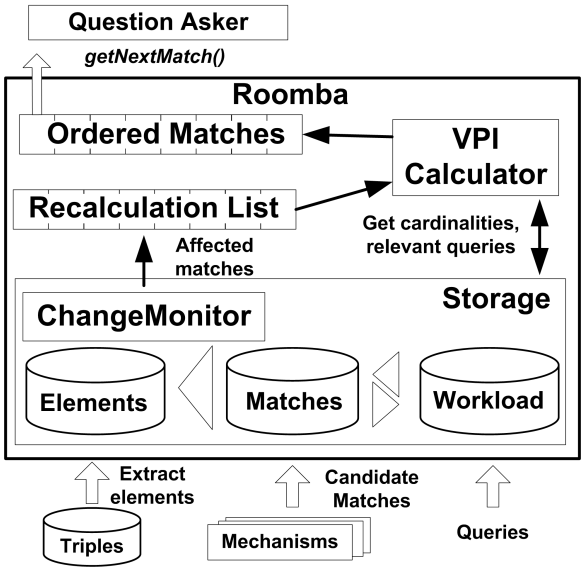


Figure 3.7. Roomba architecture.

described above. The core of Roomba is the VPI calculator which uses Equation 3.8 to estimate the benefit of confirming each match. These matches are ordered by estimated benefit and stored in the Ordered Matches list. When the Question Asker (as described in Section 3.2) requests a match, Roomba returns the top match from this list.

3.7 Related Work

While the decision-theoretic framework is based on formalisms used in the AI community [RN03], decision theory and the value of perfect information are well-known concepts in many fields such as economics [MVN44, MCWG95] and health care [CS03]. Within the data management community, there has been work on applying expected utility to query optimization [CHS99].

Previous work on soliciting user feedback in data integration systems has focused on the output of a single mechanism. The work in [DDH01] and [WYDM04] addresses

incorporating user feedback into schema matching tasks. Similarly, [SB02] introduces an active-learning based approach to entity resolution that requests user feedback to help train classifiers. These approaches are closely tied to a single type of data integration task (e.g., schema matching or entity resolution) and cannot be easily applied to other tasks. Furthermore, their overall goal is to reduce the uncertainty in the produced matches without regard to how important those matches are to queries in the dataspace. Rather than reasoning about user feedback for each mechanism separately, a primary benefit of the VPI-based framework presented here is that it treats multiple mechanisms uniformly and judiciously balances between them with the goal of providing better query results for the dataspace.

The MOBS [MDV+03] approach to building data integration systems outlines a framework for learning the correctness of matches by soliciting feedback from many users and combining the responses to converge to the correct answer. While MOBS handles the result of match confirmations, it does not address how to select which match to pose to the user in the first place. Thus, Roomba naturally fits within this framework by providing the most beneficial match for MOBS to confirm.

3.8 Chapter Summary

This chapter presented a decision-theoretic approach to ordering user feedback in a dataspace. As part of this framework, I developed a utility function that captures the usefulness of a given dataspace state in terms of query result quality. I then presented a means of selecting matches for confirmation based on their value of perfect information: the expected increase in dataspace utility upon requesting user feedback for the match. Importantly, this framework enables reasoning about the benefit of confirming matches from multiple data integration mechanisms in a uniform fashion. I described a set of experiments on real and synthetic datasets that validated the ben-

efits of this approach. Finally, I outlined Roomba, an architecture for incorporating this VPI-based approach in a dataspace system.

While this work provides a foundation for guiding user feedback in a dataspace system, there are many interesting directions for future work. First of all, Roomba could be extended to deal with *imperfect* user feedback. Here, I assumed that users answered correctly every time: a confirmation meant that the match was correct in reality. Users, however, are human and may not always be correct: matches may be ambiguous or challenging to answer correctly, or users may be malicious. To cope with uncertainty in user feedback, the dataspace could ask the same confirmation of multiple users and employ a majority voting scheme. More advanced approaches involve modeling user responses as probabilistically related to the true answer of the match and then adjusting the confidence of a match on confirmation [RN03].

Another area of future work is to explore other types of user feedback. In this chapter, I explored how to efficiently involve users in resolving uncertainty through *explicit* user feedback. A dataspace system can also leverage the wealth of research on *implicit* feedback (e.g., [JGP+05, CLWB01, RJ05]) to improve the certainty of candidate matches. For instance, the click-through rate of query results supply an indicator of the correctness of the matches employed during query answering: a click on a particular result may indicate that the matches used to compute that result are correct, causing the dataspace system to increase the confidence of those matches. A system can also use information from subsequent queries, or query chains [RJ05], to reason about the correctness of matches not employed during query processing. If, for instance, a user searches for "red" and then subsequently searches for "cardinal", then the system can increase the confidence of the candidate match ("red", "cardinal", c).

While this chapter focused on entity resolution and schema matching, Roomba can be used with many other data integration mechanisms, such as information extraction

(e.g., [CDYR08]). As such, Roomba provides a powerful tool for applications such as Total Consumer Awareness that rely on cleaned and integrated dataspace data.

Chapter 4

Metaphysical Data Independence

Roomba provides pay-as-you-go support for data integration challenges that arise when dealing with traditional relational data or data on the Web; dataspace applications such as TCA, however, depend on data collected from the physical world through sensor devices as well. Unfortunately, there is a wide gulf, that I term the *physical-digital divide*, between the raw data produced by sensors and the high-level needs of dataspace applications. In particular, sensor data is unreliable, semantically low-level, and rapidly changing. In this chapter, I define the physical-digital divide and illustrate its challenges through multiple real-world studies. I then present *Metaphysical Data Independence* (*MDI*), an interface for exposing sensor data that shields applications from the challenges that arise when dealing with the physical-digital divide.[1]

[1]Much of the work presented in this chapter has been published in [JFG08].

4.1 Introduction

With the widespread deployment of physical sensing devices such as wireless sensor networks and RFID technology, the physical world is being brought ever closer to the digital world: RFID provides enterprises with up-to-the-second information on their supply chains [EPC05b]; wireless sensor networks enable unprecedented visibility into environmental and structural processes [TPS+05]; and ubiquitous computing technology is changing the way we interact with our surroundings [AM00]. Many dataspace applications rely on this data to provide new and powerful services. For instance, the Total Consumer Awareness application presented in Chapter 1 uses sensors to provide better visibility into supply chains to consumers.

One of the main challenges in building sensor-based dataspace applications such as TCA is the wide gulf between the data produced by devices in the physical world and the needs and requirements of these applications in the digital world. I term this rift the *physical-digital divide*.

The crux of the problem is that the raw data provided by a set of sensor devices do not adequately represent the aspects of physical world being monitored. Rather, the data are noisy due to outliers and mis-calibrated sensors, incomplete as a result of dropped messages, and coarse-grained in both time and space as devices necessarily sample periodically and cannot be deployed with complete coverage. Furthermore, in many cases an application may require data for which no sensing device exists (e.g., [CKZ+05, REA99]). For instance, a user of a TCA application may interested in the environmental impact of manufacturing a given product (e.g., the amount of greenhouse gases produced during manufacturing). There is no device that provides exactly this data; rather, it must be derived using multiple sensor streams as well as stored data.

As such, sensor-based deployments typically include complex logic to map from

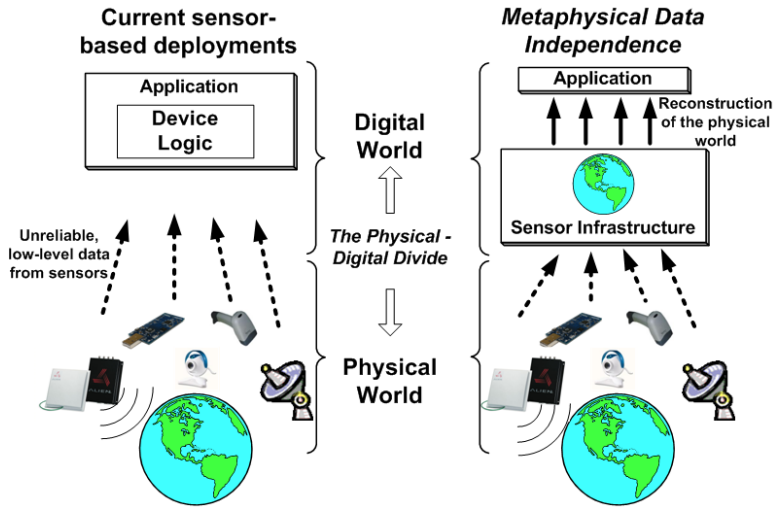


Figure 4.1. Today's sensor-based applications are tightly-coupled with the underlying sensors and thus complex and hard to manage. Metaphysical Data Independence allows applications to interact with a reconstruction of the physical world in the digital world, greatly simplifying application deployment.

low-level, dirty sensor data to high-level application concepts and maintain this mapping over time [BGH+05, TPS+05]. Any errors or fluctuations in the underlying devices must be handled directly by the application. As a result, current sensor-based applications tend to be complex, brittle, and hard to evolve.

To address this problem, I introduce a new layer of data independence, *Metaphysical Data Independence*[2](*MDI*), that shields sensor-based applications from the challenges associated with managing and accessing physical sensor devices. Just as traditional RDBMSs use *physical* and *logical* data independence to hide the complexity and changes of the physical data storage and base schema, respectively, MDI hides the complexity of the physical sensing devices from sensor-based applications.

The key philosophy behind Metaphysical Data Independence is that sensor data should be abstracted as data about the physical world; that is, applications should

[2]The term "metaphysical" is a loose reference to Plato's *Divided Line* philosophy concerning the real world and the perceived world [Pla]. He conjectured that the physical world is only accessible through a person's imperfect senses and thus knowledge and reason must be used to guide any perception of the real world.

interact with a reconstruction of the physical world in the digital world, as if the physical-digital divide did not exist (as illustrated in Figure 4.1).

To represent this reconstruction, we need to model the physical world at a level appropriate for use across many applications. The challenge in defining this model is to balance the tension between hiding the complexity of sensor data and providing data at a semantic level that is appropriate for a large class of applications.

To this end, I outline a data model for sensor-based applications based on the concepts of *objects*, *attributes*, and *uncertainty estimates*. Objects in this model correspond to real-world entities such as people, products, and rooms. Each object has a set of associated attributes such as location, temperature, and velocity. Due to the uncertainty of sensor data, an integral part of this model is the use of uncertainty estimates. The set of all objects, their attributes, and associated uncertainty estimates at a given epoch make up the "state of the world" in that epoch.

The fundamental aspect of this model and interface is that they expose as little information about the underlying physical devices as possible. Just as an application interacting with a traditional relational database using SQL does not have to know anything about the underlying storage mechanisms, disk usage, or data format, an application using MDI can be oblivious to details of the devices on which it is deployed.

The remainder of this chapter is organized as follows. In Section 4.2, I define the physical-digital divide. I then outline MDI in Section 4.3.

4.2 The Physical-Digital Divide

The physical-digital divide is the disconnect between the data produced by devices monitoring the physical world and data expected by data processing infrastructures in the digital world. I illustrate the physical-digital divide using examples from two of the

more common types of sensing devices, RFID readers and wireless sensor networks. While each type of sensing device presents specific data processing challenges, there are many issues that are common across devices. In this section, I highlight four main factors that contribute to the physical-digital divide.

4.2.1 Sensor Data Unreliability

Sensors often employ low cost, low power hardware and wireless communication, which lead to frequently dropped messages. For example, the observed *read rate*, or percentage of tags in a reader's vicinity that are actually reported, in real-world RFID deployments is often in the $60 - 70\%$ range [JAF+06a, Lau05]; in other words, over 30% of the tag readings are routinely dropped. Wireless sensors also demonstrate similar errors. For instance, in a wireless sensor network experiment at the Intel Research Lab in Berkeley, each sensor delivered, on average, only 42% of the data it was asked to report [Lab05].

Not only are readings frequently dropped, but often individual sensor readings are unreliable. In a sensor network deployment in Sonoma County, CA, for example, 8 out of 33 temperature-sensing motes failed, but continued to report readings that slowly rose to above 100^o Celsius [Son06].

Thus, the data produced by sensors must be appropriately cleaned to compensate for these failures before they can be used by any application. Incorporating cleaning logic in applications greatly increases the complexity of the application.

4.2.2 Granularity Mismatch

Sensor-based applications tend to have specific notions of time and space that usually do not correspond to the sensing granularity of the underlying devices [DM06, JAF+06b].

Temporally, the actual sensing granularity of the devices may be coarser than an application desires due to power or bandwidth limitations as in the case of wireless sensors, or it may finer such as with the high sample rate of RFID readers.

Similarly, the spatial sensing granularity of devices may not match an application's notion of space. One device may monitor multiple application-level spatial units, such as rooms or shelves. Conversely, there may be multiple devices that monitor the same spatial unit. For instance, RFID deployments usually deploy multiple readers in close proximity to ensure full coverage of the area of interest.

This mismatch in sensing granularity potentially causes semantic errors. For instance, redundant readings in space could lead to an application seeing an object in two places at the same time.

4.2.3 The Semantic Gap

Usually, sensor-based applications view the world as high-level concepts; sensing devices, on the other hand, produce low-level data that often has little meaning to the application.

In many cases, applications are interested in data for which no physical sensing device exists. For instance, TCA monitors data about products on a supply chain. Other common examples of such data include people in pervasive applications [AM00] and attributes of manufactured goods in industrial processes [REA99]. These high-

level application concepts must be derived through the combination of data from multiple devices as well as other sources of data.

Translating from low-level device readings to application-level concepts involves intimate knowledge of the environment, devices, and the data they produce, thus complicating application development and deployment.

4.2.4 Variability in Sensor Deployments

Not only do sensor data exhibit the issues described above, but the nature of these issues changes over time and from deployment to deployment.

Wireless sensor motes, for instance, lose accuracy as their batteries wear [TPS+05]. RFID readers produce different quality data and their detection fields vary depending on the environment in which they are deployed [FJPR04]. For example, during a series of RFID reader tests in a variety of environments, the quality of readings from readers in two different rooms, next door to each other, varied greatly. (See Section 5.2 for a detailed description of this experiment.)

As a result of this variability, sensor deployments that are not designed to be adjusted to varied environments tend to produce erroneous data as conditions change.

4.3 Metaphysical Data Independence

Motivated by the complexities associated with the physical-digital divide, I propose a new layer of data independence, *Metaphysical Data Independence* (*MDI*). MDI defines a separation of concerns between the application logic and the logic needed to access and manage the data from physical devices. The key philosophical statement behind MDI is that the specifics of the underlying devices are abstracted away behind

a model of the physical world that represents a reconstruction of this world in the digital world.

4.3.1 Object Model

The model for supporting MDI consists of objects, attributes, and uncertainty estimates.

Objects: At the core of this model is the concept of an object. An object is loosely defined as any entity in the real world to which an application may uniquely refer. For instance, a digital home may have people objects, room objects, and pet objects. The actual objects defined for each deployment may vary dramatically, but experience shows that within an application the appropriate objects are evident (e.g., [Mar06, SSW+05]).

Attributes: Associated with each object is a set of attributes that describe the state of that object. Attributes are attached directly to the object, and in general have little meaning without this association (e.g., temperature means nothing unless it is the temperature of some object). The primary attributes of any real-world object are time and space (location); these attributes are typically included for all objects. Examples of other attributes include temperature and velocity. An attribute may also be a complex multi-valued field instead of a single value. For instance, some attributes may describe the contour of a value in a certain area [GBT+04].

While an attribute is dependent on the object that it describes, it is independent of any physical sensing device. There may be multiple devices that observe the same value. For instance, wireless motes and some RFID tags [SSP+06] both sense light levels. Soft sensors [Qin96], on the other hand, may be used to combine data from multiple types of sensors to derive an attribute. Alternatively, no physical device may be required at all to sense a particular attribute; model-based sensing [DGM+04],

cached values, or archived data may be used. In any case, such acquisition and processing is hidden by the model.

Uncertainty: Due to the limitations of sensing devices, an estimate of the uncertainty in the reported data is essential. Thus, throughout this model is the notion of uncertainty. Uncertainty serves as the unifying means by which device-dependent processing can be hidden.

Uncertainty estimates appear at multiple levels. Attached to each object is a confidence of existence, as described in the previous chapter. Additionally, each attribute has some uncertainty value(s) with which it is associated. The exact representation of these values are dependent on the type of attribute. For instance, the uncertainty of a particular value for a temperature attribute may be represented as a range and confidence [DGM+04], while uncertainty in location could be described with a spatial probability distribution function. Regardless of the description method, uncertainty is expressed in a device-independent manner.

The set of all objects, their attributes, and uncertainty at a given time-step make up the "state of the world" at that time-step.

4.3.2 Interface

Data expressed in this model can be accessed as a stream of object states, ideally through declarative means using languages such as CQL [ABW06] or those defined as part of the HiFi [RJK+05] or SASE [WDR06] projects.

Sensor-based applications have differing needs in terms of data; the interface should allow applications to specify selection predicates on which objects and attributes are necessary. Given these predicates, the sensor infrastructure can optimize access to these data. Similarly, since different applications have different data quality

requirements, the interface needs to allow applications to request the desired level of data quality through uncertainty predicates. Such predicates can help guide the infrastructure on which devices to use and by which means to produce the requested data. Further, some sensor data processing is inherently application-specific. Thus, the interface should allow user-defined code to be pushed down into the sensor infrastructure to assist in data processing.

As there are potentially many applications accessing data from the same set of devices, but with different requirements, the sensor infrastructure should support multiple streams of MDI data. Generating this output should be done in such a way as to minimize access to the devices themselves and to maximize sharing across streams [Kri06]. Further, the interface should provide seamless access to present, past, and future MDI data. That is, access to streaming data (present), archival data (past), and future predictions based on models should all happen through the same interface in a unified manner.

Finally, note that while there will always be uses of sensor devices that demand low-level access to the data and devices, such as advanced scientific monitoring applications or tight-loop sensor-actuator systems, there is a large class of emerging sensor-based applications that would benefit from the higher level of abstraction with which to interact with sensor data that MDI provides.

4.4 Chapter Summary

To deal with the challenges associated with the physical-digital divide, dataspace applications must be shielded from any aspect of the underlying devices. In this chapter, I outlined Metaphysical Data Independence, a philosophy for building

sensor infrastructures that hides all aspects of the physical-digital divide and allows applications to use sensor data oblivious to the challenges in producing the data.

Metaphysical data independence provides a powerful basis on which to build pay-as-you-go support for dataspace systems that integrate sensor data. As a result, applications such as TCA can seamlessly incorporate sensor data streams.

In the following Chapters, I develop two techniques for providing MDI. In Chapter 5, I describe a technique for alleviating many of challenges associated with RFID utilizing statistical mechanisms. In Chapter 6, I present a framework for building data processing infrastructures that produce MDI data.

Chapter 5

Adaptive Cleaning of RFID Data

MDI protects applications from the complexities of dealing with low-level sensor data; to instantiate this interface, a dataspace system must deal with unreliable and rapidly changing sensor data. While Roomba provides such services in dataspace environments where humans can be put in the loop, when dealing with unreliable sensor data that are streaming in real-time, these processes must be automated. To this end, in this chapter I develop *SMURF* (*Statistical sMoothing of Unreliable RFid data*), a cleaning tool that alleviates issues associated with using RFID data through adaptive techniques based on a novel statistical framework.[1] I begin with a detailed study on the characteristics of RFID data. I then detail SMURF's statistical framework and present a detailed experimental study demonstrating its effectiveness. Finally, I discuss MDI-SMURF, an architecture that utilizes SMURF's statistical framework to provide Metaphysical Data Independence for RFID-based applications

[1] Much of the work presented in this chapter has been published in [JGF06, JFG08].

5.1 Introduction

RFID (Radio Frequency IDentification) technology promises revolutions in areas such as supply chain management and ubiquitous computing enabled by pervasive, low-cost sensing and identification. A primary factor limiting the widespread adoption of RFID technology is the *unreliability* of the data streams produced by RFID readers [Lau05]. The observed read rate (i.e., percentage of tags in a reader's vicinity that are actually reported) in real-world RFID deployments is often in the 60-70% range (see Section 5.2); in other words, over 30% of the tag readings are routinely dropped.

The standard data-cleaning mechanism for RFID data is a *temporal "smoothing filter"*: a sliding window over the reader's data stream that interpolates for lost readings from each tag within the time window [GS04, iAn06]. The goal is to reduce or eliminate dropped readings by giving each tag more opportunities to be read within the smoothing window. While the implementation of such filters vary, a smoothing filter's functionality can be expressed as a simplified continuous query (e.g., in CQL [ABW06]) as shown in Query 1 (for a 5 second window). This query states that if the tag appears at least once in the window, it is considered to be present for the entire window.

Query 1 *CQL Smoothing Filter to Correct for Dropped Readings.*

```
SELECT    distinct tag_id
FROM      rfid_readings_stream  [RANGE '5 sec']
GROUP BY  tag_id
```

Static Window Smoothing. Typically, a smoothing filter requires the application to fix the window size (as in the above CQL statement). Setting the window size, however, is a non-trivial task: the ideal smoothing-window size needs to carefully balance two opposing application requirements (as shown in Figure 5.1): *ensuring*

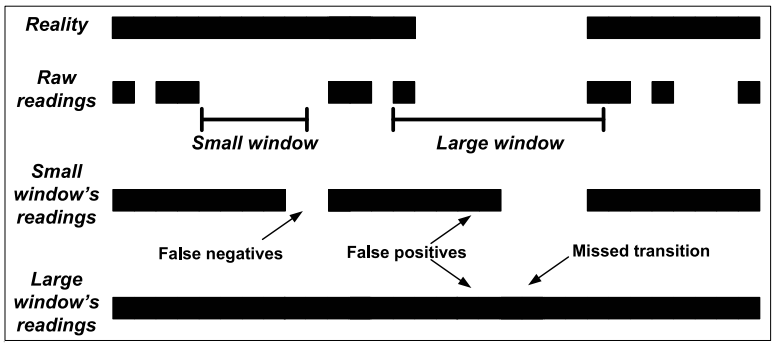


Figure 5.1. Tension in setting the smoothing-window size for tracking a single tag (dark bars indicate the tag is present/read): small windows fail to fill in dropped readings (false negatives); large windows fail to capture tag movement (false positives).

completeness for the set of tag readings (due to reader unreliability) and *capturing tag dynamics* (due to tag movements in and out of the reader's detection field).

– *Completeness:* To ensure that all tags in the reader's detection range are read, the smoothing window must be large enough to correct for reader unreliability. Small window sizes cause readings for some tags to be lost, leading to *false negatives* (i.e., tags mistakenly assumed to have exited the reader's detection range) and, consequently, a large underestimation bias (e.g., always under-counting the tag population). Adjusting the window size for completeness depends on the reader's read rate, which, in turn, depends on both the type of reader and tag as well as the physical surroundings [Dan05, FJPR04].

– *Tag dynamics:* Using a large smoothing window, on the other hand, risks not accurately detecting tag movements within the window, leading to *false positives* (i.e., tags mistakenly assumed to be present after they have exited the reader's detection range).

Adjusting the window size for tag dynamics depends on the movement characteristics of the tags, which, in turn, can vary significantly depending on the application;

for instance, a tag motionless on a shelf exhibits a different movement pattern from a tag on a conveyor belt.

As a result, a considerable challenge to deploying RFID-based applications is ascertaining the characteristics of the environment and configuring the filter to take into account the above factors. Furthermore, no single window size is expected to be effective over the lifetime of a deployment as both the reader reliability and tag behavior may vary dynamically; thus, either the window size must be repeatedly reconfigured, or the quality of the data suffers.

A second major problem with fixed-window smoothing techniques is the use of a single window size for all tags in a deployment. Different subsets of tagged objects may behave very differently from others. For instance in a warehouse environment, some tagged items may be placed on a shelf while others are moved on forklifts. The best smoothing window size for each of these groups of tags is potentially different.

Adaptive Windowing for MDI. The key to masking the unreliability of RFID data in support of Metaphysical Data Independence is to hide the window size from the application and instead automatically determine the window size initially and then adapt it as the system runs. To this end, I have developed *SMURF* (*Statistical sMoothing of Unreliable RFid data*), an adaptive smoothing filter that does not require the application to set the window size; instead, it determines the window size automatically and continuously adapts it over the lifetime of the system based on observed readings.

The main challenge for an adaptive smoothing scheme is to distinguish between periods of dropped readings and periods when a tag has moved. To address this problem, SMURF uses a statistical sampling-based approach. One of the key ideas behind SMURF's adaptive algorithms is that RFID data streams can be modeled as a *random sample* of the tags in a reader's detection range. Through this sample-based

view of observed RFID readings, SMURF employs algorithms grounded in statistical sampling theory to drive its adaptive smoothing techniques. Furthermore, SMURF adapts its smoothing-window sizes at a *much finer granularity* compared to traditional smoothing filters that fix a single window size for the entire tag population.

The remainder of this chapter is organized as follows. I first provide a deeper background on the unreliable nature of RFID in Section 5.2. With this background, I detail SMURF's core statistical framework and adaptive smoothing filter in Section 5.3. Section 5.4 presents an experimental study of SMURF's effectiveness. In Section 5.5, I discuss how SMURF is used to instantiate MDI through the MDI-SMURF architecture.

5.2 RFID Technology Challenges

In this section give a brief background on the challenges facing current RFID deployments.

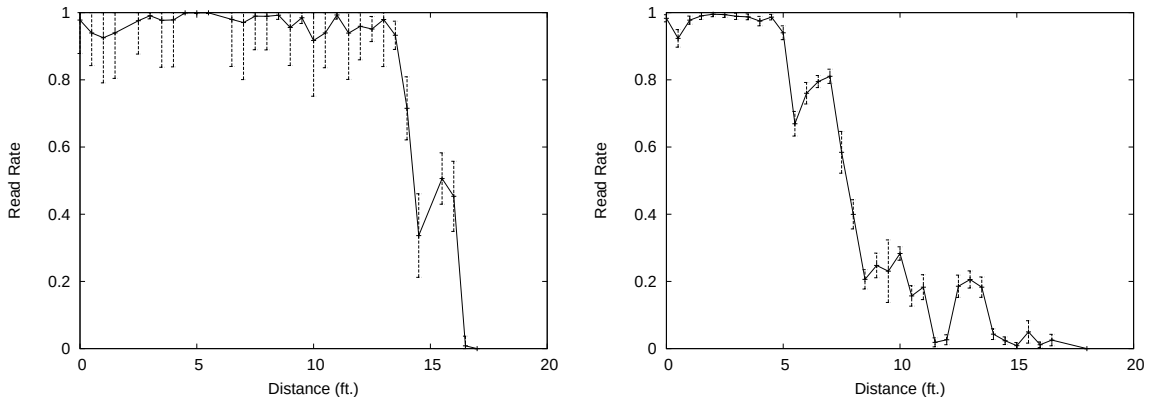
RFID Reader and Tag Performance.

To better understand the properties of RFID readings, I profile two RFID readers with different tags in two environments. The profiling methodology is as follows. I suspend a single tag at varying distances in the same plane as the antenna. For every 6-inch increment of distance from the reader, I measure the read rate (number of responses to number of interrogations) for 100 epochs.

These profiling experiments use two types of readers, the Alien ALR-9780 [ALR05] and the Sensormatic Agile 2 [Sen06], with three types of tags (Alien "I2", "M", and "Squiggle" [Ali05a]). I test various combinations of these readers and tags in two environments. The first environment, a large, wide-open room with little metal present, represents a controlled environment for RFID technology: I eliminate many

of the causes of degraded read rates [FJPR04]. The second profiling environment, a lab with metal objects such as desks and computer equipment, represents a noisy environment.

Figure 5.2 depicts the results from two different profiling experiments showing the read rate of the tag at distances ranging from 0 to 20 feet. The plots shown here are representative of the 8 different profiles I collected; all other experiments yielded a curve similar in shape to one of these two plots.



(a) Alien reader with Alien Squiggle tag in a controlled environment.

(b) Sensormatic reader with Alien I2 tag in a noisy environment.

Figure 5.2. Read rate of a single tag at varying distances from the reader in different environments. Error bars represent ± one standard deviation.

All of the profiles I collected have similar properties despite being generated using different readers, tags, and environments. First, the overall detection range of all readers and tags profiled remains relatively constant at 15-20 feet. Second, within each reader's detection range, there are two distinct regions: (1) The area directly in front of the reader, termed the reader's *major detection region* [HBF+04], giving high detection probabilities (read rates at or above 95%); and, (2) the reader's *minor detection region*, extending from the end of the major detection region to the edge of the reader's full detection range, where the read rate fluctuates as it drops to zero at the end of the detection range.

The main difference between the observed profiles lies in the percentage of the reader's detection range corresponding to its major detection region. For instance, the major detection region corresponds to roughly 75% of the full detection range for the profile in Figure 5.2(a), whereas it makes up only 25% of the range in the profile in Figure 5.2(b). All of the experiments showed similar behavior. Note that these findings are consistent with the results of in-depth commercial studies of the performance of many different tags and readers under highly-controlled conditions [Dea04].

I also test the readers to determine how they respond to the presence of *multiple tags* in their detection ranges. For these tests, I suspend 10 tags in the same plane as the reader and measure the average read rate for 100 epochs at varying distances from the reader. While the overall properties of the observed profile are similar to the single tag case (there is still a separation between a major and minor detection region), the read rate in the major detection region typically drops to around 80%. Additional tests show that the read rate in the major detection region stays somewhat constant with increasing numbers of tags, at least up to 25 tags in the reader's detection range.

In the remainder of this chapter, I use these observations in the design of MDI-SMURF's cleaning mechanisms and in the implementation of an RFID data generator for evaluating these techniques.

5.3 RFID Data Cleaning with SMURF

In this section I present the adaptive smoothing filter SMURF uses to deal with the unreliabilities of RFID data.

5.3.1 RFID Data: A Statistical Sampling Perspective

SMURF captures tag dynamics while compensating for lost RFID readings in a principled, statistical manner. The key idea is that the observed RFID readings can be viewed as a *random sample* of the population of tags in the physical world.

Consider an epoch t. Recall from the RFID background in the previous chapter that an epoch is a reader's unit of detection. Epochs represent the basic time units, many of which can be combined to make up a smoothing window [GS04, iAn06]. Without loss of generality, let N_t denote the (unknown) size of the underlying tag population at epoch t, and let $S_t \subseteq \{1, \ldots, N_t\}$ denote the subset of tags observed ("sampled") during that epoch. SMURF views S_t as an *unequal probability random sample* of the tag population.

SMURF uses a *per-epoch sampling probability* $p_{i,t}$ for each tag that represents the probability that tag i is detected in epoch t. While there are many possible means of deriving this value, SMURF utilizes the response-count information stored in the reader's tag list (Table 2.1). Specifically, for each tag $i \in S_t$, SMURF employs the response-count information for tag i in conjunction with the known number of interrogation cycles per epoch to derive $p_{i,t}$. This sampling probability $p_{i,t}$ is empirically estimated as the observed read rate for tag i during that epoch; for instance, assuming a reader configuration of 10 interrogation cycles per epoch, the sampling probabilities for the first and second tags in Table 2.1 would be $p_{x78,t} = 0.9$ and $p_{x57,t} = 0.1$, respectively. Of course, these sampling probabilities differ across tags and can also vary over time as the observed tags move within reader's detection range.

The key insight of viewing each RFID epoch as a "sampling trial" enables SMURF's novel, statistical-driven perspective on adaptive RFID data cleaning. In a nutshell, SMURF views the observed readings within a smoothing window as the result of repeated random-sampling trials, and employs techniques and estimators

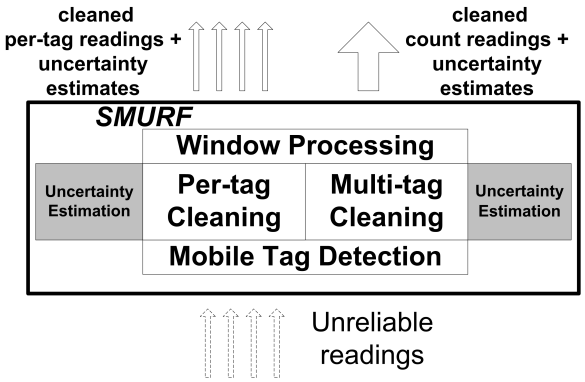


Figure 5.3. The internal architecture of SMURF's adaptive smoothing filter.

grounded in statistical sampling theory to reason about the underlying physical-world phenomena and drive its adaptive RFID data cleaning algorithms.

More specifically, SMURF uses the statistical properties of the observed random sample to appropriately adapt the size of its smoothing window based on (1) completeness requirements, and (2) signal transitions detected as *statistically-significant changes* in the underlying tag readings. Further, even for window sizes that are necessarily small (to capture fast-varying signals), SMURF uses *sampling-based estimators* [Coc77, SSW92] to provide accurate, *unbiased* estimates for tag-population aggregates (e.g., counts), and thus avoids the systematic under-counting of conventional smoothing techniques. Thus, SMURF's sampling-based foundation enables it to explore the tension between completeness and tag dynamics in a principled, statistical manner that continuously adapts the smoothing strategy based on statistical properties of the data to provide accurate, unbiased data to applications.

5.3.2 SMURF's Adaptive Smoothing Filter

The overall architecture of SMURF's adaptive smoothing filter is presented in Figure 5.3. This filter contains two primary cleaning mechanisms aimed at (1) producing accurate data streams for individual tag ID readings (*per-tag cleaning*); and

(2) providing accurate aggregate (e.g., count) estimates over large tag populations (*multi-tag cleaning*). Additionally, SMURF incorporates two modules that apply to both data-cleaning techniques: a sliding-window processor for fine-grained RFID data smoothing, and an optimization mechanism for improving cleaning effectiveness by detecting *mobile tags*. Finally, SMURF contains shadow modules for both per-tag and multi-tag smoothing to calculate uncertainty estimates.

Sliding Window Processing. As with any window-based cleaning scheme, SMURF produces an output reading for a given tag ID if there exists at least one reading for that tag within the smoothing window [GS04, iAn06]. SMURF's sliding-window processor implements two basic modifications to conventional RFID smoothing filters: (1) partitioned RFID smoothing, and (2) epoch-based mid-window slide.

To handle subsets of tagged objects that behave differently from others, SMURF's cleaning techniques adapt the smoothing-window size at a much finer granularity than traditional smoothing mechanisms. At one extreme, when tracking individual tag movements, SMURF runs its adaptive sliding-window processing *per tag ID*. In general, the granularity of SMURF's windowing mechanisms is determined by the *aggregate query of interest*. That is, by a pair `(subset, aggregate)` determining the `subset` of tags over which the `aggregate` value (e.g., count) is monitored. Note that such fine-grained processing can be expressed in a declarative fashion, such as through the `Partition By` clause in CQL.

As epochs are a sample cycle in SMURF's sampling-based model of RFID data, SMURF slides its windows by a single epoch (as opposed to a time period or by tuples). Furthermore, SMURF produces readings with a timestamp corresponding to the midpoint of the window after the entire window has been seen. This behavior captures the intuitive notion of smoothing: e.g., if there are reported readings at times

$t-1$ and $t+1$, then there is likely a reading at time t. I experimentally validated that this approach yields the most reliable readings.

5.3.3 Adaptive Per-Tag Cleaning

To clean readings from a single tag, the fundamental challenge is to distinguish between periods of dropped readings and periods where the tag has actually left the reader's detection field. SMURF must set its window size such that it provides completeness for periods of dropped readings as well as accurately captures transitions for periods where the tag has left. To help differentiate between these two behaviors and to guide subsequent window adaptations, SMURF employs statistical mechanisms based on its random-sample view of RFID data.

A Binomial Sampling Model for Single Tag Readings. Consider the simple case of cleaning the readings from a single tag (say, i) based on a reader's observations over a smoothing window of size w_i epochs (say, $W_i = (t - w_i, t]$). Assume, for the time being, that tag i is present in the reader's range throughout the window W_i, and has the same probability, p_i, of being observed in each epoch of W_i. SMURF views each epoch as an independent *Bernoulli trial* (i.e., a sampling draw for tag i) with success probability p_i. This, in turn, implies that the number of successful observations of tag i in the window is a random variable that follows a *binomial distribution* with parameters (w_i, p_i) (i.e., $B(w_i, p_i)$). In the general case, assume that tag i is seen in only a subset $S_i \subseteq W_i$ of all the epochs in W_i, and let p_i^{avg} denote the average empirical read rate over these observation epochs; that is, $p_i^{avg} = \sum_{t \in S_i} p_{i,t}/|S_i|$, where each $p_{i,t}$ is calculated based on the reader's tag list information as shown in Section 5.3.1. Note that it is assumed that within an appropriately-sized window, the $p_{i,t}$s will be relatively homogeneous and thus averaging is a valid estimate

of the actual $p_{i,t}$.[2] Based on the discussion above, and under the assumption that the tag stays within the reader's detection field throughout W_i, S_i can be viewed as a *binomial sample* (of epochs in W_i) and $|S_i|$ as a $B(w_i, p_i^{avg})$ binomial random variable; thus, from standard probability theory, the *expectation* and *variance* of $|S_i|$ is expressed as:

$$\mathrm{E}[|S_i|] = w_i p_i^{avg} \quad \text{and} \quad \mathrm{Var}[|S_i|] = w_i p_i^{avg}(1 - p_i^{avg}).$$

Next, I discuss how SMURF employs this binomial sampling model to adjust its smoothing window for per-tag cleaning and accurately detect transitions (e.g., departures of tag i).

Per-Tag Adaptive Window Size Adjustment. With the binomial sampling model in place, I first consider the problem of setting SMURF's window size w_i to guarantee *completeness*. In other words, we want to ensure that there are enough epochs in W_i such that tag i is observed if it exists within the reader's range. Given the statistical nature of the model, these guarantees are necessarily probabilistic; that is, w_i can be set to ensure that tag i is read with high probability, as described in the following lemma.

Lemma 1 Let p_i^{avg} denote the observation probability for tag i during an epoch. Then, setting the number of epochs within the smoothing window to be $w_i \geq \lceil \frac{\ln(1/\delta)}{p_i^{avg}} \rceil$ ensures that tag i is observed within W_i with probability $> 1 - \delta$. ∎

Proof: Based on the model of independent Bernoulli trials for observing tag i, the probability that we miss a reading from tag i over w_i sampling trials is exactly $(1 - p_i^{avg})^{w_i}$. Setting this probability $\leq \delta$ and taking logs gives $w_i \ln(1 - p_i^{avg}) \leq \ln \delta$. Combining this result with the inequality $-x \geq \ln(1-x)$ for $x \in (0, 1)$, we see that it

[2] In cases where this homogeneity assumption does not hold due to a tag moving rapidly away from the reader, the mobile tag detection algorithm (Section 5.3.5) allows SMURF to appropriately size its window to capture tag dynamics.

suffices to require that $-w_i p_i^{avg} \leq \ln \delta$, or, equivalently, $w_i \geq \frac{\ln(1/\delta)}{p_i^{avg}}$. This completes the proof. ∎

Thus, a window size of $w_i = \lceil \frac{\ln(1/\delta)}{p_i^{avg}} \rceil$ is sufficient to guarantee completeness (with high probability). In general, due to the weak (logarithmic) dependence on δ, small settings for δ (i.e., less than 0.1) do not have a large effect on the overall window size.

While using a smoothing-window size as suggested by Lemma 1 guarantees completeness (i.e., correct detection of tag i) with high probability, it can also lead to missing the temporal variation in the underlying signal (e.g., due to the movements of tag i). Note that in the per-tag case, we are dealing with a *binary signal*: either tag i is there (value = 1) or it is not (value = 0). As discussed earlier, large smoothing windows can *miss signal transitions*, where tag i is mistakenly presumed to be present in the reader's detection range due to the interpolation of readings inside the window (Figure 5.1). In order to avoid smoothing over transitions and producing many false positives, SMURF needs to accurately determine when tag i exited the reader's detection range (as opposed to a period of dropped readings) and decrease the size of its window. I term this process *transition detection*.

Given the unreliability of tag readings, accurate transition detection is crucial: readings *will* routinely be lost (e.g., for tags outside the reader's major detection region (Figure 5.2)), and thus an overly-sensitive transition detection mechanism can result in failing to compensate for lost readings. On the other hand, a coarse detection mechanism can miss true signal transitions, resulting, once again, in false positives. SMURF employs its binomial sampling model to detect transitions in a principled manner as *statistically-significant deviations* in the observed binomial sample size from its expected value. More formally, assuming that the current window size w_i and sampling probability p_i^{avg} are not too small, it follows from a Central Limit Theorem (CLT) argument that, assuming no transition occurred in the current window, the

value of $|S_i|$ is within $\pm 2\sqrt{\text{Var}[|S_i|]}$ of its expectation with probability close to 0.98. Based on this observation, SMURF flags a transition (i.e., exit) for tag i in the current window if the number of observed readings is less than the expected number of readings *and* the following condition holds[3]:

$$||S_i| - w_i p_i^{avg}| \; > \; 2\sqrt{w_i p_i^{avg}(1 - p_i^{avg})}. \tag{5.1}$$

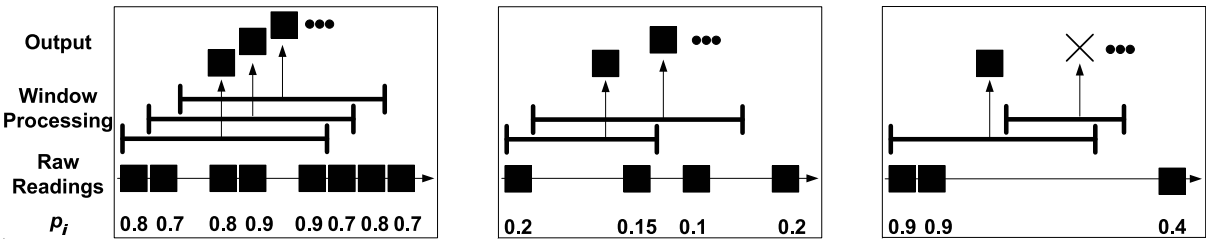
Estimating Data Quality for Cleaned Per-Tag Readings. Through its principled, statistical sampling framework, SMURF can also compute and attach *uncertainty estimates* for each tag reading emitted to higher-level applications. More specifically, consider the value $|S_i|$ for tag i in the current window, and let $w_i p_i^{avg} - |S_i| = \eta$, where η does not satisfy the transition condition (5.1) above. Then, we can attach an uncertainty indicator with the emitted reading of tag i, indicating the level of confidence in the presence of tag i (during the window). This is done by estimating an upper bound on the probability

$$\begin{aligned} &\mathbf{Pr}[w_i p_i^{avg} - |S_i| = \eta \mid \text{tag } i \text{ is present}] \\ &\quad < \; \mathbf{Pr}[|w_i p_i^{avg} - |S_i|| \geq \eta \mid \text{tag } i \text{ is present}]. \end{aligned}$$

Such upper bounds can be estimated based on either a CLT argument using the η-percentiles of the Normal distribution, or standard tail inequalities for the binomial distribution, such as Chernoff bounds [MR95]. For instance, using Chernoff bounds, we have

$$\begin{aligned} &\mathbf{Pr}[|w_i p_i^{avg} - |S_i|| \geq \eta \mid \text{tag } i \text{ is present}] \\ &\qquad \leq 2e^{-\eta^2/(2w_i p_i^{avg})}. \end{aligned}$$

[3] More conservative, non-CLT-based probabilistic criteria, e.g., based on the Chebyshev or Chernoff bounds [MR95], can also be used here.



(a) Normal sliding window processing for tag i in SMURF. At each epoch, SMURF emits a reading with a timestamp corresponding to the midpoint of the window.

(b) Ensuring completeness. In the left-most window, the p_i^{avg} demands a larger window such that the tag has a high probability $(1-\delta)$ of being detected. Thus, the window size is increased.

(c) Transition detection. In the left-most window, the number of readings indicates a statistically-significant deviation given the p_i^{avg}. Thus, a transition is likely to have occurred so the window is halved.

Figure 5.4. Graphical depiction of per-tag cleaning in SMURF.

For the experimental evaluation (Section 5.4), I use the above equation based on Chernoff bounds to compute SMURF's uncertainty estimates.

SMURF Per-Tag Cleaning Algorithm. A pseudo-code description of SMURF's adaptive per-tag cleaning algorithm is depicted in Algorithm 1. SMURF employs the common Additive-Increase/Multiplicative-Decrease (AIMD) paradigm [CJ89] to adjust its window size for each tag i, based on guidance from its binomial-sampling model as discussed above.[4]

SMURF runs its sliding-window smoothing for each observed tag i. The window size is initially set to one epoch for each tag, and then adjusted dynamically based on observed readings. (If at any point during processing SMURF sees an empty window for a tag, it resets its window size to one epoch.)

During each new epoch, and for each tag i, SMURF starts by processing the readings of tag i inside the window W_i (`processWindow`(W_i)). This processing includes

[4]Note that this algorithm uses only simple mathematical operations; thus, the overhead beyond traditional smoothing techniques is minimal.

Algorithm 1 SMURF Adaptive Per-Tag Cleaning

Require: T = *set of all observed tag IDs*

δ = *desired completeness confidence*

$\forall i \in T$, $w_i \leftarrow 1$

while (`getNextEpoch`()) **do**

  for (i in T) **do**

    `processWindow`(W_i)

    $w_i^* \leftarrow$ `completenessSize`(p_i^{avg}, δ) // *Lemma 1*

    if ($w_i^* > w_i$) **then**

      $w_i \leftarrow \max\{\min\{w_i + 1, w_i^*\}, 1\}$

    else if (`detectTransition`($|S_i|$, w_i, p_i^{avg})) **then**

      $w_i \leftarrow \max\{\min\{w_i/2, w_i^*\}, 1\}$

    end if

  end for

end while

estimating the required model parameters for tag i (e.g., p_i^{avg}, $|S_i|$) using tag-list information as well as emitting an output reading for tag i if there exists at least one reading within the window. Then, SMURF consults its binomial-sampling model to determine the number of epochs necessary to ensure completeness with high probability (`completenessSize`(p_i^{avg}, δ)), based on Lemma 1. If the required size w_i^* exceeds the current window size $w_i = |W_i|$, SMURF grows its current window size for tag i additively. This additive window growth rule allows SMURF to incrementally monitor the tag's readings as the window grows and thus remain responsive to changes in the underlying signal.

If the current window size satisfies the completeness requirement, then SMURF tries to detect if a transition occurred during W_i (`detectTransition`($|S_i|$, w_i, p_i^{avg})), based on Condition (5.1). If a transition is flagged, SMURF multiplicatively de-

creases the size of its current smoothing window for tag i (i.e., divides it in half). By multiplicatively decreasing its window size, SMURF can quickly react to detected transitions, while at the same time avoiding over-reaction in the unlikely event of an incorrect transition detection. Of course, if the completeness requirement is met and no transition is detected, SMURF continues with its current window size for tag i.

To summarize, Figure 5.4 graphically depicts some example scenarios in SMURF's basic per-tag cleaning scheme.

5.3.4 Adaptive Multi-Tag Aggregate Cleaning

In many real-world RFID scenarios, applications need to track large populations of tags, typically in the several hundreds or thousands. In addition, applications often do not require information for each individual tag, and only need to track simple *aggregates* (e.g., counts or averages) over the entire tag population. For instance, a retail-store monitoring application may only need to know when the *count* of items on a shelf drops below a certain threshold.

A simple cleaning approach in such scenarios is to apply SMURF's per-tag cleaning algorithms (Section 5.3.3) for each individual tag in the population and then aggregate the results across individual smoothing filters for each epoch. Such a solution, however, potentially suffers from underestimation bias: tags not read *at all* in a window will not be counted. Additionally, this approach incurs overhead: SMURF needs to continuously track and dynamically adapt the window for each individual tag; furthermore, many window adjustments can happen (e.g., with mobile tags) even though the underlying aggregate signal (e.g., population count) remains stable. To

avoid these problems, SMURF employs statistical-estimation techniques to accurately estimate the population count without cleaning on a per-tag basis.

Random-Sampling Model and Estimators for Multi-Tag Aggregates. Consider the problem of estimating the *count* of the tag population over a window of size w epochs (say, $W = (t - w, t]$). As earlier, I use p_i^{avg} to denote the average empirical sampling probability for tag i during W (i.e., the average read rate over all observations of i in W derived from the reader's tag list information). SMURF views each epoch as an independent "sampling experiment" (i.e., Bernoulli trial) with success probability p_i^{avg}; thus, the overall probability of reading tag i *at least once* during W is estimated as:

$$\pi_i \;=\; 1 - (1 - p_i^{avg})^w. \tag{5.2}$$

Again, the size w of the smoothing window plays a critical role in capturing the underlying aggregate signal: a large w ensures completeness (i.e., all π_i's are close to 1), but a small w is often needed to ensure that the variability in the population count is adequately captured. Unfortunately, compromising on completeness implies that RFID smoothing algorithms that simply report the observed readings count can suffer from consistent underestimation errors.

SMURF employs its unequal-probability random sampling model to correct for this under-estimation bias through the use of π-*estimators* (also known as *Horvitz-Thompson estimators*) [SSW92] to approximate population aggregates.[5] Specifically, let $S_W \subseteq \{1, \ldots, N_W\}$ denote the subset of observed (i.e., sampled) RFID tags over the window W (N_W denotes the true count), with sampling probabilities determined by Equation (5.2). The π-estimator for the population count based on the sample

[5] Although the discussion here focuses primarily on tag counts, SMURF's π-estimator scheme for adaptive multi-tag cleaning can be easily extended to other aggregates. For instance, if the goal is to estimate the sum of some measure (e.g., temperature) over the underlying tag population, then the contribution of tag i to the π-estimator formula becomes $\frac{y_i}{\pi_i}$, where y_i is the measured quantity of interest.

S_W is defined as:

$$\hat{N_W} = \sum_{i \in S_W} \frac{1}{\pi_i}.$$

In other words, the count π-estimator weights each sample point i with its sampling probability π_i. The reason for this is fairly intuitive: if tag i, which is observed with probability π_i, appears once in the sample, then, on average, we expect to have $1/\pi_i$ tags with similar probabilities in the full population (since $\pi_i \cdot 1/\pi_i = 1$); thus, the single occurrence of tag i in the sample is essentially a "representative" of $1/\pi_i$ tags in the full population.

The $\hat{N_W}$ π-estimator is *unbiased* (correct on expectation); that is, $E[\hat{N_W}] = N_W$ [SSW92]. Thus, by weighting with sampling probabilities, SMURF's π-estimator techniques correct for the underestimation bias of conventional smoothing schemes in a principled, statistical manner (even for small smoothing window sizes). Similar calculations show that, assuming independence across different tags, the variance of $\hat{N_W}$ is estimated by [SSW92]:

$$\hat{\text{Var}}[\hat{N_W}] = \sum_{i \in S_W} \frac{1 - \pi_i}{\pi_i^2}. \tag{5.3}$$

Of course, even though SMURF guarantees unbiasedness, as the window shrinks, the observed sample size and corresponding π_i's also drop, resulting in possibly lower-quality (high-variance) π-estimators. As the experimental results demonstrate, SMURF's π-estimation algorithms still significantly outperform conventional smoothing algorithms in such "difficult" settings.

Adaptive Window Size Adjustment for Multi-Tag Aggregates. As in the single-tag case, I first consider the problem of upper-bounding SMURF's smoothing window in a manner that results in reasonably complete readings over the reader's detection range. Let S_W denote the sample of (distinct) tags read over the current smoothing window W, and let $p^{avg} = \sum_{i \in S_W} p_i^{avg} / |S_W|$ denote the average per-epoch

sampling probability over all observed tags. Following a rationale similar to that used in Lemma 1, the upper bound for SMURF's smoothing window size for multi-tag aggregate cleaning is set at $w = \lceil \frac{\ln(1/\delta)}{p^{avg}} \rceil$; in other words, for completeness, we require that the "average tag" in the underlying population is read with high probability ($\geq 1-\delta$). Note that a more pessimistic window-size estimate would use the *minimum* of the p_i^{avg}s in the above calculation to ensure that the "worst" tag is read; however, since SMURF employs π-estimators to correct for missed readings, such a pessimistic window could result in over-estimation errors.

SMURF also employs its random-sampling model and π-estimator calculations in order to dynamically adapt its smoothing window size to accurately capture the temporal variation in the population count (analogous to transition detection in the per-tag case). The key observation here is that SMURF can detect transitions in the underlying aggregate signal as *statistically-significant changes* in its aggregate estimates over sub-ranges of its current smoothing window. Specifically, assume $W = (t-w, t]$ is the current window, and let $W' = (t-w/2, t]$ denote the second half of W. Also, let $\hat{N}_W$ and $\hat{N}_{W'}$ denote the π-estimators for the tag population counts during W and W', respectively. Under similar CLT-like assumptions as in Section 5.3.3, the corresponding true population counts (N_W and $N_{W'}$) satisfy $N_W \in \hat{N}_W \pm 2\sqrt{\hat{\text{Var}}[\hat{N}_W]}$ and $N_{W'} \in \hat{N}_{W'} \pm 2\sqrt{\hat{\text{Var}}[\hat{N}_{W'}]}$ with high probability. Based on these observations, SMURF detects that a statistically-significant transition in population count has occurred in the second half of W if the following condition is satisfied:

$$|\hat{N}_W - \hat{N}_{W'}| \; > \; 2\left(\sqrt{\hat{\text{Var}}[\hat{N}_W]} + \sqrt{\hat{\text{Var}}[\hat{N}_{W'}]}\right). \tag{5.4}$$

The above condition essentially asserts that the difference $|N_W - N_{W'}|$ of true counts is non-zero with high probability.

There are two important points to note here. First, remember that the key problem with adaptive smoothing-window sizing is to correct for *false-positive readings*

due to a large window W and a drop-off in the true number of tags in the detection range over W. (An increase in the tag count over W is always "caught", regardless of the current window size, since the observed new readings are by default interpolated throughout the smoothing window.) Condition (5.4) attempts to accurately capture such significant drop-offs within the current window, and allows SMURF to adaptively shrink its smoothing window size.

Second, while Condition (5.4) with $W' = (t - w/2, t]$ is sufficient to identify count changes that persist for at least $w/2$ epochs within the smoothing window, it may still miss transitions that last for $< w/2$ epochs. A more general solution here is to check Condition (5.4) for a series of dyadic-size windows $W' = (t - w/2^i, t]$ $(i = 1, 2, \ldots)$ at the tail end of W and signal a transition whenever one of these conditions is satisfied. Note that, as the window W slides across time, any transition is initially located at the tail end of W and thus can be discovered by the above technique. The caveat here is that as the sub-range within W decreases, the variability of the $\hat{N}_{W'}$ estimate goes up, making it difficult to detect very short-lived transitions. The empirical results demonstrate that using Condition (5.4) for just the second-half window $W' = (t - w/2, t]$ is sufficient to provide accurate population-count estimates to applications.

Estimating Data Quality for Cleaned Multi-Tag Readings. Similar to the single-tag case, SMURF's statistical sampling foundation allows for uncertainty indicators to be attached to derived π-estimates and emitted to higher-level applications. In the multi-tag case, such indicators take the form of appropriate *confidence intervals* for the $\hat{N_W}$ π-estimators based on their unbiasedness and observed sample variances. Such intervals can be computed through standard probabilistic methods, e.g., using the Normal distribution based on CLT arguments, or using the (more conservative) Chebyshev bound [MR95]:

$$\mathbf{Pr}[|\hat{N_W} - N_W|] \geq \eta] \;\leq\; \frac{\hat{\mathrm{Var}}[\hat{N_W}]}{\eta^2}.$$

For SMURF's confidence interval, η, I use the following equation based on the Chebyshev bound above:

$$\eta = \sqrt{\frac{\hat{\text{Var}}[\hat{N_W}]}{\alpha}}, \tag{5.5}$$

where $1 - \alpha$ is the desired confidence level; that is, for $(1 - \alpha)$ percent of the readings reported by SMURF, we expect the true value of the tag count aggregate to be in the range $\hat{N_W} \pm \eta$.

SMURF Multi-Tag Cleaning Algorithm. Algorithm 2 depicts the pseudo-code for SMURF's multi-tag cleaning scheme that incorporates the above techniques. Similar to per-tag cleaning, SMURF uses AIMD to adjust its smoothing window size; however, in contrast to the per-tag case, only a single window W is maintained (and adapted) for all observed tags.

Algorithm 2 SMURF Adaptive Multi-Tag Cleaning

Require: δ = *desired average completeness confidence*

```
w ← 1
while (getNextEpoch()) do
  processWindow(W)
  W ← slideWindow(w)
  w* ← completenessSize(p^avg, δ)          // Lemma 1
  if (detectTransition(N̂_W, N̂_W', V̂ar[N̂_W], V̂ar[N̂_W'])) then
    w_i ← max{min{w_i/2, w_i*}, 1}
  else if (w* > w) then
    w_i ← max{min{w_i + 1, w_i*}, 1}
  end if
end while
```

For each epoch, SMURF starts by processing the readings in the window W

(`processWindow`(W)). This involves computing key window parameters (e.g., p^{avg}, $\hat{N}_{W'}$, $\hat{\text{Var}}[\hat{N}_{W'}]$), determining the aggregate contribution from each tag ($1/\pi_i$), and calculating (and subsequently emitting) the estimated tag count ($\hat{N}_W$) using π-estimation.

The window is then checked for a statistically-significant change in the count estimate in its second half (`detectTransition` ($\hat{N}_W$, $\hat{N}_{W'}$, $\hat{\text{Var}}[\hat{N}_W]$, $\hat{\text{Var}}[\hat{N}_{W'}]$)) based on Condition (5.4). If a change is detected, SMURF halves its window size. Otherwise, SMURF checks if the current window meets the completeness requirement based on the average tag detection probability p^{avg} and grows its window additively, if necessary.

Note that the ordering of the increasing and decreasing phases in Algorithm 2 is reversed from the per-tag case. Since SMURF's π-estimation scales-up readings in a window to estimate the underlying tag population, the completeness requirement (i.e., a large window) is not as crucial for accurate estimation as in the single-tag case (where a missed reading causes a 100% error). Thus, multi-tag processing in SMURF focuses primarily on capturing transitions in the aggregate and uses π-estimation to compensate for small windows in an unbiased manner.

5.3.5 Mobile Tag Detection

Here I present an enhancement to SMURF processing that applies to both per-tag and multi-tag cleaning.

Tags that are detected far away from the reader with a low probability can force SMURF to use a large smoothing-window (based on Lemma 1). While large windows are necessary to accurately detect *static tags* placed far from the reader, they can cause problems in environments where tags are *mobile*. For per-tag cleaning, a mobile tag detected with a low $p_{i,t}$ just before it leaves the reader's detection range causes

a large number of false positives since it forces an abnormally large window. In the multi-tag case, a similar reading results in an overly large contribution to the overall count estimate, and thus a large over-estimation error.

To alleviate the effects of low $p_{i,t}$s produced by mobile tags, we enhance SMURF with a pre-processing stage that recognizes mobile tags that are exiting the detection range and reacts accordingly. This stage, termed *mobile tag detection*, monitors individual tag $p_{i,t}$s and attempts to determine when low detection probabilities are caused by an exiting mobile tag (as opposed to a static remote tag, which should force a large window). Mobile tag detection uses a simple heuristic: tags that are read with consistently falling $p_{i,t}$s are likely to be moving away from the reader and, thus, may be exiting the detection range soon. Such readings with low $p_{i,t}$ values are filtered out by SMURF's mobile tag detector.

SMURF's mobile tag detection algorithm forms a best-fit line using least squares fitting with the observed $p_{i,t}$s in the window. Using the slope of this line (in units of $\frac{\Delta p_{i,t}}{epochs}$), SMURF calculates a filter threshold as $filterThresh = \epsilon - slope \cdot w_{md}$. This threshold is a value of $p_{i,t}$ for which it is estimated that the $p_{i,t}$ for the tag will drop below some value ϵ in the next w_{md} epochs, where w_{md} is w_i in the per-tag case and w in the multi-tag case. The reason the algorithm looks ahead w_{md} epochs is intuitive: the larger the window the greater the potential for false positives if the tag exits; thus, SMURF more aggressively filters readings when the window size is large. Using $\epsilon = 0$ yields a good indication of whether the tag will be exiting the detection range soon. Mobile tag detection filters all readings for mobile tags whose $p_{i,t}$s fall below this threshold, thus preventing such readings from adversely influencing the window size calculation or count estimation.

5.4 Experimental Evaluation

In this section, I experimentally evaluate SMURF's data cleaning techniques. For both per-tag and multi-tag cleaning, I illustrate two key points: (1) there is no single static window that works well in the face of fluctuating tag movement, reader unreliability, or both; and (2) across a range of environments with different levels of tag movement and reader unreliability, SMURF's cleaning techniques produce an accurate stream of readings (both individual tag IDs and counts) describing tags in the physical world.

5.4.1 Experimental Setup

In order to run experiments across a wide variety of scenarios, I built a data generator to produce synthetic RFID streams given realistic configurations of tags and readers.

Reader Detection Model. The data generator is based on RFID reader detection regions as observed in the tests described in Section 5.2. I simplify a reader's detection field to derive a model of RFID readers as shown in Figure 5.5.

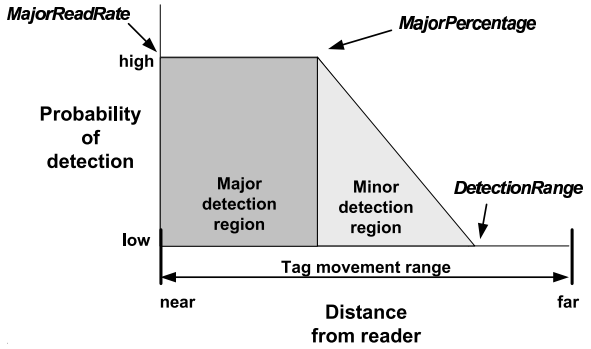


Figure 5.5. Reader model and tag behavior for the RFID data generator.

The model uses the following parameters to capture a wide variety of reader behavior under different conditions:

- *DetectionRange*: the distance in feet from the reader to the edge of the reader's detection range.
- $MajorPercentage$: the percent of the reader's overall detection range that is the major detection region.
- $MajorReadRate$: the read rate (i.e., the probability of detection) of a tag within the major detection region.

  While the experimental studies show that the read rate in the minor detection region has high variance (Section 5.2), for the sake of simplicity the read rate in this region is modeled as a linear drop-off from the end of the major detection region to the end of the reader's detection range. I ran additional tests where I introduced variance into the read rate in the minor detection region; these experiments yielded similar results to those presented here.

Tag Behavior. I randomly place $NumTags$ tags uniformly between 0 and 20 feet from the reader along its central axis. Here I have detailed data describing the read rate of the readers along this axis as described in Section 5.2. By moving the tags along this axis, I can generate readings with $p_{i,t}$s corresponding to many types of movement. For instance, the $p_{i,t}$s of readings produced by a tag passing through an RFID-enabled door can be generated by moving a tag from outside *DetectionRange* to directly in front of the reader, and then back to outside *DetectionRange*.

Tags move between 0 and 20 feet following one of two behaviors representative of a range of RFID applications:

1. *Pallet*: All tags have the same velocity. This simulates grouped tags, such as tagged items on a pallet.

2. *Fido*: Each tag chooses a random initial velocity (uniform between 1 and 3 feet/epoch). Note that the average velocity, 2 feet/epoch, is roughly equivalent to conveyor-belt speed [UWR06]. Every 100 epochs, on average, each tag switches from a moving state to a resting state (and vice versa). When a tag resumes movement, it chooses another random velocity between 1 and 3 feet/epoch. This behavior simulates tracking environments such as a digital home, where each tag displays independent random behavior.

Data Generation. The generator is run for $NumEpochs$ epochs.[6] At each epoch, the generator determines which tags are detected based on the read rate at each tag's location relative to the reader. It then produces a set of readings containing a tag ID, epoch number, and the tag's $p_{i,t}$ (the read rate at which the reader read the tag). Additionally, the generator produces the set of all tags within the reader's detection range at each epoch to serve as the reality against which I compare the output of each cleaning mechanism.

Table 5.1 summarizes the experimental parameters used to produce synthetic RFID data traces. I manipulate the other parameters as part of these experiments. The settings for the RFID detection model were chosen as they represent the average of the reader/tag combinations profiled. Recall from Section 5.2 the average read rate drops to around 0.8 with multiple tags in the reader's detection field; I set $MajorReadRate$ to reflect this behavior.

Smoothing Schemes. The data produced by the generator is cleaned using SMURF as well as various-sized static smoothing-window schemes. I denote each fixed-window scheme as *Static-x*, where x is the size of the window in epochs (1 epoch $\approx$ 0.2 seconds).

[6]To eliminate effects caused by the start or end of the trace, the generator is run for an additional 300 epochs and the first and last 150 epochs are omitted from the measurements.

Parameter	Value
$DetectionRange$	15 feet
$MajorReadRate$	0.8
$MajorPercentage$	varied
$NumTags$	25 (per-tag), 100 (multi-tag)
$Velocity$	varied
$NumEpochs$	5000 epochs

Table 5.1. Experimental parameters.

5.4.2 Per-Tag Cleaning Experiments

The first set of experiments examine cleaning techniques that report individual tag ID readings. I analyze the performance of different cleaning schemes as the environment changes in terms of tag movement and reader reliability.

The evaluation metric for per-tag cleaning is average errors per epoch. An error is a reading that indicates a tag exists when it does not (a false positive), or a (lack of) reading where a tag exists, but is not reported (a false negative). The average errors per epoch is calculated as $\sum_{j=1}^{NumEpochs}(FalsePositives_j + FalseNegatives_j)/NumEpochs$. This metric captures both types of errors in one metric that allows us to easily compare the effectiveness of each scheme.

Experiment 1: Varied Reader Reliability. In the first test, I determine how each technique reacts to different levels of reader unreliability. The tags move according to *Fido* behavior and the major detection region percentage is varied. At each value for $MajorPercentage$ between 0 and 1, I measure the average errors per epoch produced by each scheme (recall that a lower value for $MajorPercentage$ corresponds to a more unreliable environment). Figure 5.6 shows the results of this experiment.

As can be seen, when the major detection region percentage is 0 (a noisy environment), the large windows do comparatively well, producing around 4 errors per epoch (i.e., misreporting about 4 tags out of 25 per epoch, on average). The traces for

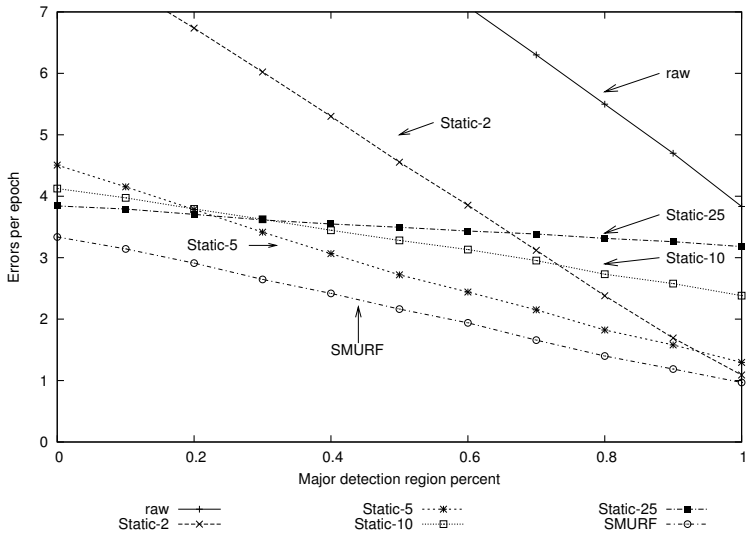


Figure 5.6. Average errors per epoch as $MajorPercentage$ varies from 0 to 1 with tags following *Fido* behavior.

raw and *Static-2* are truncated due to their poor performance. As $MajorPercentage$ increases, the accuracy of all schemes improves due to more reliable raw data. When the major detection region makes up the entire detection field ($MajorPercentage = 1$), the small windows are competitive; *Static-2* misreports slightly more than 1 tag out of 25 per epoch, on average.

In this experiment, SMURF cleaning has the lowest errors per epoch across the entire range of environments. Its relative performance is particularly good in this case because of its partitioned smoothing: it adapts, on a per-tag basis, to each tag's independent random behavior. Static windowing schemes that use a single window for all tags cannot capture this variation.

To further investigate the mechanisms behind each smoothing scheme, I drill-down on a 200 epoch trace of this experiment. I focus on readings produced from a single tag ID in a noisy environment: the major detection region percentage is set to 0 (the left-most x-value in Figure 5.6). The readings produced by the reader in this scenario are particularly challenging to clean as the data are highly unreliable and the tag

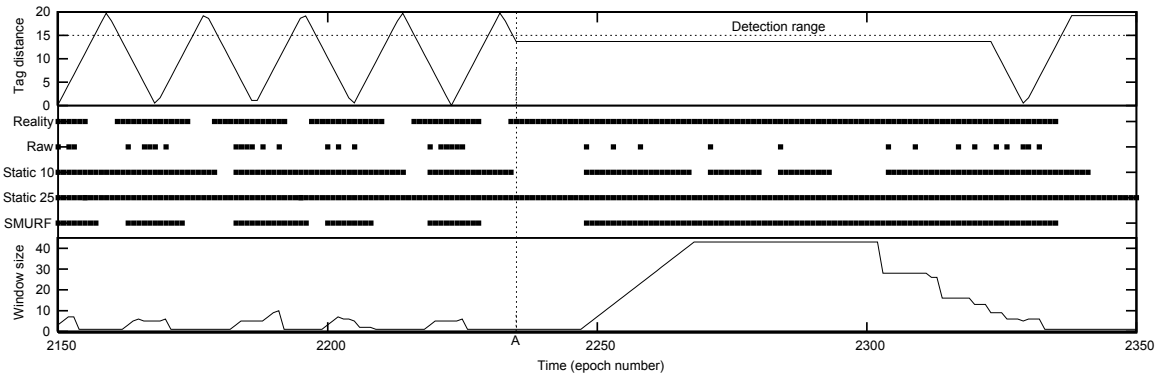


Figure 5.7. A 200-epoch trace of different mechanisms cleaning the readings from a single tag moving with *Fido* behavior.

sporadically moves at a high velocity: a smoothing scheme must be able to discern between periods of dropped readings and periods when the tag is transiently absent.

Figure 5.7 shows this time-line. The top section of the figure shows the tag's distance relative to the reader: the tag moves with a high velocity for a period, stops (at point A) for a period at the edge of the detection field, and then resumes movement. The middle section of the graph shows reality (e.g., the readings that would have been produced by a perfect reader), readings produced by the best two static window smoothing schemes (according the Figure 5.6), and the output of SMURF. The bottom section shows SMURF's window size over the course of the trace.

During the first period, the tag rapidly moves in and out of the detection field; the challenge for any smoothing scheme is to accurately capture this movement. Both static windows, however, fail to capture all of the tag's transitions. In the worst case, *Static-25* continuously reports the tag as present. Smaller windows would catch these transitions, but would perform worse during the second phase of this trace.

At point A (around epoch 2230), the tag stops at the edge of the detection range, causing the reader to infrequently report the tag. *Static-10* fails to report the tag's behavior due to lack of readings: according to *Static-10*, the tag is still moving. *Static-*

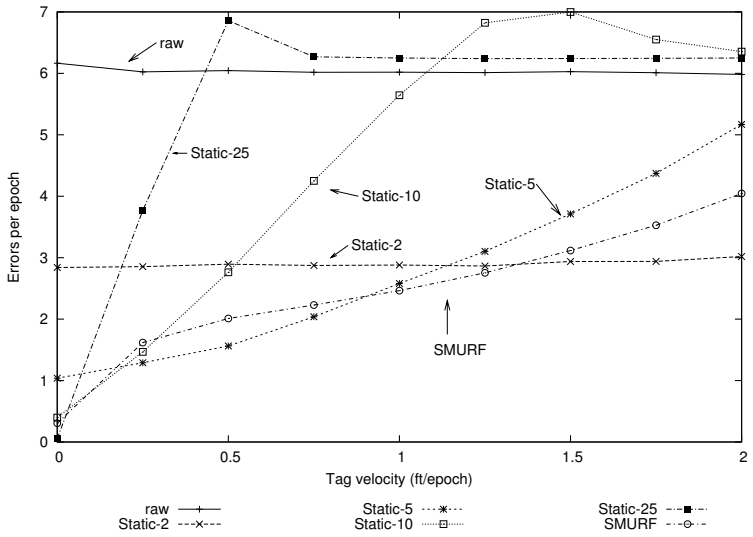


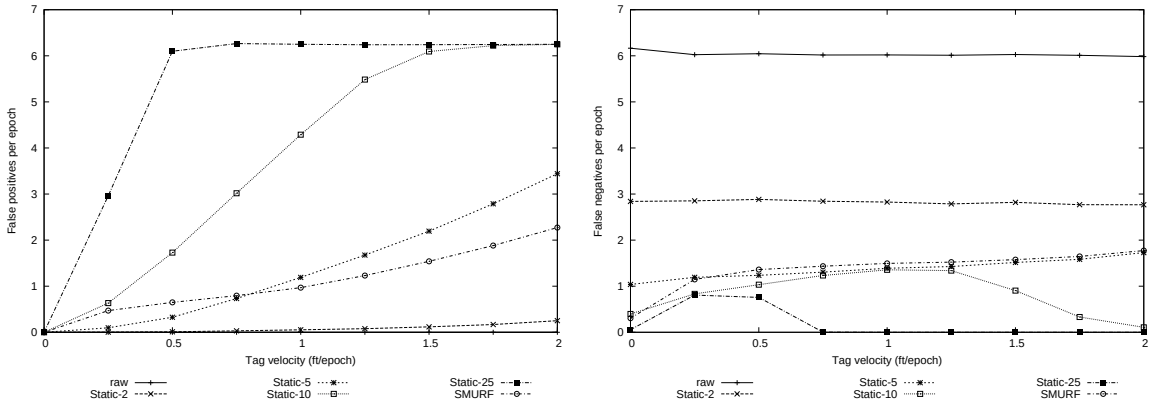
Figure 5.8. Average errors per epoch as tag velocities vary from 0 to 2 feet/epoch following *Pallet* behavior.

25 accurately reports the tag's presence only because it reports the tag's existence continuously.

SMURF, in comparison, captures the high-level behavior of the tag during the entire trace. During the first phase of tag movement, it keeps its window size small, as can be seen at the bottom of the figure, and accurately reports that the tag is moving; it succeeds at catching all transitions. Once the tag stops, SMURF grows its window in reaction to the unreliable readings it receives during this period. Thus, SMURF accurately reports the tag as present despite the severe lack of readings.[7]

Experiment 2: Varied Tag Velocity. Next, I measure each scheme's effectiveness as the tag velocity changes. The $MajorPercentage$ is fixed at 0.7 (representing a controlled environment) and tags move according to *Pallet* behavior. At each velocity from 0 and 2 feet/epoch, I measure the average errors per epoch produced by each scheme. Figure 5.8 shows the results of this experiment. Additionally, I measure

[7]Note that there is a short period just after point A where all schemes fail to report the tag while it exists. During this period, the reader produces no readings; no scheme without foreknowledge of the tag's motion can report the tag before it is read.



(a) Average false positives per epoch.

(b) Average false negatives per epoch.

Figure 5.9. Average errors per epoch as tag velocities vary from 0 to 2 feet/epoch following *Pallet* behavior, separated into false positives and false negatives.

separately the average false positives and average false negatives produced per epoch by each scheme, shown in Figure 5.9.

The results illustrate the challenge in setting a static smoothing window. As the tag velocity increases, there is no single static window that does consistently well. *Static-25* and *Static-10* do well when the tags are motionless by eliminating many of the dropped readings (they miss less than 1 tag out of 25 every other epoch, on average). As the tags speed up, however, the performance of the large windows degrade due to many false positives. The reason the errors for the two large windows drop at higher velocities is because at that point they continuously report all tags as present. Thus, while they produce a large number of false positives, they produce few or no false negatives.

On the other hand, the smaller windows (*Static-2* and *Static-5*), aren't able to fully compensate for lost readings. As the tag velocity increases, these schemes become comparatively better by filling in some of the missed readings without producing many false positives. *Static-5*, however, performs poorly at high tag speeds due to false positives. In a deployment where tags move with different velocities or change

velocities over the course of time, an application cannot set a single static smoothing window that captures the variation in tag movement to provide accurate data.

SMURF, in contrast, consistently performs well as the tags increase speed. When the tags are motionless, it removes many of the false negatives and is competitive with the large window schemes.

As the tags increase velocity, SMURF is able to generally track the best static window. At low velocities, SMURF does well, but not as well as *Static-5*. Here, tags are not moving fast and thus mobile tag detection has little effect. As a result, SMURF's binomial sampling scheme occasionally sets its window too large: it produces roughly twice as many false positives as *Static-5* at lower velocities. As the tags speed up, however, mobile tag detection filters readings from tags that are exiting and thus reduces the false positives. As can be seen in Figure 5.9, from tag velocities of 1 to 1.25 feet/epoch both SMURF and *Static-5* show similar increases in false negatives, but SMURF produces only $2/3^{rds}$ the false positives as *Static-5*.

At the highest velocities, *Static-2* performs better than SMURF. Here, the tag velocity is approaching a fundamental limitation for any detection scheme: if the time between transitions is smaller than the window size, then the transition will be lost. In this setup, at 2 feet/epoch the time between transitions is 5 epochs. Thus, for a smoothing scheme to be able to detect a transition, the window size must be set smaller than 5 epochs. In this experiment ($MajorReadRate = 0.8$, $MajorPercentage = 0.7$), SMURF uses an average window size (without transition detection or mobile tag detection) of $\lceil \frac{\ln(1/\delta)}{p_i^{avg}} \rceil = \lceil \frac{\ln(1/0.05)}{0.68} \rceil = 5$. Thus, the tag velocity in this case is at SMURF's limit; transition detection and mobile tag detection prevent it from breaking down completely.

Experiment 3: δ as a Declarative Parameter. While the primary contribution of SMURF is the removal of the imperative window size parameter from RFID data

cleaning, SMURF provides a parameter δ, where $(1 - \delta)$ is the probability of reading a tag if it exists, that allows the application to declare a preference for reduced false positives or reduced false negatives.

To analyze how the value for this parameter affects SMURF cleaning, I run SMURF with different values for δ. For this experiment, $MajorPercentage$ is set to 0.7 and tags move according to *Fido* behavior. Given that the dependence on δ of the binomial sampling approach used by SMURF is weak (logarithmic), using a small δ is common practice. Thus, I vary δ between 0.1 (90% completeness) to 0.01 (99% completeness). The results are shown in Figure 5.10. I ran this experiment with multiple tag and reader characteristics; all experiments produced similar results.

First of all, notice that the value of δ has very little impact on the overall number of errors SMURF produces. This means that an application that does not tune δ will not be adversely affected by choosing a standard value (e.g., 0.05).

Second, δ is a declarative parameter that allows an application to express *what* data it wants; SMURF then determines *how* to produce that data. If an application, such as an RFID-enabled doorway, desires responsiveness (low false positives), it can set δ closer to 0.1, in which case SMURF sets its window smaller. Conversely, an application (e.g., shelf-monitoring) that desires completeness (less false negatives) can set δ closer to 0.01 causing SMURF to grow its window larger to meet the completeness requirement. In either case, the value of δ has little impact on overall error.

Experiment 4: Accuracy of Uncertainty Estimates. Here I measure the accuracy of SMURF's uncertainty estimation. For this experiment, I use the same setup as in Experiment 2: $MajorPercentage$ is set at 0.7 and tags move according to *Pallet* behavior at varying speeds. At each tag velocity, I measure the *accuracy* of the uncertainty estimate. Accuracy measures how close the estimate is to reality, where reality is 1 if the tag is in the reader's detection field for the given epoch, 0 if it is

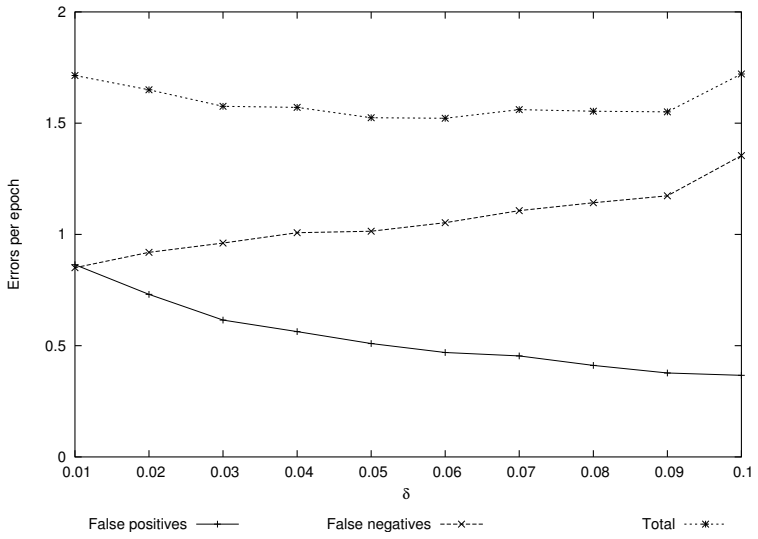


Figure 5.10. Errors per epoch using different values of δ.

not. If we denote a reading's uncertainty estimate as $conf$, then for correct readings the accuracy is defined as $conf$, and for incorrect readings the accuracy is defined as $(1 - conf)$.

For SMURF's estimation strategy, I assign an uncertainty estimate based on the Chernoff bound as described in Section 5.3. I denote this strategy as *Chernoff*. I compare the accuracy of SMURF's uncertainty estimation to three other strategies. The first comparison strategy, *Percent*, assigns a confidence based on the percent of positive readings in a window: $conf = \frac{|S_i|}{w_i}$. Strategy *Average* simply assigns a confidence using the average of the detection probabilities: $conf = p_i^{avg}$. The third algorithm I compare, *Unit*, assigns unit uncertainty estimates to every output reading: $conf = 1$.

Strategy	Average accuracy
Percent	0.73 (0.04)
Average	0.59 (0.02)
Unit	0.94 (0.03)
Chernoff	0.95 (0.01)

Table 5.2. Average accuracy of different mechanisms for estimating uncertainty across a range of tag velocities. The standard deviation is in parenthesis.

The results are shown in Table 5.2, reported as the average accuracy of the uncertainty estimates across all environments. Strategies *Average* and *Percent* perform poorly as they tend to assign quality estimates that are too low. Both *Chernoff* and *Unit* perform well. *Unit* performs well due to the fact that SMURF is very successful at producing clean data. In a sense, the goal of SMURF is to produce readings of confidence 1, so it is no surprise that *Unit* performs well. *Chernoff*, on the other hand, achieves the highest accuracy by assigning high confidence to true positive readings and lower confidence to false positive readings.

Experiment 5: Experiences with Real RFID Data. The previous experiments were based on a generator that created RFID data based on a simple model. Real-world RFID data does not follow this model exactly. Here I describe experiences with real RFID data and the performance of cleaning mechanisms on this data. First, I collect real RFID data under varying circumstances and examine how it differs from the model used in the generator. Second, I show that SMURF's cleaning techniques are robust to any discrepancies.

For these experiments, I recreate the conditions used in Experiment 2 through an RFID testbed deployed in the controlled environment from Section 5.2 using an Alien reader [ALR05] and a single Alien "I2" [Ali05a] tag suspended in the same plane as the antenna. I gather data using tag velocities ranging from 0 to 2 feet/epoch. For the motionless tag test, results from data collected every 0.5 feet from 0 to 15 feet (the reader's detection range is approximately 15 feet) are averaged. For the mobile tag tests, I move the tag back and forth between 0 and 20 feet from the reader. For tag velocities that cannot be produced in the testbed (1.5 and 2 feet/epoch) I collect data at lower velocities and then speed up the data traces. All runs are performed for 2000 epochs ($\approx$ 400 seconds). Additionally, I collect limited traces from two reader positions in the noisy environment, differing by $\approx$ 5 feet.

During the course of these experiments, I discovered that real RFID data differ from the model in two main ways. First, if the reader is deployed near obstacles (e.g., walls), its detection field does not follow the same shape as seen in all other positions: it is much more irregular. The detection field for a reader deployed close to a wall and metal desks, for instance, has multiple high and low detection regions. Such behavior argues for an adaptive approach to data cleaning: very small changes in the environment can cause dramatic changes in RFID reader and thus necessitates changes to any static windowing scheme.

Real RFID data differ from the model in another important way: the reader occasionally produces many more or many less readings than expected based on the reported $p_{i,t}$. For instance, the reader occasionally produces many readings with a very low $p_{i,t}$ (e.g., 0.1) in a window; SMURF is robust to such cases. In rare cases, a tag statically placed at very specific distances relative to the reader (e.g., $\approx$ 12 feet $\pm$ 2 inches for one of the reader positions) will cause the reader to occasionally produce only one reading in 5-10 epochs, but report the $p_{i,t}$ of the reading as greater than 0.8. Based on this $p_{i,t}$, it is expected to see roughly 8 readings in a window of 10 epochs. In such cases, the SMURF algorithm mistakenly signals a transition and shrinks the window, causing many false negatives (e.g., 12% dropped readings versus 10% for *Static-10* and 2% for *Static-25*). As such behavior occurs rarely and only in very specific locations with static tags, it is not expected to be a problem in practice. If necessary, the δ parameter can be used to help alleviate the effects of these types of readings: by setting δ to 0.01, the dropped readings are reduced to 6%.

Finally, these tests confirm the two key points. Across the different speeds and environments, there is no single static window that works uniformly well. At high speeds in the controlled environment, *Static-2* works very well, while it falters at slow speeds and in the noisy environment. On the other hand, *Static-25* works very well with a motionless tag, but performs poorly when the tag starts moving. In contrast,

SMURF handles all of these cases well. When the tag moves fast in the controlled environment, it closely follows *Static-2* while at the same time competing with *Static-25* when the tag is motionless. On average, SMURF performs the best: for instance, in the controlled environment, SMURF averages 0.05 errors per epoch, compared to 0.06 for *Static-2* and *Static-5*, 0.14 for *Static-10*, and 0.18 for *Static-25*.

Due to the difficulty in running controlled experiments with RFID technology, for the remainder of the experiments I use synthetic data streams.

5.4.3 Multi-Tag Aggregate Cleaning Experiments

As stated in Section 5.3.4, many applications only need a count of the tagged items in the area. Here I compare techniques for accurately counting the number of tags in a reader's detection field.

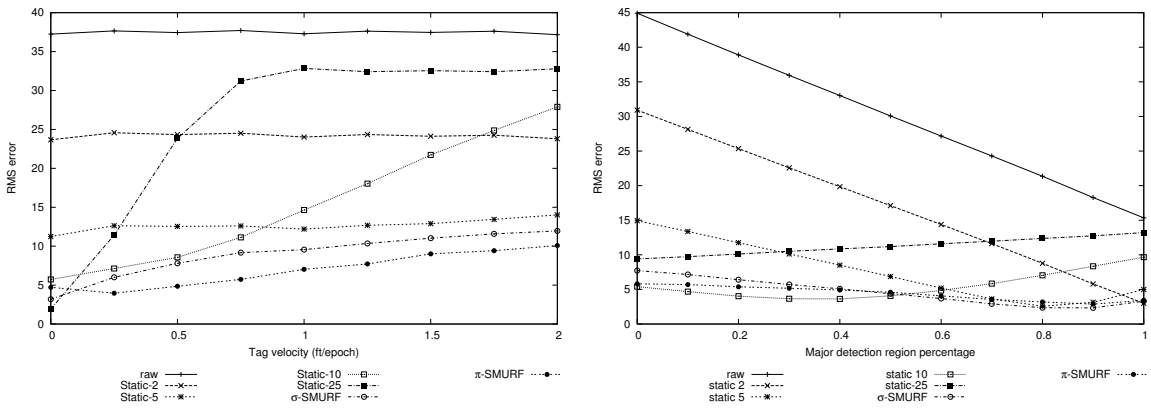
I show the same static windowing schemes as the previous experiments (*Static-2*, *Static-5*, *Static-10*, *Static-25*). For count aggregates, these schemes use the equivalent of a windowed count distinct operation. For SMURF processing, I show two versions, as outlined in Section 5.3.4: SMURF using per-tag cleaning with summation (σ-SMURF) and SMURF using π-estimators (π-SMURF).

As π-SMURF cannot produce individual tag readings, I change the evaluation metric to root-mean-square error (RMS error) of the count of reported tags compared to reality.

Experiment 6: Varied Reliability and Tag Velocity. I test the accuracy of the counts produced by each scheme as either the level of tag movement or unreliability increases. I run the same tests as Experiments 1 and 2, but with more tags (100), and measure the RMS error of each scheme's output compared to reality.

To determine the count accuracy of different schemes as the tag velocity increases,

I run a similar test to Experiment 2. The $MajorPercentage$ is set at 0.25 and the tag velocity is varied between 0 and 2 feet/epoch using *Pallet* behavior. The results are shown in Figure 5.11(a).



(a) RMS error of each scheme as the tag velocity increases. (b) RMS error of each scheme as the reader reliability increases.

Figure 5.11. The RMS error of different cleaning schemes counting 100 tags.

In most cases, both σ-SMURF and π-SMURF are more accurate than any static window. π-SMURF does particularly well here due to its unbiased nature. σ-SMURF, however, suffers from under-counting. To illustrate, I also measure the mean error of the count estimates (a measure of the bias of an estimator). At a tag velocity of 1 foot/epoch, for example, σ-SMURF has an mean error of -6.5, indicating an under-count of 6.5 items, on average. π-SMURF, in comparison, only has a mean error of -0.3: on expectation, π-SMURF provides accurate estimates.

To determine how each scheme performs as the level of reliability changes, similar to Experiment 1, I move tags using *Fido* behavior and vary the major detection region percentage. At each value of $MajorPercentage$, I measure the error of each scheme. Here, both σ-SMURF and π-SMURF are competitive with the best static window.

The results are shown in Figure 5.11(b). Across a range of environments, both σ-SMURF and π-SMURF are competitive with the best static window. Note that

this case represents the worst-case scenario for π-SMURF. As tags are moving independently and at random, on average there are an equal number of tags exiting and entering the detection field at each epoch. Since π-SMURF only looks at the count of tags, it is unable to recognize these changes. Despite this limitation, it remains close to the best scheme.

Experiment 7: Accuracy of Multi-Tag Uncertainty Estimates. I also verify the accuracy of SMURF's uncertainty estimates in the multi-tag case. For these tests, I fix the confidence bound at 95% and confirm that the true count of tags falls within the range supplied by SMURF's confidence interval. For this uncertainty estimate, I use π-SMURF for the count estimate and derive a confidence interval based on the Chebyshev bound as described in Equation 5.5. I use the same setup as in the second half of Experiment 6: tags are moved according to *Fido* behavior and the major detection region percentage is varied. For each value of $MajorPercentage$, I measure the *hit rate*, or percentage of epochs for which the true count of tags is within SMURF's confidence interval.

Across most of the range of $MajorPercentage$, the true count of tags is within the confidence interval over 95% of the time; when $MajorPercentage$ is between 0 and 0.9, the average hit rate is 0.97. When the reader is highly reliable ($MajorPercentage > 0.9$), however, the hit rate suffers: when $MajorPercentage = 1$, the hit rate falls to 0.28. This low accuracy is due to the fact that the tags are mobile in a very reliable environment: within the smoothing window the p_i^{avg} is high (> 0.8), yielding a low variance for the estimator and consequently a tight confidence interval; at the same time, the tags are highly mobile, so some tags included in the estimate have left the detection range causing a less accurate count estimate. In general, SMURF's uncertainty estimates will be less accurate when the reader is very reliable and tags are moving quickly. Since it is highly unlikely for a reader to have a

$MajorPercentage$ greater than 0.9, these poor uncertainty estimates are likely not a problem in practice. Note that SMURF's uncertainty estimates remain accurate when the reader is highly reliable and the tags are not mobile; when $MajorPercentage = 1$ and the tags are not mobile, the hit rate rises to 0.99.

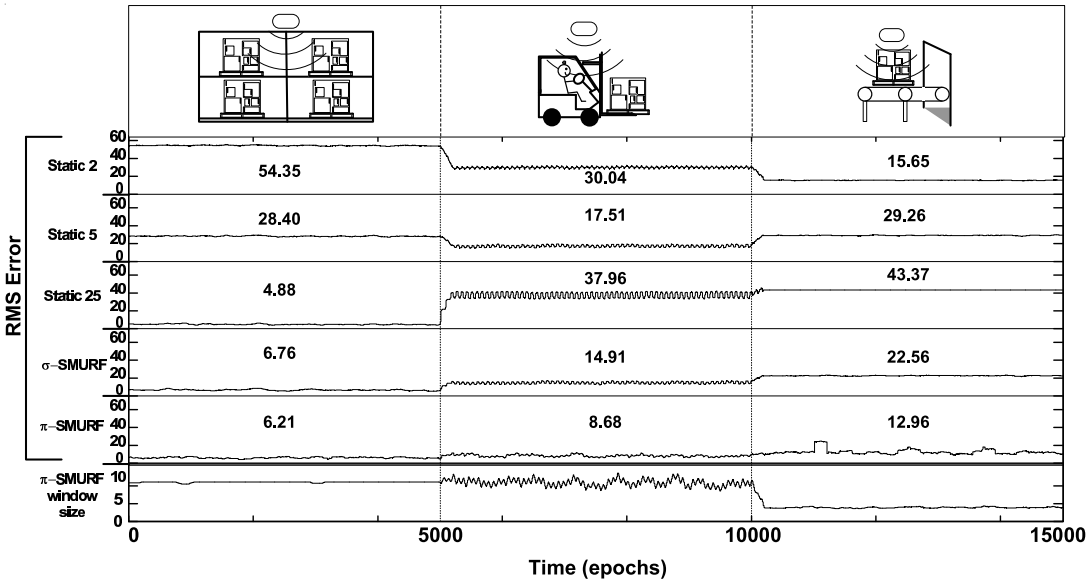


Figure 5.12. A simulated pallet moving through three phases in a warehouse: shelf, forklift, and conveyor belt.

Experiment 8: Tracking Counts in a Dynamic Environment. Here I investigate how different multi-tag cleaning schemes react as conditions change over time. I simulate tag movement and reader characteristics typical of a warehouse scenario over the course of 15000 epochs. In this scenario, an application monitors the count of 100 tags placed together on a pallet as it travels through the warehouse in three phases (as depicted at the top of Figure 5.12):

1. ***Shelf***: In the first phase, the pallet is motionless on a shelf. Due to interference from the shelf and other tags in the vicinity, the read rate is low: the $MajorReadRate$ is set to to 0.5 and $MajorPercentage$ to 0.5.

2. ***Forklift***: After 5000 epochs, a forklift picks up the pallet and begins moving. Here, there is less reader interference due to other tags or obstructions, but the forklift

reduces the major detection region ($MajorReadRate = 0.8$, $MajorPercentage = 0.25$). The forklift's motion is simulated by moving the tags at 0.5 feet/epoch.

3. ***Conveyor Belt***: In the final phase, I simulate the pallet traveling on a conveyor belt. Here, the reader environment is controlled to reduce unreliability ($MajorReadRate = 0.8$, $MajorPercentage = 0.7$). The tags, however, move very fast (2 feet/epoch).

These three phases simulate realistic conditions in terms of tag and reader behavior. Any cleaning scheme should be able to handle all of these conditions to produce accurate readings describing the count of items on the pallet as it moves through the warehouse.

The data produced by the tags on the pallet is cleaned using different schemes and measure the RMS error during each phase as shown in the middle section of Figure 5.12. Additionally, I include a trace of a 100-epoch sliding window of the RMS error for each scheme to illustrate how accuracy changes over time.

When the pallet is on the shelf, the raw data (not shown) is very poor (reporting less than 20 tags out of 100 per epoch on average). To clean this data, a large window must be used: with either counting technique, SMURF provides a stream of count readings that are competitive with *Static-25*, the largest static window (of course, larger windows would do better here, but I omit them due to poor performance during the remainder of the experiment). The bottom portion of the figure shows the trace of a 100-epoch moving average of the window size set by π-SMURF. During the period when the tags are motionless, π-SMURF sets its window large to compensate for the unreliability of the reader.

Once the tags start moving, both SMURF techniques adjust their window sizes to balance unreliability and tag movement to outperform all static window schemes.

Finally, when the tags are moving very fast in a controlled environment, π-SMURF

does particularly well as it drastically reduces its window size in reaction to the tags' movement while using π-estimators to avoid under-counting with such a small window.

As can be seen, there is no single static window that the warehouse monitoring application case use to provide accurate counts in this scenario. Using SMURF, in contrast, the application can get accurate readings throughout the pallet's lifetime without setting the smoothing window size. π-SMURF further refines its accuracy by providing an unbiased estimate.

5.5 Providing MDI for RFID-based Applications

In the previous sections, I detailed how SMURF effectively deals with RFID's unreliable and frequently changing nature by adaptively cleaning the data streams. In this section, I give an overview of MDI-SMURF, an infrastructure that utilizes SMURF and its statistical framework to provide data that conforms to the Meta-physical Data Independence interface.

MDI-SMURF Overview

MDI-SMURF is an RFID middleware platform organized as set of processing stages as shown in Figure 5.13, each designed to deal with a different aspect of RFID data. To track uncertainty for exposure through the MDI inteface, MDI-SMURF contains an associated uncertainty-tracking shadow pipeline. RFID data from readers flow into a smoothing filter as described in the previous section to compensate for missed readings. A shadow module estimates the resulting uncertainty of the cleaned readings. These cleaned readings are then streamed into a module that extends SMURF's statistical framework to address errors and semantic issues that arise from multiple RFID readers deployed in close proximity. A simple conversion module

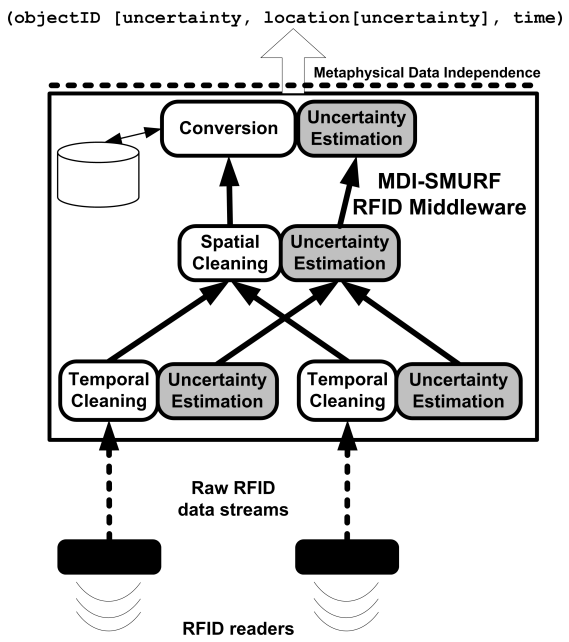


Figure 5.13. An architecture for providing MDI for RFID data using SMURF.

translates the temporally and spatially cleaned readings to MDI readings with the following schema: `(objectID[uncertainty], location[uncertainty], time)`.

In the remainder of this section, I briefly outline the spatial cleaning and conversion components of MDI-SMURF.

Spatial Cleaning

There is typically a spatial mismatch between readers' detection fields and application-level spatial units. That is, the area in which a reader is able to detect tags is usually not the same as the unit of space in which the application is interested. Furthermore, most RFID deployments contain multiple readers placed in close proximity to ensure full coverage of the area of interest and ensure completeness.

These two factors lead to overlapping detection fields, causing potential duplicate readings. Duplicates may be from readers in the same area (e.g., two readers in the same room of a digital home) and thus *reinforce* each other; in such a case, the

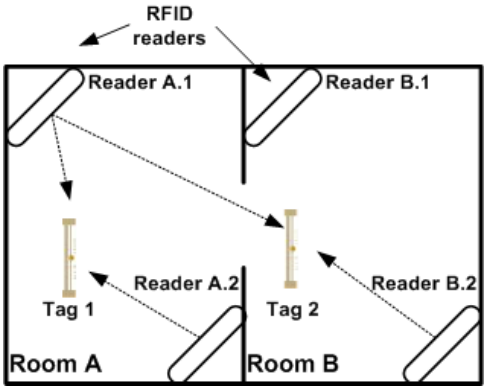


Figure 5.14. Spatial issues in RFID deployments. Readings for tag 1 reported by readers A.1 and A.2 reinforce each other, while readings for tag 2 from readers A.1 and B.2 must be arbitrated.

system should boost the confidence of these readings. In other cases, the readings may conflict with one another (e.g., two readers on adjacent shelves in a retail scenario) and must be *arbitrated*, as discussed in the previous chapter.

Figure 5.14 illustrates these issues. Here, a digital home application is monitoring room occupancy for two rooms using four RFID readers (two in each room). Reinforcement is shown by the readings for tag 1 from readers A.1 and A.2. Arbitration is necessary for the readings for tag 2 from readers A.1 and B.2.

The opportunities of reinforcement and the challenges of arbitration cause problems for sensor-based applications. In the case of reinforcement, an application has to understand the effect on overall confidence, and perhaps collapse several readings into a single reading with appropriately-adjusted (higher) confidence. For arbitration, the application is presented with a tag that is reported in two or more locations and must somehow decide on an appropriate spatial location the tag. Dealing with both of these issues involves detailed knowledge of the detection fields of the readers involved and how these fields change over time, further complicating the processing of RFID readings.

Here, I briefly outline some ideas on how to apply SMURF's statistical-modeling

techniques to handle such issues in the design of a *spatial* RFID smoothing filter designed to enable applications to effectively cope with reinforcement and arbitration in RFID readings.

In a nutshell, this spatial smoothing filter approaches RFID reinforcement and arbitration issues using a *spatial window* abstraction. A spatial window is a grouping of readers across space that provides a multi-resolution probabilistic statement about a tag's actual location The output of the filter's spatial windowing is a combination of intersections and unions of reader detection fields, with attached uncertainty estimates indicating the probability of the tag being located inside the corresponding spatial window. Applications can map these spatial windows to application-level spatial units; as we discuss below, MDI-SMURF incorporates a module that enables each deployment to specify the appropriate spatial mappings.

The abstract goal of the spatial smoothing filter, then, is to determine, for each tag, appropriate groupings of reader detection fields as well as corresponding uncertainty indicators (i.e., existence probabilities) for the tag in each grouping. I outline a brief example showing how some of the above ideas can be applied in the context of spatial cleaning.

Example 5.5.1 *Consider a simple scenario with two RFID readers A and B and let $r(A)$ and $r(B)$ denote their (possibly overlapping) detection fields. The output of each reader is fed into a temporal smoothing module that assigns an uncertainty estimate for the existence of any tag i in its detection field based on the sampling bounds outlined earlier in this paper; that is, we can estimate the probabilities $\rho_{r(A)} = \mathbf{Pr}[i \in r(A)]$ and $\rho_{r(B)} = \mathbf{Pr}[i \in r(B)]$.*

Since $r(A)$ and $r(B)$ can overlap, in order to provide a multi-resolution spatial window to applications, a spatial smoothing filter can also estimate the uncertainty indicators for other spatial regions of interest, such as $r(A) \cap r(B)$ and $r(A) \cup r(B)$.

To compute $\rho_{r(A)\cap r(B)}$, the "naive" approach of assuming independence of readers (i.e., setting $\rho_{r(A)\cap r(B)} = \rho_{r(A)} \cdot \rho_{r(B)}$) is incorrect if the detection fields overlap. Instead, the spatial smoothing filter can estimate the uncertainty of $r(A) \cap r(B)$ (i.e., the probability that tag i lies in the overlap region of readers A and B) using the formula $\rho_{r(A)\cap r(B)} = \mathbf{Pr}[i \in r(A) | i \in r(B)] \cdot \rho_{r(B)}$, where the conditional probability on the right-hand side of the equation can be estimated using a variety of techniques such as using the percentage of spatial overlap across detection ranges together with some spatial uniformity assumption, or from historical overlap data collected for the two readers. Similarly, the spatial smoothing filter can estimate $\rho_{r(A)\cup r(B)} = \rho_{r(A)} + \rho_{r(B)} - \rho_{r(A)\cap r(B)}$.

Higher-level applications are then provided with a multi-resolution spatial-window probabilistic statement of the form: $\{r(A)[\rho_{r(A)}]$, $r(B)[\rho_{r(B)}]$, $r(A) \cap r(B)[\rho_{r(A)\cap r(B)}]$, $r(A) \cup r(B)[\rho_{r(A)\cup r(B)}]\}$, and can determine the appropriate level of spatial localization for the tag based on the observed confidence levels. □

Conversion There is a semantic gap between the low-level data RFID readers provide and the needs of applications. The output of most RFID middleware is a stream of tag IDs and the reader at which they were read. Applications, on the other hand, typically want to know about high-level concepts such as products, people, rooms, and shelves.

The final stage in the MDI-SMURF pipeline is a *conversion* module that translates the smoothed and spatially processed streams from RFID-specific readings into MDI readings. Basically, the main goal of this module is to convert the data into a stream with the following schema: `(objectID[uncertainty], location[uncertainty], time)`. Producing the objectID involves a relatively simple mapping from tag ID to an application-specific object ID, such as the approach proposed for ONS [ONS05]. For the location, the spatial smoothing filter's spatial windows need to be mapped to

an application-level spatial unit as mentioned above. This module can incorporate this logic internally so as to shield applications from such complexity. In the next chapter, I show how these mappings can be defined in a declarative manner to ease configuration.

Hiding these mappings in MDI-SMURF rather than exposing them to the application provides the final step for supporting MDI: all RFID-specific information is hidden by MDI-SMURF. Thus, tag IDs in a deployment can change (e.g., the deployment can switch from pallet-level to item-level tagging) or the actual readers may change; any applications using MDI-SMURF can be oblivious to such changes.

By successively correcting all challenges associated with the physical-digital divide facing RFID-based applications, MDI-SMURF provides Metaphysical Data Independence for a wide array of emerging applications. As a result, RFID applications are less complex and easier to manage and maintain.

5.6 Chapter Summary

One of the major challenges in crossing the physical-digital divide to provide MDI to applications is unreliable data. To alleviate this challenge for RFID data, I developed SMURF, an adaptive tool for RFID data cleaning. SMURF is a smoothing filter designed to correct for missed readings through a novel statistical framework. In a detailed experimental study, SMURF was shown to provide significantly cleaner data in a wide range of scenarios.

To demonstrate how to build an infrastructure that supports MDI, I detailed MDI-SMURF, an infrastructure for providing RFID data to applications. MDI-SMURF accepts raw, unreliable RFID readings that are mostly unusable by applications, and processes them through a series of cleaning stages, including the smoothing filter

discussed above. MDI-SMURF then produces clean data streams at the application's semantic level.

RFID data has the potential to provide incredibly detailed insight into processes in the physical world for applications such as Total Consumer Awareness. SMURF and MDI-SMURF provide mechanisms by which such data can be utilized by these applications.

Chapter 6

A Declarative Framework for Sensor Data Processing

The work presented in the previous two chapters described an interface and mechanism for pay-as-you-go cleaning and integrating of sensor data for use in dataspace applications. To provide a general framework for incorporating these techniques into a dataspace system, I develop *Extensible Sensor stream Processing* (*ESP*).[1] ESP generalizes the MDI-SMURF approach to providing MDI in a dataspace system: it is a modular framework for building data cleaning pipelines focused on streaming data produced by sensor devices. The primary goal of ESP is to enable pay-as-you-go evolution of sensor data processing infrastructures. ESP is built using declarative query processing as found in relational database query languages to enable rapid deployment of cleaning solutions with little up-front cost. Over time, ESP can be incrementally upgraded due to its modular nature: individual modules can be enhanced through declarative means or by embedding more complex techniques such as those used in SMURF.

[1]Portions of the work presented in this chapter have been published in [JAF+06a, JAF+06b, SJFW06].

In this chapter, I introduce ESP and its declarative framework for building sensor data processing infrastructures in a pay-as-you-go manner. I then demonstrate its effectiveness and ease of use through three real-world studies involving RFID in a retail scenario, wireless sensor networks in environmental monitoring, and multiple types of sensors in a pervasive application. Finally, I describe how ESP tracks data quality estimates through its pipeline.

6.1 Introduction

To incorporate sensor data into dataspace applications such as TCA, data from the physical world must be captured through imperfect sensing devices and translated into MDI data understandable by such applications. This process, which I refer to as crossing the physical-digital divide, presents many issues (as described in Chapter 4): the data are unreliable, mismatched in temporal and spatial granularity, semantically low-level compared to what applications desire, and display a high degree of variability. A primary challenge in crossing this divide is building data processing infrastructures to alleviate these issues. In this chapter, I present *Extensible Sensor stream Processing* (ESP), a pay-as-you-go based framework for building infrastructures to translate sensor data into MDI data. The primary goal of ESP is to enable pay-as-you-go deployment and evolution of sensor data processing infrastructures. First, ESP is built using declarative query processing as found in relational database query languages to enable rapid deployment of cleaning solutions with little up-front cost. Second, ESP adopts a modular framework that can be incrementally upgraded over time: individual modules can be enhanced through declarative means or by embedding more complex techniques such as those used in SMURF.

To provide a simple and flexible means of programming sensor data processing infrastructures, ESP uses declarative processing based on relational query lan-

guages and exploits recent advances in relational processing techniques for data streams [ACC+03, BBD+02, CCD+03]. Programmers specify processing stages in ESP using high-level declarative queries over relational data streams[2]; the system then translates the queries into the appropriate low-level operations necessary to produce their results. Thus, programmers do not have to write low-level device interaction code (e.g., nesC for TinyOS [GLvB+03]). Additionally, declarative languages provide data independence, such that in many cases cleaning operations do not need to be changed when devices are added, removed, or upgraded. As an example of a declarative query for sensor data cleaning, consider Query 2 which fills in lost temperature readings from a wireless sensor network using a 5 second moving average over each sensor's readings.

Query 2 *Example declarative query to interpolate for lost sensor readings. This query runs a 5 second moving average over each sensor's readings.*

```
SELECT node_id, AVG(temperature)
FROM sensor_readings_stream [RANGE '5 sec']
GROUP BY node_id
```

ESP utilizes the temporal and spatial nature of sensor data to drive many of its processes. Sensor data tend to be correlated in both time and space; the readings observed at one time instant are indicative of the readings observed at the next time instant, as are readings at nearby devices. Thus, I introduce the concepts of *temporal* and *spatial granule* to capture these correlations. These granules define a unit of time and space inside which the data are mostly homogeneous. These abstractions can be used to recover lost readings or remove outliers using temporal and spatial aggregation.

The ESP framework segments its data processing into five programmable stages,

[2]In ESP, I use CQL [ABW06] as the declarative language as I use a data stream system, TelegraphCQ [CCD+03], designed to process CQL. In principle, any declarative language would provide the benefits outlined here.

each responsible for a different logical aspect of the data, ranging from operations on individual readings to operations involving complex processing across multiple devices and outside data sources.

In this chapter, I first detail ESP and how it alleviates the challenges associated with physical-digital divide using pay-as-you-go principles (Section 6.2). I then ground ESP in three real-world scenarios that illustrate how ESP is used and demonstrate its effectiveness (Sections 6.3, 6.4, and 6.5). Finally, I outline how ESP can be augmented with a quality tracking pipeline that continually estimates the closeness of the data produced by ESP to reality (Section 6.6).

6.2 ESP's Declarative Cleaning Framework

In this section, I introduce *Extensible Sensor stream Processing* (*ESP*), a declarative pipelined framework for building sensor data cleaning infrastructures. ESP is designed with the pay-as-you-go principle in mind to address the challenges associated with the physical-digital divide.

ESP enables infrastructures that clean and integrate raw physical sensor data by processing multiple sensor streams, exploiting the temporal and spatial aspects of sensor data, to produce a single, improved output stream that can be used directly by applications. I first define the temporal and spatial abstractions that drive many of ESP's cleaning mechanisms.

6.2.1 Temporal and Spatial Granules

ESP uses high-level abstractions called *temporal* and *spatial granules* to capture time and space in sensor-based applications. These granules define units of time and space inside which the data are expected to be homogeneous. ESP uses the granule

concept to aggregate, sample, and detect outliers. These abstractions exploit the fact that many applications are not interested in individual readings or devices, but with higher-level data in time and space.

Temporal Granules

Although many sensor devices can produce data at frequent intervals, applications are usually concerned with data from a larger time period, or *temporal granule*. For instance, an environmental monitoring application that builds models of micro-climates in a redwood tree needs readings at 5 minute intervals to capture environmental variations [TPS+05]. Within a temporal granule, readings are expected to be largely homogeneous.

To support this notion of temporal granules, ESP uses *windowed* processing to group readings. A window defines a finite set of readings (in terms of an interval of time) within a data stream. Within a window, ESP can aggregate multiple readings into one or compare readings to detect outliers.

Spatial Granules

Just as with readings in time, readings from devices physically close to each other are expected to be mostly homogeneous; a spatial granule defines the unit of space in which this homogeneity is expected to hold. Furthermore, a spatial granule is the smallest unit of space in which an application is interested, even though devices may have a finer spatial granularity. Examples of spatial granules include a shelf in a library scenario or a room in a digital home application.

To support spatial granules, ESP organizes sensors into *proximity groups*. A proximity group defines a set of sensors of the same type monitoring the same spatial granule. For instance, a set of motes monitoring the temperature in the same room

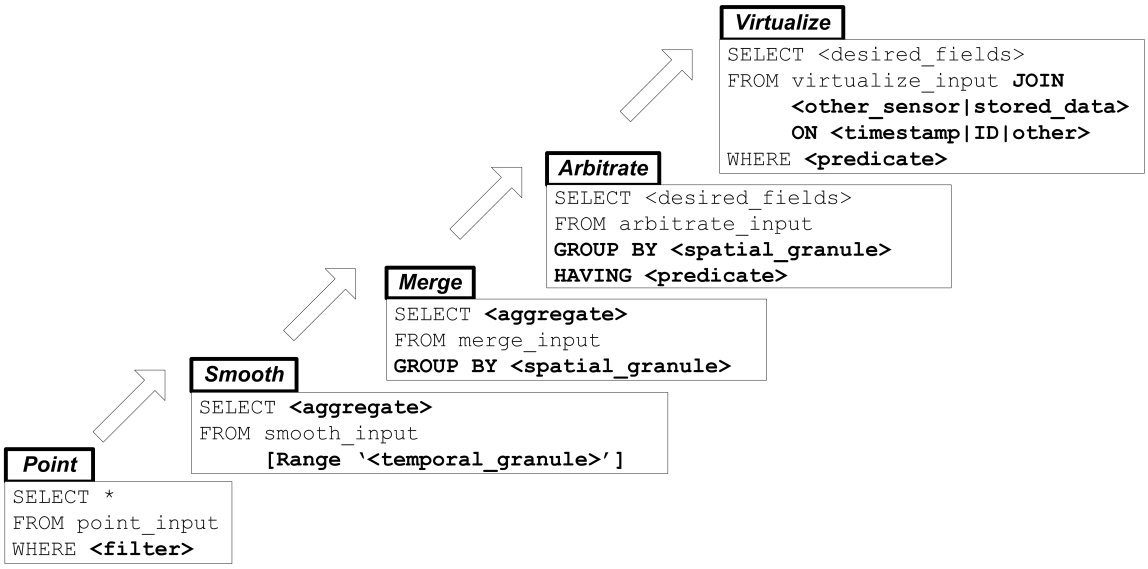


Figure 6.1. ESP processing stages with the typical form of the declarative query for each stage. The relevant portion of the query is in boldface

may be grouped into the same proximity group, as may two RFID readers monitoring the same library shelf. Just as a time window is the unit of processing for a temporal granule, a proximity group is the processing unit for a spatial granule.

In many applications, the size of the temporal and spatial granules are obvious from the nature of the application or environment (e.g., 5 minute intervals in redwood monitoring or rooms in a digital home). In some cases, however, it may be desirable to determine the granule sizes automatically as described in Chapter 5.

6.2.2 ESP Cleaning Stages

Having described the fundamental abstractions underlying ESP, I now outline ESP's processing stages. Through an analysis of typical sensor-based applications, I distilled a set of logically distinct operations that occur in a large class of applications to clean data produced by many types of sensor devices. Using these observations, ESP organizes sensor stream processing into a cascade of five programmable stages: *Point* - *Smooth* - *Merge* - *Arbitrate* - *Virtualize*. These stages operate on different

aspects of the data, from finest (single readings) to coarsest (readings from multiple sensors and other data sources). Not all stages are necessary for a given deployment.

Stage 1, *Point*: The *Point* stage operates over a single value in a sensor stream. The primary purpose of this stage is to filter individual values (e.g., errant RFID tags or obvious outliers) or to convert fields within an individual tuple. The general form for the *Point* query (as well as all other stages) is shown in Fig. 6.1. ESP applies the *Point* query to each sensor's readings, filtering all readings that do not match a predicate.

Stage 2, *Smooth*: In *Smooth*, ESP uses the temporal granule defined by the application to correct for missed readings and to detect outliers in a single sensor stream. The *Smooth* query processes its input stream, `smooth_input` (a stream of readings from a single device, provided by ESP), in windows of readings determined by the size of the temporal granule. For each of these windows, *Smooth* runs the specified aggregate function, outputs a processed reading, and then advances the window by one input reading. Note that both *Point* and *Smooth* operations can be pushed down to capable sensor devices (e.g., wireless motes).

Note that the functionality of *Smooth* is similar to SMURF's temporal smoothing filter; due to ESP's modular nature, SMURF-like techniques could be used in this stage if more advanced functionality is desired.

Stage 3, *Merge*: Analogous to the temporal processing in the *Smooth* stage, *Merge* uses the application's spatial granule to correct for missed readings and remove outliers spatially. At each time step, *Merge* processes input readings from a single type of device and groups the readings by the specified spatial granule using the `GROUP BY` clause. *Merge* then processes each of these groups using an aggregate function to produce output readings for each spatial granule.

Stage 4, *Arbitrate*: Spatial granules may not map directly to sensor detection fields, leading to possible conflicts between the readings from different proximity groups that are physically close to one another. The *Arbitrate* stage deals with conflicts, such as duplicate readings, between data streams from different spatial granules. The query for *Arbitrate* groups its input stream by spatial granule and then uses the `HAVING` clause to filter readings from spatial granules that do not match a predicate.

Stage 5, *Virtualize*: Finally, some types of data cleaning utilize readings from across different types of sensors or stored data for improved data cleaning. To provide a platform for such techniques, the *Virtualize* stage combines readings from different types of devices and different spatial granules. The *Virtualize* query uses the `JOIN` construct to combine readings from different sources based on timestamps, IDs, or other common attributes. Additional processing can be specified using an optional predicate. Furthermore, this stage provides the final support necessary to convert sensor data into MDI data using joins with static data sources (e.g., mapping from EPC ID to product ID and name).

By separating sensor data cleaning into distinct stages, cleaning pipelines are easy to deploy and configure, affording many opportunities to reuse stages from previous deployments with changes localized to individual stages. Additionally, the cleaned data produced by ESP pipelines can be shared across many applications.

In the next three sections, I show detailed ESP processing and demonstrate ESP's overall effectiveness and ease of configuration with three typical sensor deployments.

6.3 RFID-based Scenario

Here I present an example of sensor data cleaning based on ESP through a retail scenario using RFID technology, similar to the one introduced in Section 6.1. Such

technology is notoriously error-prone: tags that exist are frequently missed while other tags that are not in a reader's normal view are sometimes read. ESP can be used to build a cleaning pipeline to correct for these errors.

6.3.1 ESP Cleaning of RFID Data

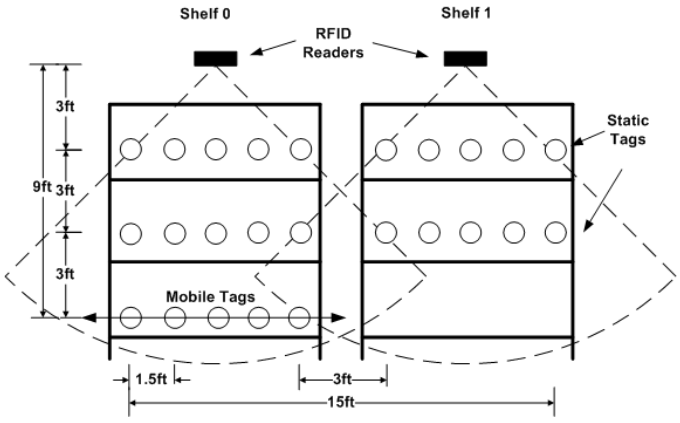


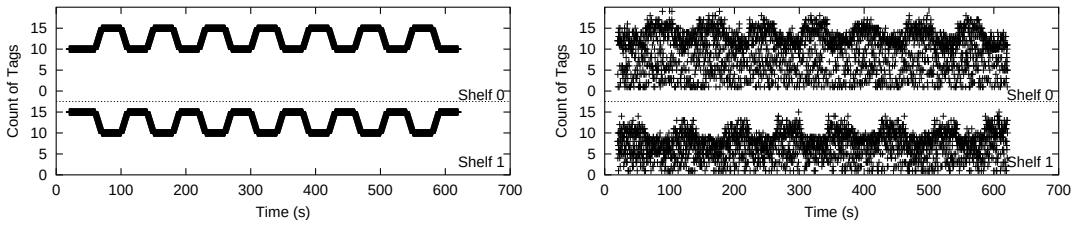
Figure 6.2. Shelf scenario setup with 2 shelves, each with an RFID reader and 10 tags statically placed within 6 feet of the antenna (5 tags at 3 feet, 5 tags at 6 feet). Additionally, 5 tags were relocated every 40 seconds.

Consider an application that continuously monitors the count of items on each shelf using Query 3 (shown below). This query looks at the stream of RFID data in 5-second slices. Within each of these slices, the query groups the readings by the shelf at which the tag was read, and then counts the number of distinct tag IDs at each shelf. Here, the window clause indicates the temporal granule (5 seconds) and the `GROUP BY` clause denotes the spatial granule (a shelf).

Query 3 *Shelf monitoring query to determine the number of products on each shelf.*

```
SELECT shelf, COUNT(distinct tag_id) AS num_products
FROM rfid_data [RANGE '5 sec']
GROUP BY shelf
```

To study ESP used for cleaning RFID data, I ran an experiment emulating a retail scenario. The experimental setup is depicted in Fig. 6.2. The setup consists of



(a) Reality

(b) Query 3 results using raw RFID data

Figure 6.3. Query 3 results in reality and over the raw data.

two 915 MHz RFID readers from Alien Technology [ALR05], each responsible for one shelf and thus each forming a proximity group. The readers' sample period (epoch size) was set at 5Hz (i.e., 5 interrogation cycles per second). Each shelf was stocked with 10 items represented with Alien "I2" tags [Ali05a], EPC Class 1 RFID tags designed for long-range detection in a controlled environment. Tags were suspended in the same plane as the reader, spaced 1.5 feet apart from each other, and at two distances from the reader, 3 feet and 6 feet. Tags were oriented such that their antennae were directly facing the reader. Note that this setup is overly favorable to RFID technology as it attempts to alleviate many of the known causes of degraded readings [FJPR04, FL04]. To introduce a dynamic component into the experiment, 5 tags placed 9 feet from the reader were relocated between the two shelves every 40 seconds.

The metric used to evaluate ESP's cleaning is the average relative error of the results of Query 3, which is defined as $\frac{1}{N}\sum_{i=0}^{N}(\frac{|R_i - T_i|}{T_i})$, where N is total number of time steps (epochs), i is the time step at the granularity of the reader (5Hz), R_i is the reported count of tags on a shelf at time i, and T_i is the true count of tags on a shelf at i. This metric denotes how far off, on average, the reported count of tags is from reality. The experiment was performed three times; all runs produced similar results.

The results of the experiment without data cleaning are shown in Fig. 6.3. Fig-

ure 6.3(a) depicts the trace of the actual count of tags on each shelf over the course of the experiment. Figure 6.3(b) shows the results of running the Query 3 over the raw data. If the application were to use the output of the RFID readers directly, the results would be near-meaningless: the average relative error of the output of Query 3 compared to reality for the duration of the experiment was 0.41 (i.e., the count of the number of tags on each shelf was off by almost half, on average). For instance, if an application wants to be notified when the number of items on a shelf drops below 5, then the query using the raw data would report that a shelf has low inventory 2.3 times per second, on average.

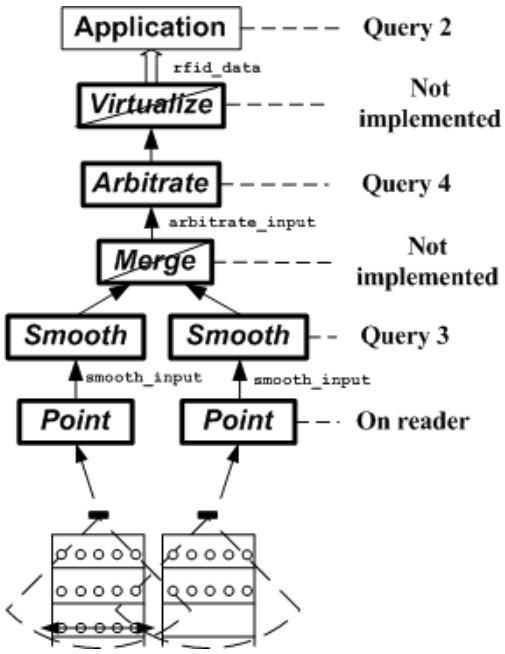
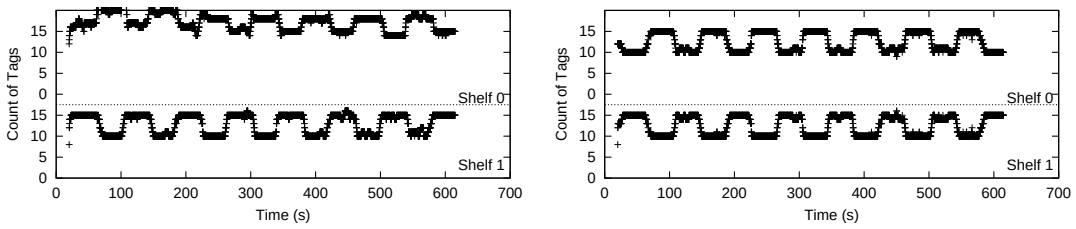


Figure 6.4. ESP pipeline for cleaning RFID data.

An ESP pipeline can be used to clean this data, as shown in Fig. 6.4. In this case, the *Smooth* and *Arbitrate* stages for ESP are used; the RFID reader already provides *Point* functionality natively by removing tags that fail a checksum [Ali05b] and since there is only one sensor per proximity group, *Merge* is not needed.



(a) Query 3 results after *Smooth* processing

(b) Query 3 results after *Arbitrate* processing

Figure 6.5. Query 3 results after different stages of processing.

6.3.2 Stage 2: *Smooth*

At the *Smooth* stage (shown in Query 4), ESP interpolates for lost readings within a temporal granule. ESP runs this query over each reader's data stream. This query begins by breaking the stream into 5-second slices (corresponding to the size of the temporal granule). For each of these slices, *Smooth* groups by tag ID and then counts the number of occurrences for that tag. The output of *Smooth*, then, is a reading for each tag seen at any point within the window and the number of times it was read. After each window is processed, ESP moves the window forward by one input reading. Through this sliding window operation, *Smooth* fills in dropped readings for any tag seen at least once in a 5 second time period. Note that this stage is a static version of the windowed-smoothing approach used by SMURF (discussed in Chapter 5).

Query 4 *Interpolating for lost readings in the* Smooth *stage.*

```
SELECT tag_id, COUNT(*)
FROM smooth_input [RANGE '5 sec']
GROUP BY tag_id
```

The results of Query 3 over the data produced by this stage are shown in Fig. 6.5(a). The *Smooth* stage is able to eliminate the constant low inventory alerts generated by the query using the raw data.

The count of items per shelf, however, is still fairly inaccurate (an average relative error of 0.24) due to the close proximity of the readers and discrepancies in their

performance. As seen in Fig. 6.5(a), the antenna for shelf 0 read more tags than that of shelf 1, despite being of the same model; the counts reported for shelf 0 were consistently 4 to 5 tags higher than reality. Different configurations of antennae produced similar results; this difference is likely due to known issues with the antenna ports on these particular RFID readers [Ali05c]. Processing in the *Smooth* stage has alleviated the issues with dropped readings, but any application using this data will be misled into thinking that shelf 0 has extra items.

6.3.3 Stage 4: *Arbitrate*

The *Arbitrate* stage (shown in Query 5) corrects for duplicate readings caused by the close proximity of the readers. At each time step, *Arbitrate* determines all tags that were read by multiple spatial granules and the number of times each tag was read by each granule. It then assigns the tag to the spatial granule that read the tag the most. ESP runs *Arbitrate* over the union of the streams produced by Query 4.[3]

Query 5 *Correcting for duplicate readings in the* Arbitrate *stage. The inner query determines the count of readings for a given tag in each spatial granule; the outer query selects the spatial granule with the highest count for each tag.*

```
SELECT spatial_granule, tag_id
FROM arbitrate_input ai1 [RANGE 'NOW']
GROUP BY spatial_granule, tag_id
HAVING COUNT(*) >= ALL(SELECT COUNT(*)
                       FROM arbitrate_input ai2
                       [RANGE 'NOW']
                       WHERE ai1.tag_id = ai2.tag_id
                       GROUP BY spatial_granule)
```

The results of running Query 3 over the smoothed and arbitrated data are shown in Fig. 6.5(b). Observe that ESP de-duplicates the readings as well as corrects for the

[3]Although the *Merge* stage is unused in this case, ESP automatically adds a `spatial_granule` attribute to each stream, corresponding to each proximity group (i.e., each shelf).

differing performance of the two antennae to provide a substantially more accurate count of the tags on each shelf. After *Arbitrate* processing, the average relative error of Query 3 is 0.04. This equates to an error of being off by less than one item, on average. The results show that in this scenario, ESP provides a significant reduction in error over the raw RFID data: recall that the original item counts using the raw data were off by almost half compared to reality.

6.4 Environment Monitoring Scenario

In the previous section, I demonstrated the ability of an ESP pipeline to clean RFID data streams. Next, I present a use case where ESP hides the unreliabilities of wireless sensor networks.

Wireless sensor networks enable new classes of pervasive applications that monitor environments such as the home or animal habitats with high resolution. In order to alleviate the effects of imprecise readings, calibration errors, outliers, and unreliable network communication, previous deployments involving sensor networks have had to post-process the readings, primarily by hand, to produce data that can be used by the application [BGH+05, DR05, DGM+04]. To reduce the complexity associated with sensor network application deployment, applications can use ESP to provide cleaned sensor data. I demonstrate two types of wireless sensor network data cleaning: outlier detection of fail-dirty motes, and temporal and spatial smoothing to correct for dropped messages.

6.4.1 Outlier Detection

Recall that sensor motes are known to "fail-dirty" and produce outlier readings. ESP can be used to alleviate the effects of these fail-dirty motes. To demonstrate

the effectiveness of outlier detection using ESP, I use a 2 day trace from a sensor network deployed in the Intel Research Lab in Berkeley to monitor the lab's environment [Lab05]. I focus on three motes in the same room, assigned to the same proximity group. In this trace, one of the motes fails by reporting increasing temperatures, rising to over 100^{o}C. I implement the *Point* and *Merge* stages of ESP to eliminate the outlier readings. *Smooth* is not used because it cannot correct for extended errors produced by one sensor.[4] *Arbitrate* is not necessary as there is only one spatial granule.

Stage 1: *Point*

The *Point* stage filters any readings beyond its expected range; in this case, ESP filters readings where the temperature is higher than 50^{o}C (Query 6) .

Query 6 *Simple filtering at the* Point *stage.*

```
SELECT *
FROM point_input
WHERE temperature < 50
```

Stage 3: *Merge*

In this example, the *Merge* stage does outlier detection within a spatial granule by computing the average of the readings from different motes in the same proximity group and then omitting individual readings that are outside of two standard deviations from the mean (shown in Query 7). Note that these techniques are not intended to be statistically complex, but to the contrary, demonstrate the extent to which relatively simple techniques can be effectively used in the ESP framework.

Figure 6.6 shows the outcome of this experiment. The top line represents the

[4] *Smooth* could, however, be used to correct for individual outlier readings in a single mote using the same mechanisms presented here.

Query 7 *Outlier detection in the* Merge *stage.*

```
SELECT spatial_granule, AVG(temp)
FROM merge_input s [RANGE '5 min'],
     (SELECT spatial_granule, AVG(temp) AS average,
                    STDDEV(temp) AS standard_deviation
      FROM merge_input [RANGE '5 min']) AS a
WHERE a.spatial_granule = s.spatial_granule AND
      a.average + (2*a.standard_deviation) < s.temp AND
      a.average - (2*a.standard_deviation) > s.temp
```

outlier mote's readings. The middle line depicts the average of all three motes. If an application were to use the average of the three motes as a representation of the room's temperature, it would see temperatures exceeding 50°C. The bottom lines show the traces of the two functioning motes as well as the output of ESP with outlier detection processing. Observe that ESP is able to detect when the outlier mote begins to deviate from the other motes and then omit its reading from its average calculation.

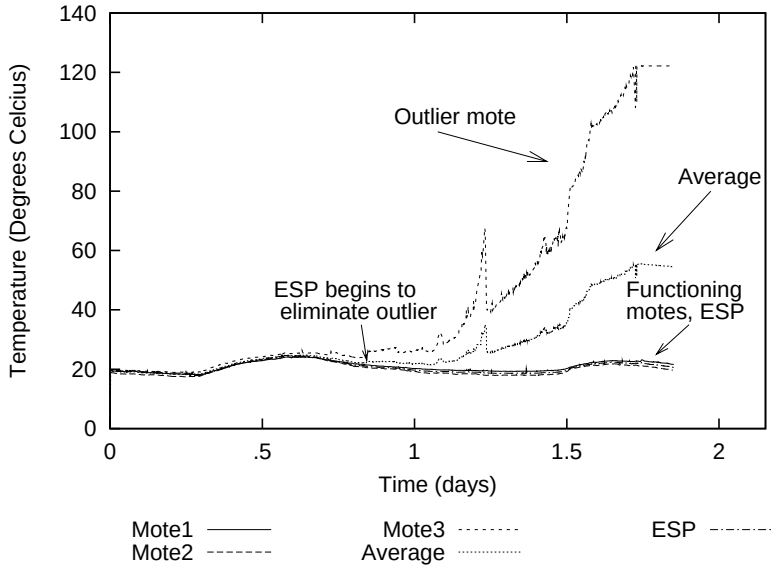


Figure 6.6. Outlier Detection using ESP. The "ESP" line tracks the two functional motes' lines.

6.4.2 Temporal and Spatial Smoothing of Sensor Data

Wireless sensor networks have another serious problem beyond fail-dirty motes: the network frequently drops messages. This problem is especially prevalent when sensor networks are deployed in the real world.

An ESP pipeline for a wireless sensor network can mask the unreliability of a sensor network by both temporally and spatially aggregating to correct for dropped readings. I demonstrate this cleaning through an application responsible for monitoring the temperature of a redwood tree at each elevation range in the tree.

I validated ESP processing on a three and a half day trace of data collected from sensors on a redwood tree in Sonoma County, CA as part of a large-scale sensor network deployment to study micro-climates of redwood trees [Son06]. 33 motes were placed along the trunk of the tree at varying heights. Data (e.g., temperature and humidity) were sensed at 5 minute intervals and logged to a local storage buffer (collected at the end of the experiment) and also sent over the multi-hop network. I grouped the motes at nearby heights into 2-node, non-overlapping proximity groups (corresponding to the spatial granules in this deployment), where the distance between motes in a proximity group was less than 1 foot.

Note that the log data is incorrect with respect to the ground truth due to fail-dirty sensors: 8 out of the 33 motes failed dirty. The readings from these motes were removed by hand shortly after data collection, but before I received the data.[5]

As ESP is addressing communication errors in this case, the metric of success is the epoch yield. Epoch yield describes the number of the readings reported to the application as a fraction of the total number of readings the application requested. For the raw data, the epoch yield in this trace was 40% (ideally, the epoch yield should be 100%). In other words, the application only received 40% of the data it requested.

[5]ESP could employ the techniques shown in Section 6.4.1 to remove these outliers automatically.

Additionally, I measure the percent error in the readings. For this application, an error of less than 1^oC is typically acceptable for trend analysis. Therefore, the goal of ESP in this application should be to increase the epoch yield while minimizing the percent of readings with an error greater than 1^oC.

Here, the *Smooth* and *Merge* stages in ESP temporally and spatially aggregate sensor readings to increase the epoch yield of the sensor deployment.

Stage 2: *Smooth*

In the *Smooth* stage, ESP temporally aggregates readings from a single sensor. By running a sliding window average on each sensor stream, lost readings from a single mote are masked within the window. After the *Smooth* stage, the epoch yield is increased to 77%. 99% of these readings were within 1^oC of the logged data.

Stage 3: *Merge*

In the *Merge* stage, ESP performs spatial aggregation for each spatial granule (again, in the form of a windowed average) to further alleviate the effects of lost readings. The *Merge* stage increases the epoch yield to 92%. This improvement of reporting is at the slight cost of decreasing the percent of readings within 1^oC of the logged data to 94%. Thus, with ESP cleaning, biologists can get nearly complete data with a slight decrease in the accuracy.

Through the use of simple outlier detection and temporal and spatial smoothing, in this case an ESP pipeline is able to increase the ability of applications to make sense of the data they are getting from their sensors. Rather than spending time tediously post-processing the data, applications can focus on the high-level logic, rather than conversion, calibration, and error correction.

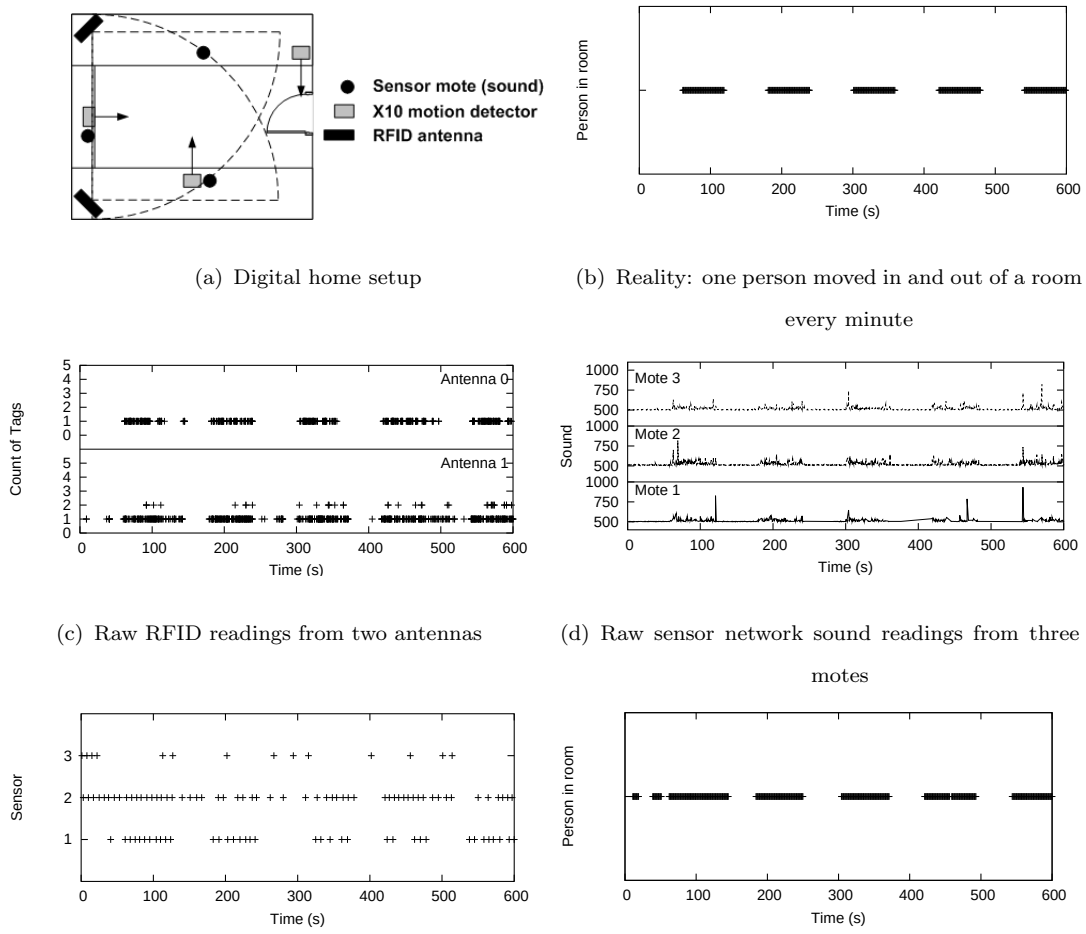
6.5 Digital Home Scenario

In Sects. 6.3 and 6.4, I demonstrated how ESP can provide a cleaning infrastructure to correct for a wide variety of problems associated with different physical devices. Next, I demonstrate the ease of configuration of ESP and highlight the use of multiple types of sensors to enhance data cleaning. Furthermore, this section shows how ESP can be used to provide Metaphysical Data Independence.

Multiple projects are developing sensors and infrastructures to instrument the home to provide both a better living experience for inhabitants as well as a more efficient use of home resources [MIT06, KOA+99]. Such applications use a wide variety of sensor devices providing low-level data (e.g., RFID, sensor motes, pressure sensors). In this section, I show that pipelines defined for other deployments (i.e., pipelines from the previous two sections) can be easily re-tasked to a new environment due to ESP's high-level declarative nature. Furthermore, ESP can serve as a platform for combining readings from multiple devices to provide a virtual "person detector" sensor. This type of processing is a higher level of cleaning; data from multiple heterogeneous devices, appropriately combined, can provide higher quality data. The output of ESP is a stream of MDI-level events describing the presence of a person in the room.

I demonstrate the use of ESP in a digital home scenario by outfitting a room with two RFID readers, a small sensor network of three motes, and three X10 motion detectors [X1005] tasked to determine when someone is in the room (Fig. 6.7(a)). The room corresponds to one spatial granule for the application; thus, the two RFID readers make up one proximity group, the motes constitute another, and the X10 detectors form a third. During the experiment, one person, outfitted with an RFID tag, moved in and out of the room, while talking, at one minute intervals (Fig. 6.7(b)). The raw data from the sensors are presented in Figs. 6.7(c)- 6.7(e)

I present the ESP processing to clean the individual sensor streams and then describe how ESP utilizes these streams to create a person detector.



(a) Digital home setup

(b) Reality: one person moved in and out of a room every minute

(c) Raw RFID readings from two antennas

(d) Raw sensor network sound readings from three motes

(e) Raw X10 motion detectors. A mark indicates that the device reported movement

(f) Data after ESP processing

Figure 6.7. A "Person Detector" in the digital home.

6.5.1 Low-Level Sensor Cleaning

Recall the main advantages of ESP's declarative pipelined approach: previously built stages can be reused, changes necessary to tailor processing to each new deployment are isolated to small logical units, and these changes are easy to make and reason about as the stages are expressed as high-level queries. In this deployment,

the ESP pipelines to clean the individual sensor streams (RFID, wireless sensors, and motion detectors) utilize almost exactly the same processing stages as defined in the previous two sections. Changes necessary for this deployment involved slightly modifying queries in a small number of stages.

Stage 5: *Virtualize*

The main new feature of this use case (as compared to the previous scenarios) is the use of the *Virtualize* stage. *Virtualize* allows a deployment to combine readings from multiple different types of devices to perform application-level integration. In this case, *Virtualize* turns the set of heterogeneous devices into a "person detector." It uses a voting query that normalizes all sensor input streams to a single vote of whether or not it has determined that a person is in the room (Query 8). The query then adds up the votes and registers that a person is in the room if the sum is higher than a threshold.

Query 8 *"Person Detector" logic at the* Virtualize *stage.*

```
SELECT 'Person-in-room'
FROM (SELECT 1 AS cnt
      FROM sensors_input [RANGE 'NOW']
      WHERE sensors.noise > 525) AS sensor_count,
     (SELECT 1 AS cnt
      FROM rfid_input [RANGE 'NOW']
      HAVING count(distinct tag_id) > 1)
      AS rfid_count,
     (SELECT 1 AS cnt
      FROM motion_input [RANGE 'NOW']
      WHERE value = 'ON') AS motion_count
WHERE sensor_count.cnt +
      rfid_count.cnt +
      motion_count.cnt >= threshold
```

The output of the ESP pipeline is shown in Fig. 6.7(f). As can be seen in this case, simple and easy to deploy logic is capable of generally approximating reality. ESP is able to correctly indicate that a person is in the room 92% of the time.

Since the data produced by ESP is not perfect, the application must be able to obtain information on quality through a means independent of the underlying devices. In the next section, I describe a set of techniques for estimating the *quality* of a data stream produced by ESP.

6.6 Quality Estimation for Sensor Data Streams

While cleaning sensor data streams can substantially improve the quality of reported readings, no technique can reliably eliminate all errors. I argue that in many cases, it is essential for an application to be made aware of the level of reliability of its input streams, so that it can make appropriate decisions. For instance, a digital home application acting on unreliable data produced by a mediocre cleaning tool may sporadically turn on and off the house's lights. With an accurate estimate of the quality of the data stream, such an application could threshold its decision-making on the quality of the data, thus "calming" the application. As another example, a TCA application could use quality information to ensure that consumers only received high-quality data on which to make their decisions.

The goal of this section is to develop techniques that can be used to provide applications with useful information about the quality of the cleaned data streams. Here, I introduce a *quality tracking pipeline* that consists of a set of quality estimation modules that shadow ESP's main data cleaning pipeline, as shown in Figure 6.8. Each module in this shadow pipeline is associated with a particular module in ESP and tracks the quality of the data as the sensor streams are cleaned.

I develop these quality tracking algorithms in the context of *object-detection applications*: applications that rely on continuous detection of real-world objects. The person detector application in the previous section is an ODA; other example

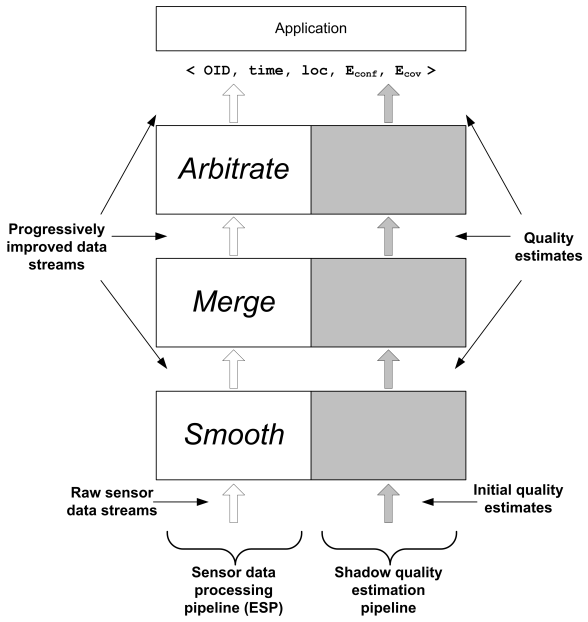


Figure 6.8. Example cleaning pipeline including the shadow quality estimation pipeline.

ODAs include military tracking, monitoring inventory, and tracking wildlife [CKRS04, HKL+06, Hog06, Mar06, SOS05].

I derive online algorithms for the quality tracking modules that provide accurate measures of quality as the ODA streams flow through the cleaning process. These modules track two measures of quality: *confidence* and *coverage* [Wid05]. Confidence, a value associated with individual readings, tracks the estimated probability of a reading in the data stream being correct, i.e., the probability of the reported object actually being present at the given location and time. Coverage, a value associated with a window of readings, tracks the estimated fraction of present objects that are accounted for within a window of the data stream. While there may be many possible instantiations for these modules, an experimental evaluation show that that this approach produces good results.

Applications can make direct use of confidence or coverage. For example, in the above digital home scenario, the application may want to turn on the lights only when the estimated confidence that someone is in the room exceeds a threshold.

In an animal tracking application reporting numbers of animals present in specific regions, counts derived from the data stream may be scaled up based on estimated coverage.

6.6.1 Data Model and Cleaning Techniques

For the purposes of this section, I simplify ESP's model and cleaning techniques for use in ODAs. In such applications, the data stream can be abstracted for cleaning and quality estimation purposes to the schema `<oid,time,loc>`. `oid`'s identify the *objects* in the application (such as a person or an animal), whose presence are reported by sensors (such as RFID readers, wireless sensors, or image analysis). As before, each sensor is associated with a spatial granule in which it detects objects; `loc` identifies this granule (not the actual spatial coordinates) of the object. The temporal granule is denoted as `time`. Since there could be multiple sensors associated with a spatial granule, there could be duplicate `<oid,time,loc>` values in the stream.

In this section, I focus on three ESP stages: *Smooth*, *Merge*, *Arbitrate*. Each stage is augmented with a shadow quality estimation stage that tracks the quality of the readings produced by that stage.

Smooth: For applications that track objects, *Smooth* emits an output reading at time t if there is at least one input reading in a window of size W ending at t. Formally, a reading `<o,t,l>` is output after temporal smoothing if there exists a reading `<o,t',l>` in the input data stream with $(t - W) \leq t' \leq t$. Notice that temporal smoothing does not eliminate any readings in the input stream, but only adds additional readings.

Merge: In these applications, *Merge* aggregates multiple streams from the same spatial granule as follows: the output of *Merge* contains a reading for an object if any one of the sensors in the granule reports it. Formally, a reading `<o,t,l>` is output

after *Merge* if there exists a reading `<o,t,l>` in the input of at least one of the sensors in the spatial granule. (The presence of sensor IDs is assumed.)

Arbitrate: The objective of *Arbitrate* is to identify readings of the form `<o,t,l1>`, `<o,t,l2>` that report the same object at multiple different locations (i.e., spatial granules) at the same time. Since these conflicts are usually resolved using application-level logic (e.g., in a *Virtualize* stage), this cleaning module simply identifies such conflicting readings and marks them for later resolution.

6.6.2 Quality of ODA Data Streams

In this section, I define the quality of an ODA data stream as the difference between the reported data and reality.

Conceptually, the *reality stream* is a virtual stream consisting of readings at each time interval for all objects present in each of the application's spatial granules. This stream represents physical reality: it is the stream that the ODA sensors would produce if the technology were perfect (i.e., if sensors reported all objects present in their spatial granule at each time instant). At a high level, the *quality* of an ODA stream measures the deviation (in terms of false positives and negatives) of the data stream from the reality stream.

I use two components for an intuitive notion of quality —*confidence* and *coverage*— to capture how closely the data stream resembles the reality stream. Confidence accounts for objects that are not present in reality but are reported, and coverage accounts for objects that are present in reality but are not reported. I first define confidence and coverage formally, then discuss estimating these values in an online ODA cleaning pipeline.

Definitions of Confidence and Coverage

Consider the reality stream R, and an ODA data stream D whose quality we want to measure after the application of zero or more cleaning modules. Given R and D, I define: (1) True Positives, *TP*, as the set of all readings that are present in R as well as D; (2) False Positives, *FP*, as the set of all readings that are present in D but not in R; (3) False Negatives, *FN*, as the set of all readings present in R but not in D. Confidence and coverage are defined as follows:

Definition 6.6.1 (Confidence) *Confidence is a per-reading value that gives the probability that the reading exists in the reality stream (i.e., that the object is present in reality at the given time and location). Ideally, confidence is* 1 *for all readings in* TP *and* 0 *for all readings in* FP*. This metric naturally extends to the entire data stream D, or any subset of readings: The confidence for a set of readings is the average of the confidences of the individual readings.*

Definition 6.6.2 (Coverage) *Coverage is a window-level value assigned to a set of readings for a given time period T. It gives the fraction of readings from R in the time period T that are present in D. Hence, for the entire stream, or for any time-window having associated values of* TP*,* FP *and* FN*, Coverage* $= \frac{|\mathrm{TP}|}{|\mathrm{TP} \cup \mathrm{FN}|}$.

Quality Estimation

Since the exact values for *TP*, *FP* and *FN* are not known, the exact confidence and coverage of a cleaned data stream cannot be determined; thus, the goal is to provide *estimates* of confidence and coverage. The estimates of confidence and coverage are denoted by E_{conf} and E_{cov}, respectively. Though the ideal confidence of each reading is either 0 or 1, E_{conf} estimates the probability of a reading being correct and ranges between 0 and 1.

The schema of a reading in an ODA stream is augmented with these estimates. Quality estimation for each cleaning module is based on the incoming data stream of readings with the schema <oid,time,loc,E_{conf},E_{cov}>, the output data stream with the same schema, and knowledge of the ESP stage's cleaning mechanism.

To seed the estimation process, initial estimates for E_{conf} and E_{cov} are needed before any cleaning has been performed. Confidence and coverage for a raw data stream usually depends on several properties, which are heavily dependent on the sensors, objects, and the particular ODA deployment; thus, there are no general techniques for initial quality estimation of an ODA stream. There are, however, a wide variety of techniques that are applicable. For instance, work on building models of sensor error characteristics and detection fields [DGM+04, FGB02] can be used to infer initial estimates.

As an example of how to derive an initial confidence, estimated confidence for an RFID reading can be estimated as follows. Initially, an RFID middleware system receives a tag list from an RFID reader. As previously mentioned, each tag_id in the tag list contains the number of times it has been read in the epoch, which can be used to calculate the *read rate*. The read rate for each tag t is the ratio of responses to total interrogations, $\frac{R_t}{I_t}$. This read rate can be used as the initial confidence.

Next I look at each of the cleaning modules individually and show how these techniques estimate the quality of the resulting stream. I focus in the remainder of this discussion on confidence calculation; see [SJFW06] for a description of coverage calculations.

6.6.3 Quality Calculation for *Smooth* and *Merge*

Recall that *Smooth* aggregates expected readings across time, taking advantage of the redundancy of readings in time. *Merge*, on the other hand, spatially aggre-

gates streams from multiple sensors that are monitoring the same spatial granule; it combines readings from individual unreliable sensors to produce a single data stream with higher quality.

Both stages operate over a window of readings of width W and thus the two can be treated similarly for quality estimation purposes. It is assumed that windows slide by 1 epoch, although this approach applies directly to other slide granularities. *Smooth* uses a time window (temporal granule) of W_t epochs; *Merge* uses a *spatial window* consisting of the readings from W_s sensors in a spatial granule for the same epoch. The output of either stage is a reading for each window (i.e., a reading at each time-unit) if there exists at least one reading within the window.

Confidence Calculation

Consider a particular object O in a temporal window (for *Smooth*) or spatial window (for *Merge*) of size W (I use W to represent either W_t or W_s). Let the number of readings of object O in the window be r, and let their confidences be $c_1, c_2, \ldots, c_r$.

The goal of confidence estimation here is to assign a high confidence to an output reading if the corresponding object was actually present in reality, and a low confidence if it was not. E_{conf} is calculated on the intuition that the greater the evidence of an object being present in a window, the greater the output confidence for that reading. Intuitively, two factors contribute to the evidence of the presence of the object within a window: (1) higher individual confidences of the input, denote greater evidence of the presence of an object, and (2) a larger number of input readings denote a greater the evidence of the presence of an object.

While there are potentially many ways to account for the above factors in the output confidence, I use a linear combination of these factors. In Section 6.6.5, I

show empirically that this approach produces accurate confidence estimates. After processing a window with confidences $c_1, c_2, \ldots, c_r$, both quality estimation stages emit a confidence E_{conf} for each object in W as given by Equation 6.1:

$$E_{conf} = \frac{1}{W}[\alpha S + (1-\alpha)r] \tag{6.1}$$

$$\text{where } S = \sum_{i=1}^{r} c_i \text{ and } W \text{ is either } W_t \text{ or } W_s.$$

Note that I use the same equation for both *Smooth* and *Merge*. The weighting factor, α, could be different in the two cases.

6.6.4 Quality Calculation for *Arbitrate*

Arbitrate is designed to detect objects that are reported by sensors monitoring different (but possibly adjacent) spatial granules to the granule in which the object is present. *Arbitrate* attempts to determine which of the conflicting readings are correct (true positives), and which are incorrect (false positives).

I assume that *Arbitrate* does not remove any readings, but only adjusts the confidences. For example, if sensors for two spatial granules both report an object with an equal confidence, it would be desirable to retain both of the readings with lower confidence since they are in conflict with each other. Conflicts can then be resolved at a higher-level such as in *Virtualize* or by a human.

Confidence Calculation

I now show how to revise the confidences of readings in *Arbitrate*. The input to this stage is a set of n data streams, one from each of the application's spatial granules, $L_1, L_2, \ldots, L_n$. Let the input streams contain r ($r \leq n$) conflicting readings

for an object and epoch from different spatial granules. I denote the confidences of these r readings by $c_1, c_2, \ldots, c_r$ respectively.

Since each conflicting reading reduces the likelihood of any other conflicting reading being correct, this stage reduces the confidences of each conflicting reading. The revised confidence c_i^{new} for each reading is estimated based on the following two considerations. First, each additional spatial granule reporting an object reduces the new confidences of each of the outgoing readings of the same object in that epoch. Second, the higher the confidence of conflicting readings, the greater should be the decrease in their updated confidence values. The intuition here is that the higher the confidence of a particular reading, the lower the chance that the object is present in some other spatial granule.

The confidence of outgoing readings is updated by scaling down each confidence using the probability that the object is not present in the other spatial granules. Equation 6.2 shows the formula for the new confidence.

$$c_i^{new} = c_i * \Pi_{j=1, j \neq i}^{j=r}(1 - c_j) \tag{6.2}$$

Since c_i estimates the probability of the object being present in spatial granule L_i, $(1 - c_i)$ estimates the probability of it not being in L_i.

6.6.5 Experimental Evaluation

In this section, I experimentally evaluate the algorithms for tracking quality using an RFID scenario similar to that described in the retail scenario above.

I validate the quality tracking algorithms by measuring the accuracy of the confidence estimates, E_{conf}, denoted as A_{conf}. Intuitively, A_{conf} gives the closeness of the estimate E_{conf} to the actual confidence. I compute A_{conf} as follows. For an

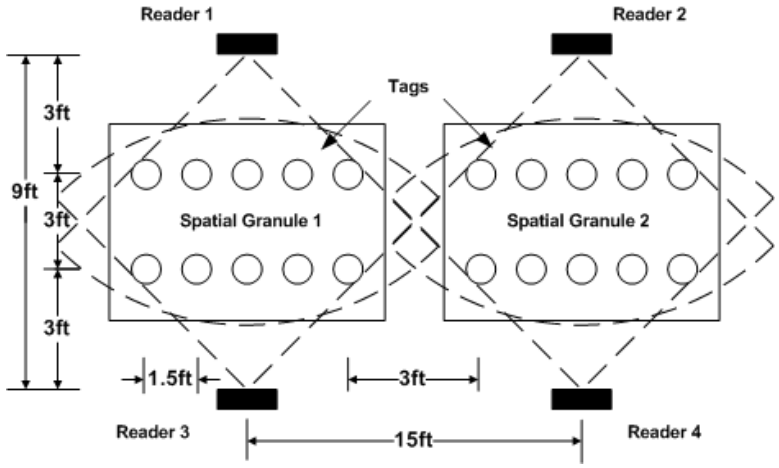


Figure 6.9. Experimental RFID Setup.

incorrect reading, a lower E_{conf} implies higher accuracy and for a correct reading, a higher E_{conf} implies a higher accuracy. Therefore, I define A_{conf} to be $(1-E_{conf})$ for incorrect readings and E_{conf} for correct readings. I define the confidence estimation accuracy of an entire stream D as the average of A_{conf} for all readings in D.

Experimental Setup

These experiments use an RFID data generator that produces data streams based on the setup shown in Figure 6.9. This deployment consists of two spatial granules, each containing 10 tags arranged in two rows separated by 3 feet. Each spatial granule is monitored by 2 RFID readers, placed 3 feet from the first row of tags on either side.

Based on the RFID detection model illustrated in Figure 6.10 and this RFID deployment, the generator produces RFID readings for use in these experiments. Here, I set p to 0.9, d to 8, θ to 45, and s to 0.1. (similar to the model used in [HBF+04]). During each epoch, the simulated readers attempt to detect each tag

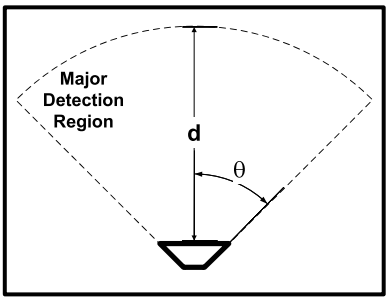


Figure 6.10. Simplified RFID detection model. The probability of detection within the major detection region is p; outside it is s.

using a biased coin-flip based on either p or s (depending on where each tag is located relative to the reader).

This deployment demands all three types of cleaning modules. *Smooth* is needed to account for false negatives produced by the unreliable readers. *Merge* is used to further alleviate the effects of false negatives using multiple readers associated with the same spatial granule. Finally, due to the proximity of the two spatial granules, *Arbitrate* is necessary to remove duplicate readings.

Experimental Results

To investigate the effectiveness of these estimation algorithms, I test the accuracy of the quality estimation for the full pipeline with all three types of cleaning modules. Here, I run *Smooth* over each stream from the four readers, then the streams are spatially aggregated using *Merge*, and finally, conflicts are detected using *Arbitrate*. I then measure the accuracy of the confidence estimates of the resulting stream. In this case, the overall accuracy of the confidence estimation is 0.86. This shows that the confidence estimates are very close to the actual confidence. Hence, low confidences are assigned most often for wrong readings, and high confidences for correct readings.

To further explore how the confidence estimation performs, I plot the distributions for the confidence estimations for both correct readings in the cleaned stream (true

positives) and incorrect readings (false positives). Figure 6.11 shows these distributions.

The graphs show that, as desired, the algorithms for confidence estimation does indeed assign a high confidence for correct readings and very low confidence to false positives. For most readings in the resulting data stream that are correct (i.e., the corresponding object exists in the reported location at the reported time), the estimates assign a high confidence between 0.7 and 1 (Figure 6.11(a)). For incorrect readings in the data stream, the estimates assign a low confidence; most false positives are given a confidence less than 0.1 (Figure 6.11(b)).

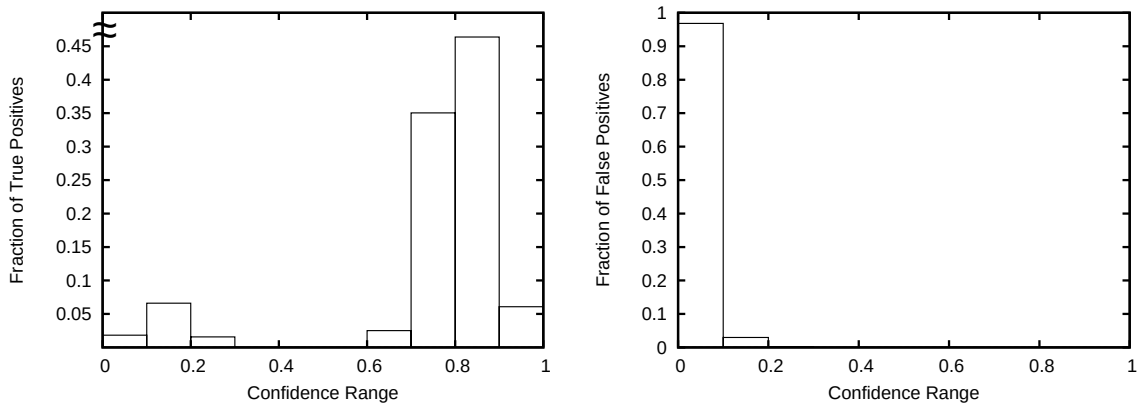


Figure 6.11. Confidence distributions for true/false positives after *Smooth + Merge + Arbitrate*.

Note the cluster of low confidence true positives in Figure 6.11(a). Before arbitration, some false positives were assigned too high a confidence: some false positives received confidences higher than 0.8 and many received a confidence of 0.1-0.2. These high confidence estimates are all reduced after arbitration, where all false positives receive a confidence of 0-0.1. As a result of these conflicting readings, however, arbitration incorrectly lowers the confidence of some correct readings, resulting in the small peak for the low confidence range in Figure 6.11(a). This effect is due to the

fact that arbitration lowers the confidence of readings, overall thus shifting the entire distribution of confidences to the left.

These results show that with simple and intuitive algorithms for tracking quality across multiple processing stages, an infrastructure such as one built using ESP can produce good estimates of the resulting data stream for use in higher-level applications.

6.7 Chapter Summary

In this chapter, I developed ESP, a framework for building pay-as-you-go cleaning and integration infrastructures for sensor data. ESP employs declarative query processing as its means of programming, and as such cleaning infrastructures built using ESP are easy to deploy and maintain over time. Finally, I demonstrated how uncertainty estimates can be attached to the data produced by ESP, thus informing applications of the inherent uncertainty in the data. ESP forms a foundation for building sensor infrastructures to support dataspace applications.

Chapter 7

Conclusions

Through the innovative use of disparate data sources, powerful new applications are emerging in a many areas: Total Consumer Awareness has the potential to revolutionize consumer behavior by exposing supply-chain data and other product information, Web mash-ups correlate multiple data sources to provide new services, and world-wide sensor networks provide unprecedented understanding of the physical world.

In order to realize the potential of these applications, however, the data must be organized and processed such that these applications can make use of them. To this end, pay-as-you-go data cleaning and integration provide a foundation for *next-generation data integration platforms*: data integration systems that unify access, management, and understanding of these heterogeneous data sources (e.g., dataspace systems). The pay-as-you-go paradigm captures the nature of data cleaning and integration in dataspaces and provides a powerful philosophy for organizing specific techniques.

In this thesis, I have developed key building blocks designed to provide pay-as-you-go data cleaning: *Roomba*, a technique for effectively involving user feedback

to augment data cleaning mechanisms; *Metaphysical Data Independence*, a means of hiding all details of sensor data cleaning and integration under a single interface; *SMURF* an adaptive mechanism for cleaning RFID data; and *ESP*, a declarative-query based cleaning framework for sensor data streams. These techniques embody three important principles that should underlie any pay-as-you-go cleaning approach: ease of setup and deployment, efficient use of human input, and adaptability.

Additionally, I showed that a focus on the pay-as-you-go philosophy does not preclude effective data integration mechanisms. Indeed, in many cases the techniques developed in this thesis are capable of producing higher-quality data than current cleaning and integration techniques.

The work in this thesis lays a foundation for unifying disparate data sources in a dataspace from which there are many directions for future work.

One area of future work is investigating the application of human input to sensor data. Many sensor-based applications are real-time and thus preclude such approaches, but there are cases where it makes sense to have humans in the loop. For instance, humans could be leveraged to clean sensor data that is to be archived for later analysis. At an extreme, mass collaboration could potentially be used to move human-based cleaning into near-real time.

The work in this thesis addressed quality assessment of dataspace data by multiple means; however, one of the keys to supporting dataspace applications is the development of a unified framework for quality assessment and tracking, applicable to all data sources and processes within a dataspace. Such a framework and associated interface for exposing quality data is essential in enabling robust dataspace applications.

One of the most interesting areas of philosophical exploration is the expansion of the MDI approach to data sources beyond sensors. MDI advocated a complete

independence from the sources of the data, so a natural next step is to investigate how such a philosophy applies to Web data or to the unification of Web and sensor data, such as in TCA. Such an interface, utilizing a common abstraction for data as well as quality, could become the standard means by which applications interact with dataspace data.

Finally, there remains an open question on how to build *shared* dataspace systems that provide the same underlying data to multiple applications, where each application has a different notion of how much cleaning and integration is necessary. For instance, an application utilizing supply-chain data for inventory planning needs more accurate sensor data than a TCA application. As another example, different applications may have different notions of semantic equivalence; e.g., the colors "cardinal" and "red" are the same for most applications, but different to graphic design applications. A single dataspace system servicing these diverse applications would need to internally track multiple versions of the same data and expose them through different interfaces. Determining a flexible means of exposing dataspace data in different ways to different applications in an efficient manner is a challenging open research direction.

In summary, this thesis makes two broad contributions. First, it demonstrates that a pay-as-you-go approach to data cleaning and integration is both necessary and feasible to access large collections of heterogeneous data sources and make their data usable for an emerging class of applications. Second, it proposes a suite of pay-as-you-go based data cleaning and integration techniques that address different challenges associated with providing dataspace data to applications. Dataspace applications have the power to change the world by presenting disparate data in never-before-seen forms; the contributions of this thesis provide the necessary first steps in realizing the potential of these applications.

Bibliography

[ABW06] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.

[ACC+03] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, and S. Zdonik. Aurora: a data stream management system. In *ACM SIGMOD*, 2003.

[ALE05] Application Level Event (ALE) Specification Version 1.0, 2005. http://www.epcglobalinc.org/standards_technology/ EPCglobal_ApplicationALE_Specification_v112-2005.pdf.

[Ali05a] Alien RFID tags. Retrieved March, 2005. http://www.alientechnology.com/products/rfid-tags.

[Ali05b] Alien Technology, 2005. Nanoscanner Reader User Guide.

[Ali05c] Alien Technology, 2005. personal correspondence.

[ALR05] Alien ALR-9780 915 MHz RFID Reader. Retrieved March, 2005. http://www.alientechnology.com/products/rfid-readers.

[AM00] Gregory D. Abowd and Elizabeth D. Mynatt. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction*, 7(1):29–58, 2000.

[AS06] Eugene Agichtein and Sunita Sarawagi. Scalable information extraction and integration (tutorial). In *KDD '06*, 2006.

[AZN08] Amazon.com Home Page. Retrieved April, 2008. www.amazon.com.

[BBD+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. In *Principles of Database Systems*, 2002.

[BDG+07] Lukas Blunschi, Jens-Peter Dittrich, Olivier René Girard, Shant Kirakos Karakashian, and Marcos Antonio Vaz Salles. A dataspace odyssey:

The imemex personal dataspace management system (demo). In *CIDR*, pages 114–119, 2007.

[Ber01] Michael Bergman. The deep web: Surfacing hidden value. *JEP the Journal of Electronic Publishing*, 7(1), 8 2001.

[BGH+05] Phil Buonadonna, David Gay, Joseph M. Hellerstein, Wei Hong, and Samuel Madden. TASK: Sensor Network in a Box. In *EWSN*, 2005.

[BGS01] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Towards Sensor Database Systems. In *Proc. Mobile Data Management*, volume 1987 of *Lecture Notes in Computer Science*, Hong Kong, January 2001. Springer.

[BLHS04] Christof Bornhövd, Tao Lin, Stephan Haller, and Joachim Schaper. Integrating Automatic Data Acquisition with Business Processes - Experiences with SAP's Auto-ID Infrastructure. In *VLDB*, 2004.

[BPSM98] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0 - W3C recommendation 10-february-1998. Technical Report REC-xml-19980210, 1998.

[Car08] Cars.com. Retrieved April, 2008. http://www.cars.com/.

[CB02] George Casella and Roger Berger. *Statistical Inference*. Duxbury, 2002.

[CCD+03] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R. Madden, Vijayshankar Raman, Fred Reiss, and Mehul A. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *CIDR*, 2003.

[CD97] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1), March 1997.

[CDS04] Surajit Chaudhuri, Gautam Das, and Utkarsh Srivastava. Effective use of block-level sampling in statistics estimation. In *SIGMOD '04*, 2004.

[CDYR08] F. Chen, A. Doan, J. Yang, and R. Ramakrishnan. Efficient information extraction over evolving text data. In *ICDE '08*, 2008.

[CGGM03] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 313–324. ACM Press, 2003.

[CGMH+94] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer

Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, 1994.

[CHS+95] M.J. Carey, L.M. Haas, P.M. Schwarz, M. Arya, W.F. Cody, R. Fagin, M. Flickner, A.W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J.H. Williams, and E.L. Wimmers. Towards heterogeneous multimedia information systems: the garlic approach. *ride*, 00:124, 1995.

[CHS99] Francis Chu, Joseph Y. Halpern, and Praveen Seshadri. Least expected cost query optimization: an exercise in utility. In *PODS '99*, 1999.

[CJ89] D.-M. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Comput. Netw. ISDN Syst.*, 17(1), 1989.

[CKRS04] Sudarshan S. Chawathe, Venkat Krishnamurthy, Sridhar Ramachandran, and Sanjay E. Sarma. Managing RFID Data. In *VLDB*, 2004.

[CKZ+05] Jianfeng Chen, Alvin Harvey Kam, Jianmin Zhang, Ning Liu, and Louis Shue. Bathroom Activity Monitoring Based on Sound. In *Pervasive*, 2005.

[CL96] Jim Cowie and Wendy Lehnert. Information extraction. *Commun. ACM*, 39(1):80–91, 1996.

[CL08] Craigslist.com Home Page. Retrieved March, 2008. http://www.craigslist.com.

[CLWB01] Mark Claypool, Phong Le, Makoto Wased, and David Brown. Implicit interest indicators. In *Intelligent User Interfaces*, pages 33–40, 2001.

[CM03] William W. Cohen and Andrew McCallum. Information extraction from the web (tutorial). In *KDD '03*, 2003.

[Coc77] William G. Cochran. *"Sampling Techniques"*. John Wiley & Sons, 1977.

[CPS08] U.S. Consumer Product Safety Commision Home Page. Retrieved April, 2008. http://www.cpsc.gov/.

[CRF03] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string distance metrics for name-matching tasks, 2003.

[CS03] Gretchen B. (Editor) Chapman and Frank A. (Editor) Sonnenberg. *Decision Making in Health Care: Theory, Psychology, and Applications*. Cambridge University Press; New Ed edition (September 1, 2003), 2003.

[Dan05] Daniel Dobkin and Steven Weigand. Tags vs. the World: HF and UHF Tags in non-ideal environments. *WCA RFID SIG*, June 2005.

[DBL08] DBLP Computer Science Bibliography. Retrieved March, 2008. http://dblp.uni-trier.de/.

[DDH01] AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Reconciling schemas of disparate data sources: a machine-learning approach. In *SIGMOD '01*, 2001.

[Dea04] Daniel D. Deavours. Performance Analysis of Commercially Available UHF RFID Tags Based on EPCglobal's Class 0 and Class 1 Specifications. *RFID Alliance Lab*, 2004.

[Del07] del.icio.us home page. retrieved june, 2007. http://del.icio.us.

[DGM+04] Amol Deshpande, Carlos Guestrin, Sam Madden, Joseph M. Hellerstein, and Wei Hong. Model-Driven Data Acquisition in Sensor Networks. In *VLDB Conference*, 2004.

[DH05] AnHai Doan and Alon Y. Halevy. Semantic-integration research in the database community. *AI Mag.*, 26(1):83–94, 2005.

[DKB03] Christian Decker, Uwe Kubach, and Michael Beigl. Revealing the retail black box by interaction sensing. In *ICDCSW '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 328, Washington, DC, USA, 2003. IEEE Computer Society.

[DM06] Amol Deshpande and Samuel Madden. MauveDB: supporting model-based user views in database systems. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, 2006.

[DR05] Demand response enabling technology project. retrieved july, 2005. http://cbe.berkeley.edu/research/briefs-demand.htm.

[DRC+06] AnHai Doan, Raghu Ramakrishnan, Fei Chen, Pedro DeRose, Yoonkyong Lee, Robert McCann, Mayssam Sayyadian, and Warren Shen. Community information management. *IEEE Data Eng. Bull.*, 29(1):64–72, 2006.

[DRV06] A. Doan, R. Ramakrishnan, and Vaithyanathan. Managing information extraction: state of the art and research directions (tutorial). In *SIGMOD*, 2006.

[DS+07] Pedro DeRose, Warren Shen, Fei Chen 0002, Yoonkyong Lee, Douglas Burdick, AnHai Doan, and Raghu Ramakrishnan. Dblife: A community information management platform for the database research community (demo). In *CIDR*, pages 169–172, 2007.

[EN03] Eiman Elnahrawy and Badri Nath. Cleaning and querying noisy sensors. In *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, 2003.

[EPC05a] EPC Tag Data Specification Version 1.1, 2005. http://www.epcglobalinc.org/standards_technology/EPCTagDataSpecification11rev124.pdf.

[EPC05b] EPCGlobal, Inc. Home Page. Retrieved November, 2005. http://www.epcglobalinc.org/.

[FBA08] Facebook ads. retrieved march, 2008. http://www.facebook.com/ads.

[FDA08] U S Food and Drug Administration Home Page. Retrieved May, 2008. www.fda.gov.

[FGB02] Anton Faradjian, Johannes Gehrke, and Philippe Bonnet. GADT: A probability space ADT for representing and querying the physical world. In *ICDE*, 2002.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1, 1999.

[FHM05] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: A new abstraction for information management. *Sigmod Record*, 34(4):27–33, 2005.

[Fie00] Roy Thomas Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[FJK+05] Michael J. Franklin, Shawn R. Jeffery, Sailesh Krishnamurthy, Fredrick Reiss, Shariq Rizvi, Eugene Wu, Owen Cooper, Anil Edakkunni, and Wei Hong. Design Considerations for High Fan-In Systems: The HiFi Approach. In *CIDR*, 2005.

[FJPR04] Kenneth P. Fishkin, Bing Jiang, Matthai Philipose, and Sumit Roy. I Sense a Disturbance in the Force: Unobtrusive Detection of Interactions with RFID-tagged Objects. In *Ubicomp*, 2004.

[FL04] Christian Floerkemeier and Matthias Lampe. Issues with RFID usage in ubiquitous computing applications. In *Pervasive*, 2004.

[Fli07] Flickr home page. retrieved june, 2007. http://www.flickr.com.

[GB07] Google Base home page. retrieved july, 2007. http://base.google.com.

[GBT+04] Carlos Guestrin, Peter Bodik, Romain Thibaux, Mark Paskin, and Samuel Madden. Distributed regression: an efficient framework for modeling sensor network data. In *IPSN'04: Proceedings of the third international symposium on Information processing in sensor networks*, 2004.

[GC07] Google co-op home page. retrieved june, 2007. http://www.google.com/coop.

[GFSS00] Helena Galhardas, Daniela Florescu, Dennis Shasha, and Eric Simon. Ajax: an extensible data cleaning tool. *SIGMOD Rec.*, 29(2):590, 2000.

[GHM+06] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. Soap version 1.2 part 1: Messaging framework. Technical report, 2006.

[GLvB+03] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *SIGPLAN*, 2003.

[GM08] Gmail. retrieved may, 2008. http://mail.google.com.

[GMA08] Google Maps API. Retrieved April, 2008. http://code.google.com/apis/maps/.

[GS108] GS1 US Home Page. Retrieved March, 2008. http://www.uc-council.org/.

[GS04] Alka Gupta and Mayank Srivastava. Developing Auto-ID Solutions using Sun Java System RFID Software. Oct 2004.

[HBF+04] D. Hahnel, W. Burgard, D. Fox, K. Fishkin, and M. Philipose. Mapping and Localization with RFID Technology. In *ICRA*, 2004.

[HBZ+06] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen K. Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. CarTel: A Distributed Mobile Sensor Computing System. In *4th ACM SenSys*, Boulder, CO, November 2006.

[HFM06] Alon Y. Halevy, Michael J. Franklin, and David Maier. Dataspaces: A new abstraction for information management. In *DASFAA*, pages 1–2, 2006.

[HKL+06] Tian He, Sudha Krishnamurthy, Liqian Luo, Ting Yan, Lin Gu, Radu Stoleru, Gang Zhou, Qing Cao, Pascal Vicaire, John A. Stankovic, Tarek F. Abdelzaher, Jonathan Hui, and Bruce Krogh. Vigilnet: An integrated sensor network system for energy-efficient surveillance. *ACM Trans. Sen. Netw.*, 2(1):1–38, 2006.

[HKPH03] Eric Horvitz, Carl Kadie, Tim Paek, and David Hovel. Models of attention in computing and communication: from principles to applications. *Commun. ACM*, 46(3):52–59, 2003.

[HM08] Housingmaps.com. retrieved april, 2008. http://www.housingmaps.com.

[Hog06] Hogthrob Home Page. Retrieved June, 2006. http://www.hogthrob.dk/.

[HS98] Mauricio A. Hernandez and Salvatore J. Stolfo. Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[iAn06] Manage Data Successfully with RFID Anywhere Edge Processing. Retrieved March, 2006. http://www.ianywhere.com/developer/rfid_anywhere/rfidanywhere_edgeprocessing.pdf.

[II08] IBM Information Integrator. Retrieved March, 2008. http://www.ibm.com/software/data/integration/.

[Inf08] Informatica home page. retrieved march, 2008. http://www.informatica.com/.

[Ioa03] Yannis E. Ioannidis. The history of histograms (abridged). In *VLDB*, pages 19–30, 2003.

[iPh08] Apple iPhone Home Page. Retrieved March, 2008. www.apple.com/iphone.

[iRo07] iRobot Roomba Home Page. Retrieved November, 2007. http://www.irobot.com.

[JAF+06a] Shawn R. Jeffery, Gustavo Alonso, Michael J. Franklin, Wei Hong, and Jennifer Widom. A Pipelined Framework for Online Cleaning of Sensor Data Streams. In *ICDE*, 2006.

[JAF+06b] Shawn R. Jeffery, Gustavo Alonso, Michael J. Franklin, Wei Hong, and Jennifer Widom. Declarative Support for Sensor Data Cleaning. In *Pervasive*, 2006.

[Jen96] Finn V. Jensen. *Introduction to Bayesian Networks*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[JFG08] Shawn R. Jeffery, Michael J. Franklin, and Minos Garofalakis. An adaptive rfid middleware for supporting metaphysical data independence. *VLDB Journal*, 2008.

[JFH08] Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy. Pay-as-you-go user feedback for dataspace systems. *SIGMOD*, 2008.

[JGF06] Shawn R. Jeffery, Minos Garofalakis, and Michael J. Franklin. Adaptive Cleaning for RFID Data Streams. In *VLDB'2006: Proceedings of the 32nd international conference on Very large data bases*, pages 163–174, 2006.

[JGP+05] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR '05*, 2005.

[KBS06] Nodira Khoussainova, Magdalena Balazinska, and Dan Suciu. Towards correcting input data errors probabilistically using integrity constraints. In *MobiDE '06: Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access*, 2006.

[KNLZ07] Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao. Senseweb: An infrastructure for shared sensing. *IEEE MultiMedia*, 14(4):8–13, 2007.

[KOA+99] Cory D. Kidd, Robert Orr, Gregory D. Abowd, Christopher G. Atkeson, Irfan A. Essa, Blair MacIntyre, Elizabeth D. Mynatt, Thad Starner, and Wendy Newstetter. The Aware Home: A Living Laboratory for Ubiquitous Computing Research. In *Cooperative Buildings*, pages 191–198, 1999.

[Kri06] Sailesh Krishnamurthy. *Shared Query Processing in Data Streaming Systems*. PhD thesis, University of California, Berkeley, 2006.

[Lab05] Intel Lab Data. Retrieved July, 2005. http://berkeley.intel-research.net/labdata/.

[Lau05] Laurie Sullivan. RFID Implementation Challenges Persist, All This Time Later. *Information Week*, Oct 2005.

[LCC+05] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter, J. Tabert, P. Powledge, G. Borriello, and B. Schilit. Place lab: Device positioning using radio beacons in the wild. In *Proceedings of PERVASIVE 2005, Third International Conference on Pervasive Computing*, Munich, Germany, 2005.

[LHC08] The Large Hadron Collider Home Page. Retrieved February, 2008. http://lhc.web.cern.ch/lhc/.

[LHM+84] Bruce G. Lindsay, Laura M. Haas, C. Mohan, Paul F. Wilms, and Robert A. Yost. Computation and communication in r*: a distributed database manager. *ACM Trans. Comput. Syst.*, 2(1):24–38, 1984.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the Twenty-second International Conference on Very Large*

Databases, pages 251–262, Bombay, India, 1996. VLDB Endowment, Saratoga, Calif.

[MAG+97] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass, and Jennifer Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, 1997.

[Mar06] Margaret Martonosi. Embedded systems in the wild: Zebranet software, hardware, and deployment experiences. In *LCTES*, 2006.

[MCP+02] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, 2002.

[MCWG95] Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford, 1995.

[MDV+03] Robert McCann, AnHai Doan, Vanitha Varadaran, Alexander Kramnik, and ChengXiang Zhai. Building data integration systems: A mass collaboration approach. In *WebDB*, 2003.

[MFHH02] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: A Tiny AGgregation service for ad-hoc sensor networks. In *Proc. of OSDI 2002*, December 2002.

[MHC+06] Jayant Madhavan, Alon Y. Halevy, Shirley Cohen, Xin Luna Dong, Shawn R. Jeffery, David Ko, and Cong Yu. Structured data meets the web: A few observations. *IEEE Data Eng. Bull.*, 29(4):19–26, 2006.

[MHF06] David Maier, Alon Y. Halevy, and Michael J. Franklin. Dataspaces: Co-existence with heterogeneity. In *KR*, page 3, 2006.

[MIT06] MIT House_n Home Page. Retrieved May, 2006. http://architecture.mit.edu/house_n/.

[MJC+07] Jayant Madhavan, Shawn R. Jeffery, Shirley Cohen, Xin (Luna) Dong, David Ko, Cong Yu, and Alon Halevy. Web-scale data integration: You can only afford to pay as you go. In *CIDR*, 2007.

[MPD04] Shoubhik Mukhopadhyay, Debashis Panigrahi, and Sujit Dey. Data aware, Low cost Error correction for Wireless Sensor Networks. In *WCNC*, 2004.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *"Randomized Algorithms"*. Cambridge, 1995.

[MVN44] Oskar Morgenstern and John Von Neumann. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[NFC08] Nokia Near Field Communication. Retrieved March, 2008. http://europe.nokia.com/A4307097.

[ONS05] Object Naming Service (ONS) Standard, Version 1.0, 2005. http://www.epcglobalinc.org/standards/Object_ Naming_Service_ONS_Standard_Version_1.0.pdf.

[Ora08] Oracle Data Integration. Retrieved April, 2008. www.oracle.com/technology/products/oracle-data-integrator/index.html.

[Pax94] A Paxon. *The Food Miles Report: The Dangers of Long Distance Food Transport*. SAFE Alliance, 1994.

[PGM05] Mark A. Paskin, Carlos Guestrin, and Jim McFadden. A robust architecture for distributed inference in sensor networks. In *IPSN*, 2005.

[PHL+05] Joseph Polastre, Jonathan Hui, Philip Levis, Jerry Zhao, David Culler, Scott Shenker, and Ion Stoica. A unifying link abstraction for wireless sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 76–89, New York, NY, USA, 2005. ACM.

[PIR08] Georgetown Protein Information Resource. Retrieved March, 2008. http://pir.georgetown.edu/.

[Pla] Plato. *"Republic"*.

[Qin96] S. Qin. Neural networks for intelligent sensors and control — practical issues and some solutions. In Neural Networks for Control, D. Elliott, Ed. Academic Press, 1996. Chapter 8., 1996.

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[RD00] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[RDTC06] Jun Rao, Sangeeta Doraiswamy, Hetal Thakkar, and Latha S. Colby. A deferred cleansing method for rfid data analytics. In *VLDB'2006: Proceedings of the 32nd international conference on Very large data bases*, 2006.

[REA99] Juho Rousu, Tapio Elomaa, and Robert J. Aarts. Predicting the Speed of Beer Fermentation in Laboratory and Industrial Scale. In *IWANN (2)*, pages 893–901, 1999.

[Rec08] Recalls.gov Home Page. Retrieved March, 2008. www.recalls.gov.

[RJ05] F. Radlinski and T. Joachims. Query chains: Learning to rank from implicit feedback, 2005.

[RJK+05] Shariq Rizvi, Shawn R. Jeffery, Sailesh Krishnamurthy, Michael J. Franklin, Nathan Burkhart, Anil Edakkunni, and Linus Liang. Events on the edge. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 2005.

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[SAL+96] Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. Mariposa: A wide-area distributed database system. *VLDB Journal*, 5(1):48–63, 1996.

[SB02] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *KDD '02*, 2002.

[SDK+07] Marcos Antonio Vaz Salles, Jens-Peter Dittrich, Shant Kirakos Karakashian, Olivier René Girard, and Lukas Blunschi. itrails: pay-as-you-go information integration in dataspaces. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 663–674. VLDB Endowment, 2007.

[Sen04] SensorID Series 2 Agile Reader Query Protocol, Apr 2004.

[Sen06] Senosrmatic Agile 2 915Hz RFID Reader, 2006. http://www.sensormatic.com/RFID/stationary/.

[SF08] Salesforce.com home page. retrieved march, 2008. http://www.salesforce.com.

[SHMM98] Craig Silverstein, Monika Henzinger, Hannes Marais, and Michael Moricz. Analysis of a very large altavista query log. Technical Report 1998-014, Digital SRC, 1998. http://gatekeeper.dec.com/pub/DEC/SRC/technical-notes/abstracts/src-tn-1998-014.html.

[Sho08] Shopping.com Home Page. Retrieved April, 2008. www.shopping.com.

[SJFW06] Anish Das Sarma, Shawn R. Jeffery, Michael J. Franklin, and Jennifer Widom. Estimating Data Stream Quality for Object-Detection Applications. In *IQIS*, 2006.

[Son06] Sonoma Redwood Sensor Network Deployment. Retrieved January, 2006. http://www.cs.berkeley.edu/get/sonoma/.

[SOS05] Luca Schenato, Songhwai Oh, and Shankar Sastry. Swarm Coordination for Pursuit Evasion Games using Sensor Networks. In *Proc. of ICRA*, 2005.

[SP08] SwissProt Home Page. Retrieved March, 2008. http://expasy.org/sprot/.

[SS07] SecondString Home Page. Retrieved November, 2007. http://secondstring.sourceforge.net/.

[SSP+06] Joshua R. Smith, Alanson P. Sample, Pauline S. Powledge, Sumit Roy, and Alexander Mamishev. A Wirelessly-Powered Platform for Sensing and Computation. In *Ubicomp*, pages 495–506, 2006.

[SSW92] Carl-Erik Särndal, Bengt Swensson, and Jan Wretman. *"Model Assisted Survey Sampling"*. Springer-Verlag New York, Inc. (Springer Series in Statistics), 1992.

[SSW+05] C. Sharp, S. Schaffert, A. Woo, N. Sastry, C. Karlof, S. Sastry, and D. Culler. Design and Implementation of a Sensor Network System for Vehicle Tracking and Autonomous Interception. 2005.

[Sti61] George J. Stigler. The economics of information. *Journal of Political Economy*, 69:213, 1961.

[Sti02] Joseph E. Stiglitz. Information and the change in the paradigm in economics. *American Economic Review*, 92(3):460–501, June 2002.

[Sto86] Michael Stonebraker. The design and implementation of distributed ingres. pages 187–196, 1986.

[Sur04] J. Surowiecki. *The wisdom of crowds*. Doubleday, 2004.

[TF08] TheFind.com Home Page. Retrieved February, 2008. http://www.thefind.com.

[TPS+05] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David E. Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong. A Macroscope in the Redwoods. In *SenSys*, pages 51–63, 2005.

[UWR06] UW RFID Lab. Retrieved March. 2006. http://www.uwrfidlab.org/.

[vAD04] Luis von Ahn and Laura Dabbish. Labeling Images with a Computer Game. In *ACM CHI*, 2004.

[Vij01] Vijayshankar Raman and Joseph M. Hellerstein. Potter's Wheel: An Interactive Data Cleaning System. In *The VLDB Journal*, pages 381–390, 2001.

[VR79] C. J. Van Rijsbergen. *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow, 1979.

[VVSK00] Panos Vassiliadis, Zografoula Vagena, Spiros Skiadopoulos, and Nikos Karayannidis. Arktos: A tool for data cleaning and transformation in data warehouse environments. *IEEE Data Eng. Bull.*, 23(4):42–47, 2000.

[Wan04] Roy Want. The Magic of RFID. *ACM Queue*, 2(7):40–48, 2004.

[WDR06] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance Complex Event Processing Over Streams. In *SIGMOD Conference*, 2006.

[Wid05] Jennifer Widom. Trio: A System for Integrated Management of Data, Accuracy, and Lineage. In *CIDR*, 2005.

[Wie92] Gio Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, 1992.

[Win99] W. Winkler. The state of record linkage and current research problems, 1999.

[WL05] Fusheng Wang and Peiya Liu. Temporal Management of RFID Data. In *VLDB*, pages 1128–1139, 2005.

[WMD08] WebMd Home Page. Retrieved March, 2008. www.webmd.com.

[WSF95] Richard Y. Wang, Veda C. Storey, and Christopher P. Firth. A framework for analysis of data quality research. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):623–640, 1995.

[WYDM04] Wensheng Wu, Clement Yu, AnHai Doan, and Weiyi Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *SIGMOD '04*, 2004.

[X1005] X10 Home Page. Retrieved March, 2005. http://www.x10.com.