WiSDM: A Platform for Crowd-Sourced Data Acquisition, Analytics, and Synthetic Data Generation

Ananya Choudhury

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

April 29, 2016
Blacksburg, Virginia

Keywords: Human Computation, Crowd Sourcing, Synthetic Dataset, Data Analytics, Data Visualization, Epidemiology, WiSDM

WiSDM: A Platform for Crowd-Sourced Data Acquisition, Analytics, and Synthetic Data Generation

Ananya Choudhury

(ABSTRACT)

Human behavior is a key factor influencing the spread of infectious diseases. Individuals adapt their daily routine and typical behavior during the course of an epidemic – the adaptation is based on their perception of risk of contracting the disease and its impact. As a result, it is desirable to collect behavioral data before and during a disease outbreak. Such data can help in creating better computer models that can, in turn, be used by epidemiologists and policy makers to better plan and respond to infectious disease outbreaks. However, traditional data collection methods are not well suited to support the task of acquiring human behavior related information; especially as it pertains to epidemic planning and response[37].

Internet-based methods are an attractive complementary mechanism for collecting behavioral information. Systems such as Amazon Mechanical Turk (MTurk) and online survey tools provide simple ways to collect such information[68]. This thesis explores new methods for information acquisition, especially behavioral information that leverage this recent technology.

Here, we present the design and implementation of a crowd-sourced surveillance data acquisition system – WiSDM. WiSDM is a web-based application and can be used by anyone with access to the Internet and a browser. Furthermore, it is designed to leverage online survey tools and MTurk; WiSDM can be embedded within MTurk in an iFrame. WiSDM has a number of novel features, including, (i) ability to support a model-based abductive reasoning loop: a flexible and adaptive information acquisition scheme driven by causal models of epidemic processes, (ii) question routing: an important feature to increase data acquisition efficacy and reduce survey fatigue and (iii) integrated surveys: interactive surveys to provide additional information on survey topic and improve user motivation.

We evaluate the framework's performance using Apache JMeter and present our results. We also discuss three other extensions of WiSDM: Adapter, Synthetic Data Generator, and WiSDM Analytics. The API Adapter is an ETL extension of WiSDM which enables extracting data from disparate data sources and loading to WiSDM database. The Synthetic Data Generator allows epidemiologists to build synthetic survey data using NDSSL's Synthetic Population as agents. WiSDM Analytics empowers users to perform analysis on the data by writing simple python code using Versa APIs. We also propose a data model that is conducive to survey data analysis.

WiSDM: A Platform for Crowd-Sourced Data Acquisition, Analytics, and Synthetic Data Generation

Ananya Choudhury

(GENERAL AUDIENCE ABSTRACT)

In this work, we present WiSDM (Widely Internet Sourced Distributed Monitoring), a web-based framework that facilitates acquisition and analysis of human behavioral information. The information is acquired through online surveys, designed and hosted in WiSDM's data collection framework and is utilized by researchers to predict how disease spreads during an outbreak. Researchers collect as much relevant information as possible and build computer models that can predict future epidemics well in advance. WiSDM provides several features that enable researchers to design intelligent surveys adaptible to users' behavior and which only pose questions most relevant to responders. The computer models used for disease prediction need to be trained before they can accurately predict disease outbreaks. But training these models require a large set of survey responses and it is not always feasible to collect responses from such a large number of people. That is why WiSDM provides a module called Synthetic Data Generator that synthetically generates near-real survey responses which can be used to train and test the computer models. WiSDM also has an analytics module – WiSDM Analytics – through which the generated or collected survey responses can be analyzed and can also be visualized as graphs and charts. Information from regions beyond WiSDM's reach can be collected using external survey tools which are locally popular and then using WiSDM's Adapter module, the information can be pulled into the WiSDM system for further analysis and processing.

Acknowledgements

I would like to express my appreciation to my advisory committee: Dr. Madhav Marathe, Dr. Sandeep Gupta, and Dr. Keith Bisset. Thanks for giving me the opportunity to be a part of the NDSSL research group.

I offer my sincerest gratitude to my advisor Dr. Madhav Marathe, who has supported me thoughout my thesis with his knowledge and patience while giving me the freedom to explore on my own. He has supported me not only by providing research assistantship for over a year but also by giving research ideas during various stages of this thesis.

Dr. Sandeep Gupta's insightful comments and constructive criticisms at different stages of my research were thought-provoking and they helped me focus my ideas. I am grateful to him for his prompt support whenever I needed his help.

I acknowledge my gratitude to Dr. Keith Bisset who has always been there to listen and give advice.

A very special thanks to my mentor Mandy Wilson, for the immense help and encouragement she provided at all levels of this research project. Without your guidance and assistance, I would not have finished this thesis.

In conclusion, I owe my gratitude to all those people who have helped me and supported me and because of whom my graduate experience has been one that I will cherish forever.

Contents

List of Figures

Chapter 1

Introduction

Scenario

Consider two theories.

- Theory 1: The CDC has predicted mild flu activity for this season, and
- Theory 2: Flu trends in Blacksburg match CDC flu predictions.

Now given these two theories, researchers in Blacksburg conduct an online study to acquire information about what strategies people are taking to avoid getting the flu. They analyze the collected data and find that people incorporated many strategies necessary to avoid the flu, such as getting vaccinated, washing hands regularly, etc. However, they also notice a below average trend in the vaccination rates among school-aged children. They hypothesize that vaccinations for young children are not widely available in Blacksburg or parents of school-aged children are reluctant to have their children vaccinated. The first hypothesis is rejected because traditional flu vaccines, approved for most people above 6 months of age, are widely available throughout Blacksburg and neighboring areas. To evaluate the second hypothesis, they launch a new study targeting parents of children in that age range, collect behavioral information, and analyze the data. Based on the results of their new study, they accept their new hypothesis as a fact since there is no other plausible reason to explain the event. But if required, the researchers could revise their hypothesis and iterate through the process again.

This is a realistic portrayal of response during a disease outbreak. Response to such an emergency is carried out in multiple stages and involves multiple organizations like health department services, healthcare professionals, etc. It is recognized to be an important process but never been structured in a circular flow model. We call this the abductive loop. The term "abductive loop" is derived from abductive reasoning, where one infers a fact as a plausible explanation for observed behavior. Like abductive reasoning wherein one makes a hypothesis to find the most likely explanation for a cause, researchers usually hypothesize to find the most plausible reason for an event. But they perform this reasoning in a loop by following a series of steps shown in Figure 1.1. The first stage usually involves collecting ground data which is then followed by preparing the data for modeling and simulation. Data preparation, in essence, assists in hypotheses generation. The next stage, modeling and simulation, is a deductive process which effectively leads to accepting or refuting those hypotheses. The stages in the figure are discussed further in the next section. Although we explain abductive loop and WiSDM in this thesis with respect to epidemiology, the concept, and the framework are generic enough to be extended to most other domains.

Abductive Loop

Abductive reasoning is a form of logical inference that goes from data describing an event to a hypothesis that best explains or accounts for that data. Given a background theory and an observation, abductive reasoning conjectures the best possible explanation for the observation from the theory provided the explanation is consistent with further observation[77]. It follows below syllogistic form[38].

- Theory: All the beans from this bag are white.
- Observation: These beans are white.
- Explanation: These beans are from this bag.

According to [34], logic-based abduction can be formally represented as follows:

Let

T *= a theory that represents all relevant domain knowledge*

M *= set of manifestations or observations*

H *= set of plausible hypotheses*

Then,

a suitable set $S \subseteq H$ *is an explanation of* M *such that*

$\forall S$, $T \cup S$ *is consistent and logically entails* M.

But sometimes no subset of available hypotheses within the current conceptual frame is consistent with all the observations. In such cases, the explanation involves the creation of a new concept that provides a new description of the relevant phenomena[3] .

Therefore,

if $\exists S' \in S$ *such that* $T \cup S$ *is not consistent for manifestation* $M' \in M$,

then

$\forall H' \subseteq H$, H' *is defeated and a new set* $S^* \nsubseteq H$ *is constructed as an explanation of* M *such that* $\forall S^*$, $T \cup S^*$ *is consistent and logically entails* M.

Abduction logic is common in scientific reasoning on large and small scales. In Artificial Intelligence, identification of a cause or diagnosis is often described as an abduction problem. Diagnosis can be viewed as producing an explanation that best accounts for the patient's (or device's) symptoms[86]. The task of a diagnostic reasoner is to come up with the best explanation for the symptoms, which are typically those findings for the case that show abnormal values. Taking diagnosis as abduction determines the classes of questions that are fair to ask to find the best plausible cause. It also suggests that computer-based diagnostic systems should be designed to make answering such questions straightforward and precise.

Let us elaborate on the flu epidemic example again in terms of the stages in the abductive loop as represented in Figure 1.1. Given a relevant set of theories that CDC has predicted a mild flu activity this season and that flu trends in Blacksburg match CDC flu predictions, researchers want to acquire data about what actions people are taking to prevent falling sick. There are active or passive data collection methods. Passive data collection involves gathering user data without notifying the user, such as collecting a user's geolocation information. Conversely, active data collection requires a user to spend some time entering information and requires that a user feels comfortable with providing information. This process is active learning where the system only asks the most maximally informative questions[70]. In this thesis, we concentrate on active learning. So our researchers build an intelligent survey that poses most relevant questions to the people living in Blacksburg. The collected data is then prepared for simulation and modeling. In this stage, the data is audited for errors and anomalies, consolidated for analysis, and outputted into a format so that it can be used in simulation and modeling stage. During data preparation, the researchers observe an abnormal pattern in the vaccination rates among school children. They generate a set of hypotheses that are plausible causes for below average vaccination rates among school going children.

1. Flu vaccines that are approved for children are not widely available in Blacksburg.
2. Parents of school children are reluctant to get their children vaccinated.

In order to find the most plausible explanation, researchers run simulations on their stochastic models built to establish causes for an effect. The simulation and modeling stage evaluates the hypotheses. The first hypothesis is refuted because several reports suggest that seasonal flu vaccines for both children and adults are widely available in various locations across Blacksburg. The other hypothesis which suggests that a certain section of the population did not get vaccinated is accepted because there is evidence that supports this explanation.

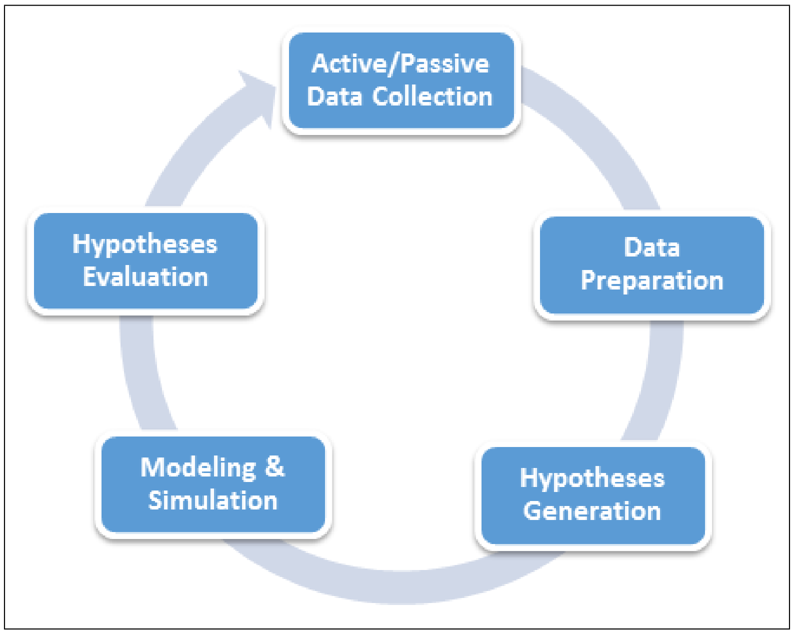


Figure 1.1: Abductive Loop in WiSDM

Now to confirm this new hypothesis, the researchers iterate through the abductive loop again. They launch another study. But now they change their questionnaire to seek information only about the transmitter population. They try to find why so many parents did not get their children vaccinated. After following the stages of data preparation, modeling and simulation, they analyze the results. A significant number of school children indeed did not get vaccinated this flu season. This was because many parents who responded to the survey cited a local news report published recently which said that getting so many vaccinations will overwhelm a child's immature immune system.

So to summarize, researchers perform abductive reasoning iteratively to find the best research conclusion. They can start this cycle without any hypothesis; for example, from active data collection stage and then eventually building and testing hypotheses as plausible causes of an observation (as indicated in our flu example above). They can also start the reasoning loop by first constructing hypotheses from belief or they can start from the modeling and simulation phase where they run simulation models on exogenous data sets to build hypotheses. Basically, by following the abductive loop, researchers can infer the best explanation for a phenomenon or reject all previous inferences and create a new concept that is the cause of the event. The inferred explanations must be consistent with the phenomenon and hold true even when additional observations are added.

Hypotheses generation is a crucial step but this may be skipped when an abductive loop is essentially followed to accept or refute a pre-existing hypothesis, as seen in the second iteration of the above flu example. In the second iteration, the hypotheses generation stage was skipped because that iteration was conducted primarily to assess a hypothesis approved in the first iteration. However if any new pattern were observed in the collected data during the second iteration, then the hypotheses generation stage would have been included.

The Modeling sub-stage in the abductive loop requires manual intervention and has not been implemented as a part of this work.

1.1 Background

Longitudinal Studies

A longitudinal study provides data about the same individual or a cohort of individuals at different points in time, allowing the researcher to track changes at the individual level. In this kind of study, subjects are followed over time with continuous or repeated monitoring of risk factors or health outcomes, or both. Such investigations vary enormously in their size and complexity. At one extreme, a big population may be studied over decades. For example, researchers at Harvard followed 268 men who entered college in the late 1930s through war, career, marriage, divorce, and old age, examining the question "What makes us happy?"[97]. This fascinating study set out to find if there is any formula for a good life. These subjects' mental states were monitored for 72 years; one of the participants was John F. Kennedy, who became the president of the United States. These 268 men underwent regular medical exams and psychological tests, returned questionnaires, and sat for interviews over all these years.

Longitudinal studies play a key role in epidemiology and clinical research. They examine associations between knowledge of or exposure to known or suspected causes of disease, along with subsequent morbidity or mortality rates. In the simplest design, a baseline health status measurement is obtained, an intervention is administered, and a follow-up measurement is collected. [109] states that in a longitudinal study, the tests determine whether a change in outcome variable Y(effect) is associated with a change in predictor variable X(cause), or whether there is an effect of one or more interventions on the outcome variable Y. The ultimate research design is to evaluate long-term and short-term effects on certain outcome variable Y. By comparing the change in measurements, a relative risk can be estimated. [57] states that the real strength of longitudinal studies is that they give a true description of the life course of real populations. They permit quantitative analysis of social, economic, psychological, and biological processes as they occur.

Internet-Based Surveys

The Internet is a popular medium for conducting behavioral research. It allows researchers to reach to thousands of people in a short time while they work on other tasks[61]. Once an invitation to respond to a survey is posted online, researchers can collect data while working on other projects[4]. First generation online survey researchers used email surveys where they had to cut-and-paste responses from emails into databases and data analysis programs. Now with the advent of online survey creation software packages, data is saved directly to the backend database and can also be exported to analytics software. Online surveys are also cost efficient. It eliminates the need for paper, postage, printing and data entry[61]. Moreover paid crowdsourcing services offer a relatively inexpensive pool of responders. In addition, citizen science projects such as Zooniverse[100] offers a similar pool of responders who are willing to provide services at no cost.

While Internet-based surveys are cost and time efficient, they still have some limitations. Certain sections of the population are underrepresented in Internet-based surveys[23]. For example, people with lower incomes, less education, living in rural areas or aged 65 and over are underrepresented among Internet users[114]. Among those who have access to the Internet and respond to the surveys, relatively little may be known about them, aside from basic demographic information and even that information may be questionable[27]. It is also observed that responders of such self-administered surveys have a tendency to choose status quo options[73] which may affect data quality. Traditional data collection methods such as face-to-face and telephonic surveys provide more coverage and respondents of such surveys have higher attention level which is a concern in self-administered surveys[19]. However, social desirability bias may be introduced[33] because of the presence of the interviewer. A relatively new concept in data collection has emerged lately, mixed mode survey involving the web, which addresses these problems[20]. This survey methodology offers to the responders multiple modes of taking a survey. By doing so, researchers attempt to raise participation appeal and increase population coverage. But 'mode effect' which refers to the difference in the way respondents may answer question depending upon the chosen mode can be a crucial downside[23]. However empirical studies[24, 49] show relatively small differences between different modes of surveying with the exception of surveys on sensitive topics.

[113] conducted a series of tests to evaluate how Internet surveys compare to traditional methodologies. [19, 117] provide an exhaustive review of the advantages and disadvantages of conducting online surveys. Both suggest that several factors are to be taken into consideration before selecting a data acquisition methodology.

CrowdSourcing In Epidemiology

Traditional disease monitoring methods have relied on public health data, such as hospital or census data. However, such methods have many drawbacks; particularly, they require individuals to consult a physician when they are ill, and for physicians to diagnose and report the disease. But an unknown proportion of the population, irrespective of age, sex or social status, do not seek healthcare when they are ill. This proportion may alter an infection model, especially if there is a change in the levels of public concern, as was observed in the United Kingdom (UK) during the 2009 H1N1v influenza pandemic[106]. On the other hand, participatory surveillance via the Internet rapidly assesses the level of infection within a community. [106] observed that Internet based surveillance captured a wider range of cases than traditional surveillance systems. While traditional surveillance methods required individuals to attend physicians when they were ill, Internet-based surveillance was quick and relatively inexpensive, hence a wider range of people could be polled, even those who did not seek health care.

Crowdsourcing platforms leverage this concept of participatory knowledge-sharing, paid or voluntary, which empowers the general public to report incidents. In epidemiological studies, crowdsourcing is used to collect rapid real-time information as individuals are diagnosed or even before. It fosters early detection of an emergent outbreak, which is vital in building infection models and predictions[17]. Amazon Mechanical Turk (also known as MTurk) is one of the dominant worker-on-demand platforms. Workers (also known as MTurkers or just Turkers) browse batches of human intelligent tasks (HITs) and complete HITs of interest[76]. They are paid by "requesters" upon successful completion of the accepted tasks. MTurk's large user base and simple payment system make it a convenient means of data collection. Consequently, MTurk has increasingly become a popular way for researchers to conduct behavioral experiments that involve data collection from the public.

We discuss more work on crowdsourcing in the next chapter.

Branched Surveys

These surveys can be designed to adapt depending upon user's response, thus avoiding redundant and irrelevant questions. This feature provides the ability to build "intelligent" surveys, meaning respondents will be asked only those questions which are most relevant to them. For example, consider a three question survey which does not have branched logic, as shown in Figure 1.2A:

Question 1: Did you get vaccinated? Yes or No

Question 2: If yes, when did you get vaccinated?

Question 3: If no, why didn't you get vaccinated?

Here responders will be asked all three questions irrespective of their response to the first question. But if one answered 'Yes' to Question 1, then Question 3 is irrelevant. Similarly, if one answered 'No' to Question 1, then Question 2 is not applicable. Consider a longer survey as shown in Figure 1.3. There is a total of 18 questions to be answered. In a fixed length survey, responders would have to answer all 18 questions irrespective of their responses to specific questions. However with branched logic, responders will answer questions only pertaining to them, which in this survey is at most 7. Having redundant questions in a survey

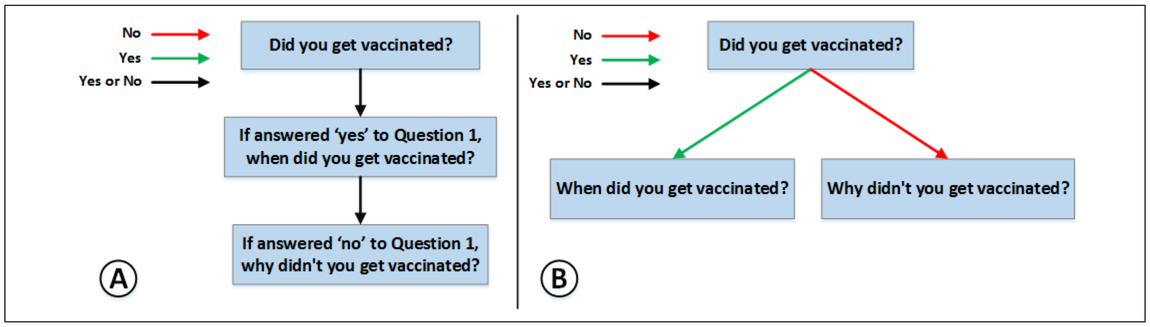

No
Yes
Yes or No
Did you get vaccinated?
If answered 'yes' to Question 1, when did you get vaccinated?
If answered 'no' to Question 1, why didn't you get vaccinated?
A
No
Yes
Yes or No
Did you get vaccinated?
When did you get vaccinated?
Why didn't you get vaccinated?
B

Figure 1.2: Difference between A. an ordinary survey and B. a branched survey

tends to increase survey fatigue. By avoiding such redundancies, the number of questions posed to a responder can be reduced to $O(\log n)$ from n. This significantly enhances survey efficacy and reduces survey cost.

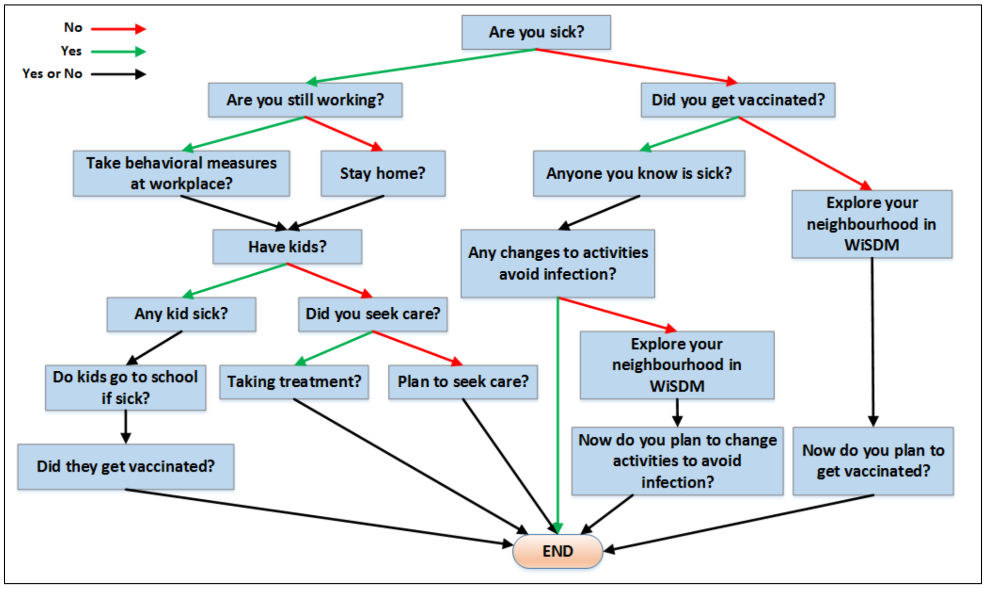

No
Yes
Yes or No
Are you sick?
Are you still working?
Did you get vaccinated?
Take behavioral measures at workplace?
Stay home?
Anyone you know is sick?
Explore your neighbourhood in WiSDM
Have kids?
Any changes to activities avoid infection?
Any kid sick?
Did you seek care?
Explore your neighbourhood in WiSDM
Do kids go to school if sick?
Taking treatment?
Plan to seek care?
Now do you plan to change activities to avoid infection?
Now do you plan to get vaccinated?
Did they get vaccinated?
END

Figure 1.3: A Branched Survey

Survey branching injects conditional logic in the survey flow which enables responders to skip questions that are not relevant to them. So with conditional logic, the previous three question survey can be converted to

Question 1: Did you get vaccinated? Yes or No

Branch logic: If answered 'Yes' to Question 1, go to Question 2

Question 2: When did you get vaccinated?

Branch logic: If answered 'No' to Question 1, jump to Question 3

Question 3: Why didn't you get vaccinated?

The survey takes the form of a tree as shown in Figure 1.2B. So now responders will answer only 2 questions. They will not see the underlying branched logic or the questions that are not in their branch path.

1.2 Motivation

In recent years, conducting longitudinal studies has become easier with the growth of Internet and crowdsourcing platforms. Researchers can easily draw respondents for any type of web-based surveys, and collect data almost instantaneously. There are many tools available that help facilitate building surveys online, but these survey tools do not support dynamic surveys because they do not maintain any change history. Making any alterations to existing surveys may lead to data loss, dangling questions, or responses not associated with an existing survey.

Survey design is a crucial factor in making surveys a more pleasant experience, an important factor for improving data quality. Survey branching can help improve user experiences by curtailing dull and lengthy questionnaires, but it is not a straightforward feature in most tools, and, for some, this feature is available only at an additional cost. Studies[13, 31] also suggest that gamification of surveys and increased interactivity lead to a better user experience if not always, better data quality. Not all surveys can be gamified, but they can easily be made more interactive by providing additional information, like CDC Flu Map, leading to increased user engagement. But few tools to date support surveys that can be integrated with external APIs. These survey tools are expensive, and they do not integrate easily with Amazon Mechanical Turk; they do offer services to recruit responders, but these services tend to be highly-priced when compared to recruiting workers from MTurk[103, 89]. A price comparison is shown in Table 1.1. MTurk also provides a requester UI to design and publish HITs, but setting up the UI may be a complicated process.

Table 1.1: Price comparison of various commercial data acquisition frameworks

Frameworks	Responders	Price
SurveyGizmo	1100 (Brazilians)	$5500 - $8800
Qualtrics	1100	$5500
QuestionPro	1100	$2300
Amazon Mechanical Turk	1100	$770

The information that is collected helps researchers predict an epidemic, or future trends for a prevalent outbreak, using inference algorithms. Before these inference models can accurately predict a situation, they need to be tested extensively, which often requires a large, non-biased data set. [105] states that much of the data that is collected from crowd workers are self-reported and that it cannot be verified whether the participant accurately reported the condition, which may introduce bias to the inference model. An artificially generated data set can overcome both issues of bias and data-size. Because in artificially generated data sets researchers have control over what data is being generated. They can define how virtual responders will behave under certain circumstances, and can also dictate how much data needs to be generated.

Analysis of collected data presents researchers with a big picture for situation assessment. But, there is little research on how survey data can be modeled for simpler analytics, or how existing data structures or analysis frameworks will scale when faced with a much larger data set. An appropriate data structure is crucial for simpler data analytics and a framework that can perform analytics across disparate data sources is important.

1.3 Contribution

This thesis describes a web-based system called WiSDM (Widely Internet Sourced Distributed Monitoring). WiSDM facilitates interactive data acquisition, inspection and forecasting of social dynamics in a population, and allows for a crowd-sourced collection of situational and behavioral data. It is based on an information economy paradigm – wherein users can provide individualized but anonymous information in return for novel insights produced by computational models. WiSDM presents users with data customized for them and gathers individual or collective data of and around the users in return. It can be integrated with online survey services and also with online labor markets such as Amazon Mechanical Turk; this provides a cost-effective approach to real-time surveillance of a potentially evolving disease outbreak. WiSDM is designed to support logic-based abduction combined with non-Bayesian inferences. It can be used by anyone with access to the Internet and a browser.

A basic version of WiSDM as a data acquisition system was already present from previous work[2]. Our contributions in WiSDM, which goes beyond implementing above mentioned attributes, has been built on the earlier version and can be summarized as below.

- Our primary contribution was to redesign and build WiSDM as a data collection system that adapts according to user's background information and behavior. It enables researchers to conduct surveys personalized for every responder. Surveys can be built with branched logic so that only relevant questions are posed to responders. We also redesigned the survey interface to make it more interactive (the responder needs to click 'Submit' to navigate to a new page to view next question) to improve responder's attention level[87, 107]. Moreover, a survey preamble guiding responders on how to

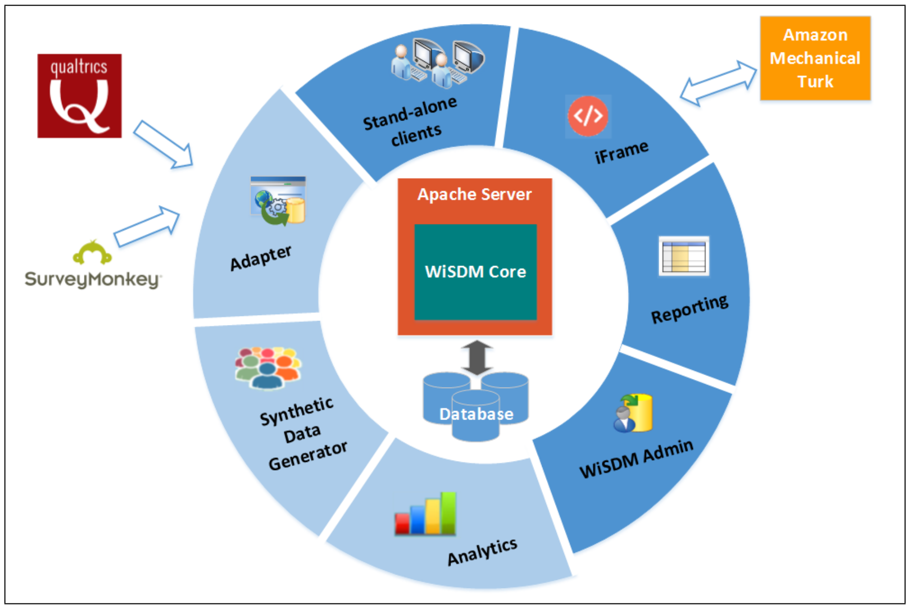


Figure 1.4: WiSDM Integrated Framework. The inner semi-circle (in a darker shade of blue) represents frameworks currently integrated with WiSDM and the outer semi-circle (in light blue) represents extensions of WiSDM.

perform a task is displayed at the beginning of the survey. The new survey interface is also designed to save responses from incomplete surveys. Researchers now have the ability to download survey data in multiple formats. The backend data model has also been changed to support more normalized data structure which enabled running simpler queries on survey questions and corresponding user responses.

- Brazil is a Portuguese-speaking country where crowdsourcing platforms like MTurk are not popular. Platforms that have reach in Brazil provide their own interface and do not support third party integration. For maximum outreach, such situations may not be ignored. Having an ETL extension to a data collection system will equip the tool to carry out surveys using other applications and then extract data from that external source and load to its local databases. We introduce an external module of WiSDM, the Adapter, which can perform Extract-Transform-Load (ETL)[16] operations.

- Building a synthetic data set is a controlled process that enables researchers to build a data set customized to their requirements by modulating parameters and features as necessary. The researchers can also control the size of the generated data set. We provide a parameterized framework, Synthetic Data Generator, that uses the Network Dynamics and Simulation Science Laboratory's (NDSSL) Synthetic Population as agents who can be used to generate synthetic survey results.

- We propose a data model that can be used as a backend data structure for analytics on survey data. We performed analytics on this data model using Versa, a python framework for analytics and visualization. In this work, we show how WiSDM can be used in conjunction with Versa to build advanced visualizations on the data.

Figure 1.4 is a pictorial representation of WiSDM integrated framework. In the subsequent chapters, we elaborate on its above-mentioned capabilities.

Chapter 2

WiSDM: A Crowdsourced Data Acquisition System

Epidemic spread depends on a number of network and behavioral parameters. Understanding the interaction between these various parameters is crucial to perceiving susceptibility and severity of the situation. For that, we need an adaptive approach to situation assessment and information acquisition. Internet-based data acquisition is one such approach that is flexible yet bears a low operational cost. There are several other advantages to Internet-based data acquisition [91, 92]. Not only do online data acquisition techniques reduce social desirability biases which are prevalent in traditional survey methods such as in-person surveys or voice-over surveys[37], but it also allows automation and experimental control over the survey. This means researchers can customize question sets based on subject demographics or responses to previous questions; a kind of flexibility that would be impossible without the Internet[68].

WiSDM, an online data acquisition and data analysis system, leverages this idea to create a framework that allows for abductive studies, where a researcher runs multiple experiments to infer the most plausible causes of an event based on facts and observations. Abductive studies are logical studies conducted in loops over longer durations of time. WiSDM integrates traditional data acquisition with abductive reasoning and captures the abductive reasoning loop inherent in observational and analytical studies. It facilitates the entire process, from establishing a hypothesis to behavior sensing to concluding if an inference can be taken as a fact.

In rest of the chapter, we discuss features of WiSDM in detail.

2.1 Hypotheses

We hypothesize that:

- WiSDM can facilitate the ability to pose the most maximally informative questions to gather most relevant information
- WiSDM can provide seamless plug and play integration with crowdsourcing platforms to access diverse groups of people.
- WiSDM can be extended to import data collected through external sources such as commercial survey tools so that analytics can be performed locally.

2.2 Related Work

Crowdsourcing

Many studies have been done on conducting social research and collecting behavioral data on crowdsourcing platforms like Amazon Mechanical Turk[76]. MTurk provides an existing and diverse pool of potential subjects that remains relatively stable over time. Such platforms also maximize the efficiency of abductive loop. For example, one can rapidly go from generating hypotheses to testing them, analyzing the results, and updating the theory. There are numerous studies that show correspondence between the behavior of workers on crowdsourcing platforms and behavior offline or in other online contexts. [104] replicated a public goods experiment that was conducted in the classroom[36], and, despite the difference in context and the relatively lower pay on Mechanical Turk, there were no significant differences between the results of the MTurk study and those from a prior study conducted in the classroom.

It is also important to create simple and robust interfaces so that certain HITs can be posted easily on MTurk. [83] built a framework TurkServer that provides an infrastructure to perform longitudinal and synchronous experiments that require bidirectional and real-time interaction among workers. However, the framework does not provide any user interface, which means that the infrastructure only provides skeleton classes which the researchers must extend to build user interfaces. This is a powerful medium, but, for researchers without a programming background, this may not be the best option. Moreover, Turkserver is targeted only towards Amazon Mechanical Turk. It can be a major limitation when we want to maximize outreach. In WiSDM, we provide a simple framework and user interface to post HITs on crowdsourcing platforms with iFrame support. [75] is another paper aimed towards running experiments on MTurk. This framework, Turkit, employs an iterative approach towards task completion, which, at the onset, looks similar to longitudinal studies; however, it works best for tasks that improve with repetition. An example given in the paper is of handwriting recognition. MTurkers were given the task of deciphering a bad handwriting note while skipping words they were not sure about. The results of the task were then posted again to another set of workers who tried to fill the remaining gaps and so on. After six iterations, the complete set was successfully deciphered by the crowd workers. The user responses after the first iteration in the study were built on the outcomes of all the preceding iterations, unlike longitudinal studies where only the questions conducted in subsequent tests are built on the outcomes of previous tests.

Ushahidi[46] and GeoChat[1] are two well-known crowdsourcing platforms to map crisis information from multiple data streams. Both require that participants use their interactive platforms to input ground information for better situation awareness. GeoChat has been used for disease reporting and monitoring in countries like Cambodia and Thailand. Ushahidi was comprehensively used during Haiti earthquake for situational awareness. Sickweather[99] is another crowdsourced platform for sickness forecasting and mapping.

Sensing

Data Acquisition is an indispensable part of any research. Researchers have employed different strategies – automated, semi-automated and manual – to acquire behavioral information. Much work has been done on collecting data. [42, 41, 43] uses sensors to collect user data using various in-situ methods. The approach is to use sensors to sense user behavior and also detect the context against which a relevant question can be asked. [25] collects data from sensors and processes the data to collect high-level context information. Known as the Context Toolkit, it is a programming toolkit which provides widgets that can be used by developers to gather and process context information from sensors. [98] studies that humans with access to the web act as sensors, sharing their observations and views with web and mobile devices. WiSDM is a behavioral data acquisition system that uses humans as sensors to collect background information and pose only relevant questions to the participants. [56] provides a framework to implement context-aware system. It is based on Entity-Relationship model and stores context information at runtime. WiSDM also collects context information from users and stores them at runtime.

[64] is another work that uses sensors to detect seismic events and their influences on the population. Here they used the cellular phone infrastructure to sense seismic events and infer opportunities for assessment. The call traffic is used as a major factor to determine the impact of a disaster. The model assumes that persistent high volume of call activity several days after a disaster indicates an opportunity for assistance in those regions. The model is further extended to predict regions where persistent assistance may be required. In WiSDM we use humans as sensors to detect the impact of anomalous activity in a population. [43] discusses how various methods of incentive and how providing information on the environment both motivate the user to choose appropriate options[42]. Motivating users to gather genuine information is necessary for WiSDM as well. We have 'integrated surveys' in WiSDM that can provide additional information on the topic to the responder to increase user awareness and interest.

Crowdsensing

Mobile crowdsourced sensing[119] is a new paradigm which takes advantage of the pervasiveness of mobile phones to efficiently collect data. It extends existing web-based crowdsourcing applications to a larger contributing crowd[45, 15]. Several studies [11, 112] have evaluated that mobile phones can be very effective in public health interventions, notably in the collation of data for healthcare research. Smartphones are increasingly used to monitor and better understand the causes of health problems. [95] compares web apps to native apps for healthcare across several factors such as user interface, ease of development, capabilities and cost. While native apps provide inherent signaling mechanism to notify users, web apps reduce development time and cost[14]. However, responders can be notified via SMS from a server in case of web apps. WiSDM incorporates bootstrapping in its templates which enable it to be accessed over the Internet through a smartphone's micro browser. In case of either apps, mobile devices can potentially provide two levels of data generation modes: implicit i.e. user data captured from respective network services and explicit data collected from participatory surveillance[50]. Because they are carried by the user at all times, mobile devices are potent platforms for conducting in-situ and context based surveys (explicit data generation) whereby participants can provide real time rather than retrospective information[39]. With mobile phones and crowdsourcing, crowdsensing can provide near real-time and detailed information about a crisis situation such as a disease outbreak. Mobile4D[44] requests affected citizens to submit reports about their local situation. During the 2013 avian flu outbreak in Laos, emergency services used this application to directly communicate with the citizens in distress. [52] uses a Bayesian Network to interpret the data collected by Mobile4D for accurate and faster situation assessment at real time.

Data Quality

One potential issue when crowd becomes the source of data is data quality[22]. Researchers often struggle to obtain high-quality results from crowd workers, especially on content-creation task because such tasks cannot be easily verified. [80] suggests that unsupervised participants often do not read questions or instructions before performing a task. To solve the problem, they propose a methodological tool: Instructional Manipulational Check (IMC) which can measure attention level of participants by asking 'test' questions in the middle of experiments. [82] found that MTurkers were equally attentive as other participants when the study was short (~5 minutes) and the attention check question had an easily identifiable answer. [81] proposes an unsupervised model to assess quantitative data sent by the crowd about a public space. [72] identifies that the most important feature of truth discovery is the ability to estimate source reliability. [12, 82] found that the quality of data provided by MTurk most of the time met the psychometric standards associated with published research. [48] recommends that researchers using Mechanical Turk (1) include screening questions that gauge attention and language comprehension, (2) avoid questions with factual answers, and

(3) consider how individual differences in financial and social domains may influence results. When using factual knowledge questions, researchers may consider using simple interventions that ask participants not to look up answers. In WiSDM, we have a preamble page which serves as an introduction page where researchers can educate responders on how to perform the task. [30] proposes a shepherding system that provides peer feedback to workers such that the data quality can be improved over time. This can only be implemented in a survey system consisting of many open-ended responses. [87, 107] also suggest that splitting surveys across multiple pages and displaying fewer questions per survey page increases responder's focus and reduces bias[21] which in turn improves survey data quality. For the same reason, WiSDM surveys are designed to display only one question per page.

Interface Design

The usual design and implementation requirements of an online data acquisition system are 1. support multiple platforms and browsers[118], 2. prevent multiple submissions by same user [118], 3. present questions in a logical or adaptive manner [88], 4. provide opportunities for saving the work in long questionnaires [102], 5. collect both predefined-choice question responses and open-ended question responses [118], and, 6. thank the responder or provide feedback upon completion of the survey [102]. WiSDM has been designed to meet most of the above requirements.

2.3 Features

Below is a list of features offered in WiSDM, some of which are novel and some of which are also offered by other commercially available tools.

- Abductive Loop: WiSDM conceptualizes abductive reasoning within the domains of disease modeling. This is a novel feature that interprets the actions performed in modeling as abductive reasoning carried out in multiple loops to find the best answers to a research question. Hence, WiSDM has been expanded to areas beyond data acquisition, but is still extremely applicable to research which applies abductive reasoning.
- Question Routing (Survey Branching): With WiSDM one can create dynamic surveys. Survey designers can easily add branching logic to any survey. We support multiple patterns of branching as defined in [78].
  - Linear Pattern: Respondents proceed from Question n to Question $n+1$, irrespective of how they answer any particular question. The survey is linear.
  - Linear Skip Pattern: When a skip pattern is introduced, it means that one may go from Question n to Question $n+m$, where m is the number of questions skipped.

  - Linear Branching Pattern: In previous patterns, a set of questions is either added or omitted. In this, respondents may take different linear paths through different sets of questions depending on the conditional responses.
  - Looping Pattern: Repeat loops occur when the same set of questions is repeated for each element of some set; for example, a question set to be administered for each member of the family or for each visit to a doctor.

- Integration to crowdsourcing platforms: Currently we can seamlessly integrate WiSDM with MTurk in an iFrame. Most commercially available survey tools do not integrate with MTurk; some can be integrated, but a user must follow a series of convoluted steps in order to finish their survey.

- Longitudinal Survey: WiSDM, like other survey tools, support longitudinal studies.

- Standalone data collection system: WiSDM can be used as a standalone data collection system. The surveys and corresponding responses will be hosted on the WiSDM server.

- Integrated Surveys: This is a novel feature supported only in WiSDM. Integrated Surveys are surveys with an iFrame window in which external web pages supporting iFrame integration can be embedded. WiSDM provides an inline frame template that can embed informative sites such as CDC Flu map. [31] shows that adding interactivity in online surveys have a potential to improve user engagement. Providing relevant information within a survey page when the survey is going may not only improve responders' interest to take the survey as they will be gaining knowledge in return but such survey scenarios also elicit a study of biases introduced due to information injection during the task.

- Reporting module: This module allows an administrator to run queries on the WiSDM application database directly, and create dynamic reports based on the underlying models. It can also consolidate and aggregate data, and filter and sort it.

- Response and Summary Download: A survey administrator can download survey responses or survey summaries through the WiSDM administrator interface.

- Adapter: WiSDM can also function as an ETL[111] system. Developed as a proof-of-concept, this is a working feature and is configurable with external data sources. None of the survey tools provide this facility.

We analyzed features of 3 major survey tools, QuestionPro, SurveyGizmo, and Qualtrics. The above comparison is based on that.

2.4 WiSDM Workflow

There are two types of clients on WiSDM, first is the administrator – who has the privilege to design surveys and build reports on the collected survey data using the Admin interface. The second type is the responders who take surveys from their Internet browsers. WiSDM also provides two more interfaces, download – for downloading data and reporting – for building reports, to the administrator.

2.4.1 WiSDM Admin Interface: Build Surveys

A survey administrator is a user who has valid credentials to access the WiSDM admin interface. User login is mandatory to use the interface. When a WiSDM instance is launched for the very first time, a superuser admin is created by running the following command.

```
$ python manage.py createsuperuser
```

The command line tool prompts for a username and password to be set, along with the administrator's email address. Once the credentials are successfully set, the admin can log into the WiSDM administrator interface with those credentials.

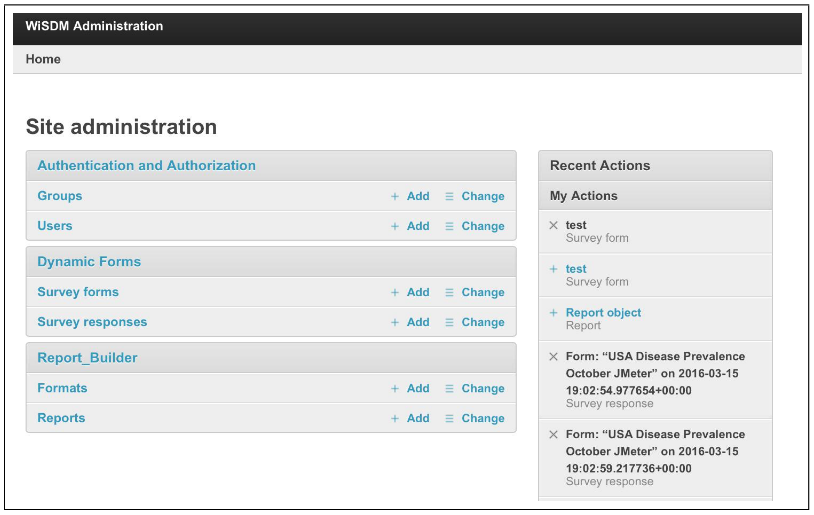
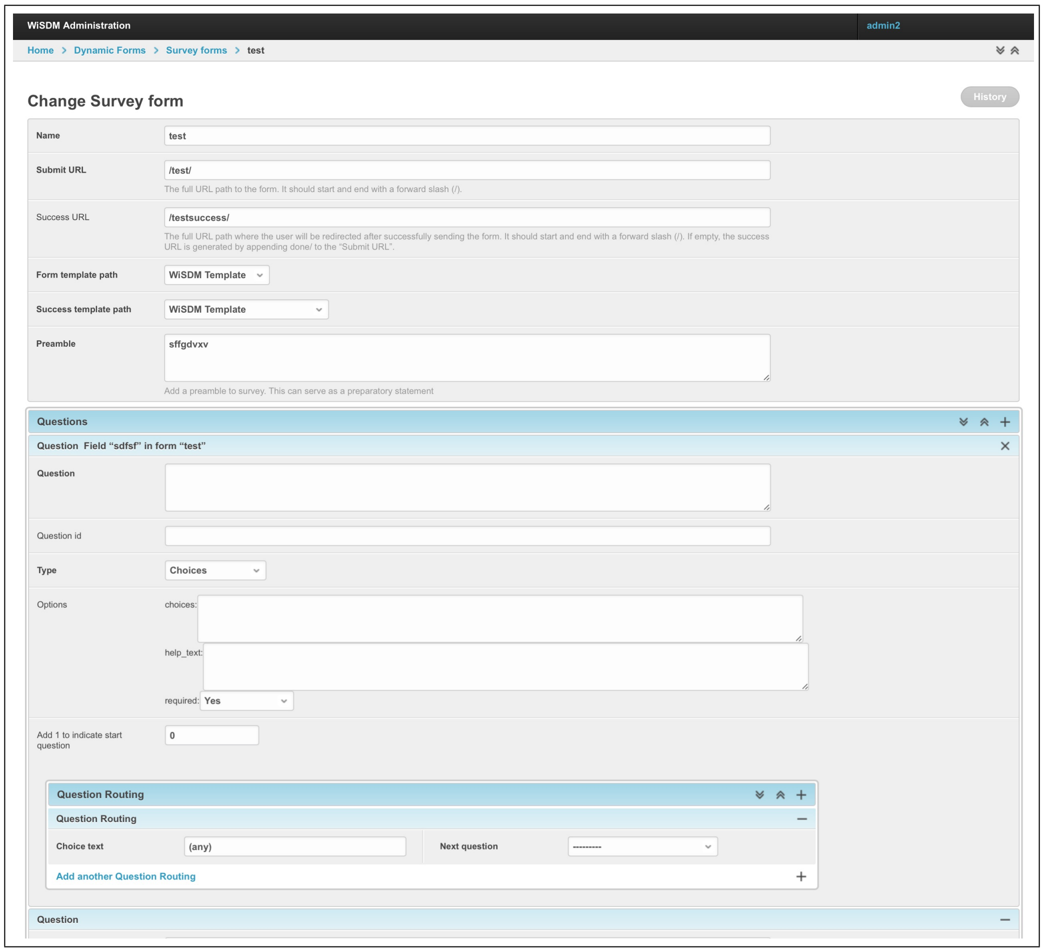


Figure 2.1: WiSDM Admin screen

The WiSDM administrator interface is a Django built-in feature which has been customized to suit WiSDM's requirements. The first page of the WiSDM administration dashboard is shown in Figure 2.1. The Authentication and Authorization section is where a survey admin can add new users and assign privileges to them. To design/build/alter surveys or edit/add responses, an admin can go to the Dynamic Forms section. They also can download/delete survey results and summaries. The Report_Builder module lets an administrator build and download survey reports with their desired database columns.


WiSDM Administration admin2

Home > Dynamic Forms > Survey forms > test

Change Survey form

History

Name	test
Submit URL	/test/ The full URL path to the form. It should start and end with a forward slash (/).
Success URL	/testsuccess/ The full URL path where the user will be redirected after successfully sending the form. It should start and end with a forward slash (/). If empty, the success URL is generated by appending done/ to the "Submit URL".
Form template path	WiSDM Template
Success template path	WiSDM Template
Preamble	sffgdvxv Add a preamble to survey. This can serve as a preparatory statement

Questions

Question Field "sdfsf" in form "test"

Question	
Question id	
Type	Choices
Options	choices: help_text: required: Yes
Add 1 to indicate start question	0

Question Routing

Question Routing

Choice text	(any)	Next question	---------

Add another Question Routing

Question

Figure 2.2: WiSDM Add Survey Form

A survey administrator can go to the survey builder page by navigating to Dynamic Forms –> Survey forms –> Add Survey Form. Figure 2.2 shows the Add Survey Form. The first section of the form is about providing the survey metadata. All fields in this section are mandatory. The admin must provide a unique name for the survey and a unique survey URL. Providing a success URL is optional, as by default Django would append a '/done/' string to the current survey URL. A separate success URL is necessary because WiSDM displays a different template for the last page. The template defines how a survey should look to a responder. WiSDM provides three template options: a default template containing a plain HTML page with a survey question, a WiSDM template which contains the survey name, copyright information and better formatting, and an iFrame template similar to the WiSDM template, but formatted to be displayed in an iFrame.

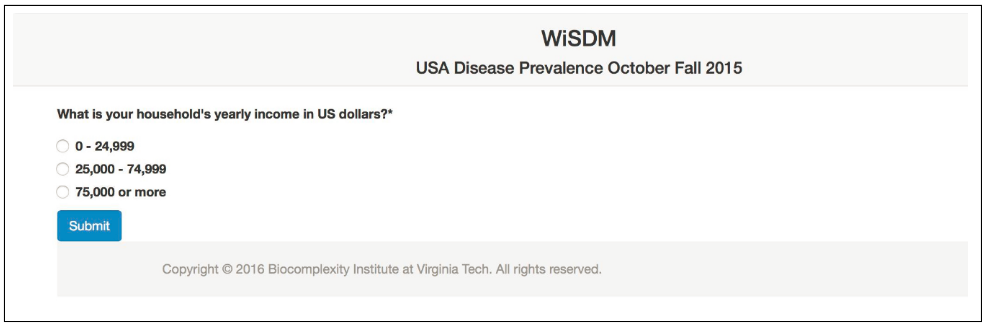
For the survey success page, WiSDM provides one additional template – the Response Summary template which provides a summary of all the submitted responses to the responder. The last field in the survey details section is the Preamble. The text entered here is displayed on the first page of the survey; it can provide additional information about the survey, why the survey is being conducted, or special instructions for the responder. The survey admin may save the survey settings at this point by clicking on the 'Save and continue editing' button or just the 'Save' button.

The survey administrator may now add Questions in the next section. Each Question is a subsection in the larger Questions section. The survey administrator enters the question text in the Question textbox, selects a question type from the Type dropdown list, and enter 1 to the small textbox labeled "Add 1 to indicate start question" if, indeed, this question should be the first one presented in the survey. The Question ID is an auto-complete field, filled in by WiSDM. WiSDM supports five question types which are single select questions, multi-select questions, short text answers, longer text answers, and integer inputs. The administrator can continue adding more questions before saving the survey again. When all questions are entered, the admin can click on 'Save and continue editing'. The questions are saved in the order in which they are entered.

After the questions are saved, the admin is presented with textboxes to enter choices and other information (help text, min/max range, required, etc) depending on the question type. The admin now adds branching information for each question in the Question Routing section. Any questions or routing information can be added or deleted using the controls in the upper right corner of each section. The admin can then click on 'Save' to save the survey.

2.4.2 WiSDM Survey Interface: Respond To Surveys

The survey responder is a person who takes the survey. They can be a crowd worker or just someone who is interested in taking the survey. They do not need to log in to take the survey. The responder must remain online during the entire duration of the survey.


WiSDM
USA Disease Prevalence October Fall 2015

What is your household's yearly income in US dollars?*

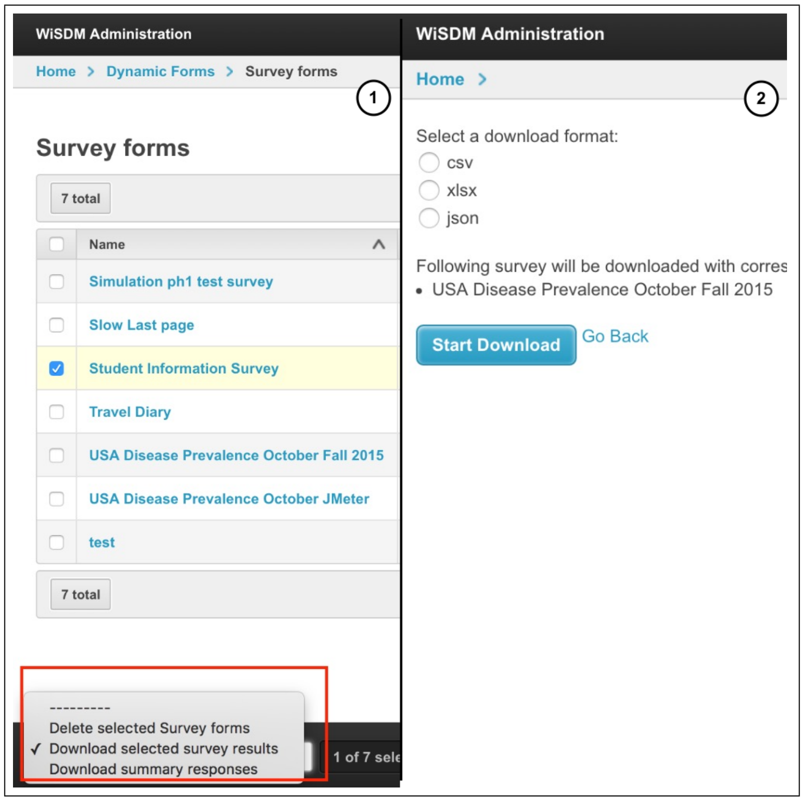
- 0 - 24,999
- 25,000 - 74,999
- 75,000 or more

Copyright © 2016 Biocomplexity Institute at Virginia Tech. All rights reserved.

Figure 2.3: WiSDM Survey Page as presented to a responder

The responder is first presented with a Preamble page that contains instructions about the survey. Only upon clicking 'Take Survey' can they start the survey. The WiSDM survey as presented to the responder is shown in Figure 2.3. The responder must answer each question and click on 'Submit' to go on to the next question. The responder may skip non-mandatory questions. After the last question in the survey is answered, the survey success page is displayed to the responder.

The previous version of the WiSDM interface provided all of the survey questions presented on one page. We changed that design to display one question per page so that a responder can focus on one question at a time, which may lead to better survey results[87, 107]. Studies also suggest that having multiple questions grouped together in one page encourages responders to derive correlations among them[21], which may introduce bias. The responders may also manipulate their responses to the current question based on other questions on the same page. This is one of the reasons why WiSDM surveys navigate only forward – the responders cannot go back and change their answers. Before moving to the next question in the survey interface, WiSDM saves the current response to the database. In this way, WiSDM collects data even for unfinished surveys.


WiSDM Administration

Home > Dynamic Forms > Survey forms

Survey forms

7 total

	Name
☐	Simulation ph1 test survey
☐	Slow Last page
✓	Student Information Survey
☐	Travel Diary
☐	USA Disease Prevalence October Fall 2015
☐	USA Disease Prevalence October JMeter
☐	test

7 total

Delete selected Survey forms
✓ Download selected survey results
Download summary responses

1 of 7 sele

WiSDM Administration

Home >

Select a download format:
- csv
- xlsx
- json

Following survey will be downloaded with corres
- USA Disease Prevalence October Fall 2015

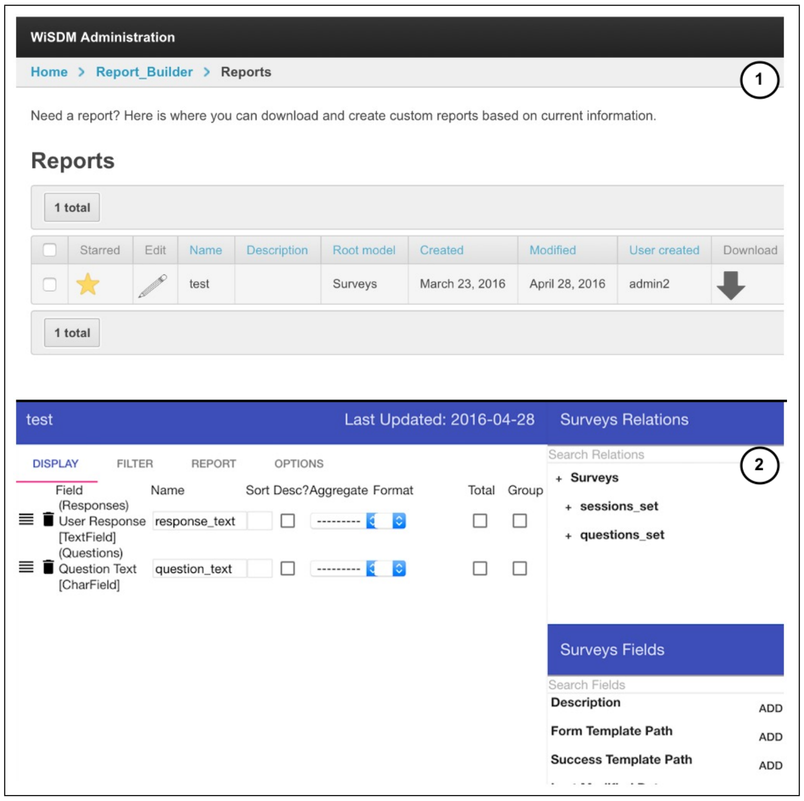
Start Download Go Back

Figure 2.4: WiSDM Download pages. 1. Admin selects a survey and chooses one of the download actions. 2. Admin then chooses a download format and clicks 'start download'.

2.4.3 WiSDM Download Interface: Download Survey Results And Summaries

Survey administrators can download complete survey results, survey responses, and survey summaries. They can select one or more surveys and choose one of the download actions at the bottom of the page (survey results download or summary responses download). The administrators are then navigated to the next page where they can choose one of the download formats CSV, XLSX or JSON. Both the pages are shown in Figure 2.4. This feature is further elaborated in the next chapter.

2.4.4 WiSDM Reporting: Build Survey Reports


WiSDM Administration

Home > Report_Builder > Reports

Need a report? Here is where you can download and create custom reports based on current information.

Reports

1 total

	Starred	Edit	Name	Description	Root model	Created	Modified	User created	Download
☐	★		test		Surveys	March 23, 2016	April 28, 2016	admin2	

1 total

test — Last Updated: 2016-04-28

DISPLAY FILTER REPORT OPTIONS

Field	Name	Sort Desc?	Aggregate	Format	Total	Group
(Responses) User Response [TextField]	response_text	☐	---------		☐	☐
(Questions) Question Text [CharField]	question_text	☐	---------		☐	☐

Surveys Relations

Search Relations

+ Surveys
  + sessions_set
  + questions_set

Surveys Fields

Search Fields

Description	ADD
Form Template Path	ADD
Success Template Path	ADD

Figure 2.5: WiSDM Reporting module. 1. Admin selects a report to edit or clicks 'Add Report' to build a new report. 2. Reporting interface where admin can build or edit reports.

WiSDM allows survey admins to create reports by adding one or more filters on the data set. It was built using Django-report-builder module. The reporting interface empowers the admins to join database tables on filter criteria without directly accessing WiSDM backend database and build reports which can be downloaded as XLS or CSV files. The reporting interface also allows simple aggregations such as sum, count, average, etc. The reporting interface is shown in Figure 2.5.

2.5 Conclusion

In this chapter, we presented WiSDM, an advanced data collection system that unites abductive reasoning with data acquisition and analysis and that can be integrated with Amazon Mechanical Turk or used as a standalone application for behavioral data acquisition. It can easily support advanced online experiments such as longitudinal studies on MTurk. It offers features such as branched logic, reporting, and download. With features like question routing and easy integration to Amazon Mechanical Turk, we prove our first two hypotheses that WiSDM is an adaptable system that poses the most relevant questions to responders and that WiSDM can be easily integrated with crowdsourcing platforms such as the MTurk. We prove our third hypothesis that WiSDM can extract data from external data sources in the next chapter. The next chapter focusses on WiSDM architecture and a WiSDM extension called the API Adapter that works as an ETL tool.

Chapter 3

WiSDM Architecture

This chapter describes the high-level design of WiSDM. It describes the decomposition of the software into components. Then, for each component, it describes the relationship to WiSDM's external interfaces and the dependencies on other components. WiSDM is a system designed and developed by the NDSSL group at Virginia Tech. It is designed to help researchers detect behavioral information to build infection models.

WiSDM is built on Django framework which is briefly discussed in the next section. The system architecture and its dependencies within the various components are described in section 2. Section 3 illustrates backend communications when a request is generated and a response is sent. The staging and new versions of the backend database are discussed in section 4. Section 5 elaborates on the specific WiSDM features that were implemented by me. Lastly, an extension of WiSDM, API Adapter is discussed in section 6.

3.1 Django Framework – An Introduction

Django is an open source, high-level python web framework designed to enable rapid web development. It is a database-driven web application which comes with an object-relational mapper[65] in which the database layer must be described. It follows the Model-Template-View (MTV) pattern of software architecture closely.

According to *The Django Book*[58],

- Model is the data access layer which contains anything and everything about the data: how to access it, how to validate it, which behaviors it has, and the relationships between the data.
- Template is the presentation layer which decides how something should be displayed on a webpage.

- View is the business logic layer which acts as the bridge between models and templates. It accesses the model and defers to the appropriate template.

However there are four primary code divisions in Django applications.

Model

Django models are traditional python classes. An Object-Relational Mapping (ORM) mapper under the hood allows these models and their instances to interact with the database. The ORM shields developers from dealing with the database directly using SQL (Standard Query Language), the standard way to access contents in a database. This not only prevents the code from quickly becoming unmanageable since developers may not be SQL experts, but also mitigates malicious attacks, like SQL-Injection, that target data-driven applications.

View

Django views are designed as standard Python functions that are called when a user requests a specific URL. A view accepts user requests, behaves according to the application's interaction logic, and returns a display that is suitable for accessing the data represented by the models. Views can access the models, retrieving and updating information as necessary to accomplish the task requested by the user. Since views are simply called as functions, without requiring any specific structure, they can take the form of any Python callable object, including classes, instance methods, and curried or decorated functions.

Template

While views are technically responsible for presenting data to the user, the task of how that data is presented is generally delegated to templates. Templates handle all the presentational details the user will see. Django provides a simple template language for this purpose, primarily used for HTML, but can also be used to generate any text-based format. Views may use HTTP redirects to other URLs, third-party Portable Document Format (PDF) libraries, or anything else to generate their output.

URL Configuration

This is a separate layer that makes views available to the outside layer. It can accommodate regular expressions as URL components to allow a view to be configured to more than one URL. Views that are used for common needs, also called generic views, have configuration options that allow them to be used in any application, requiring only a URL configuration to enable them.

Design Philosophies

The above code division closely follows Model-View-Controller (MVC) architecture, where the views and templates in MTV are actually controller and view in MVC framework respectively. One of the key features of the MVC design pattern, and of the slightly modified form followed by Django, is that sections of code that perform significantly different functions don't rely on how the others operate, i.e. sections should be loosely coupled. It is a flexible design pattern that is useful in developing a dynamic web application[71].

Django also follows the principle of DRY (Don't Repeat Yourself), which basically means code should be as reusable as possible. It reduces the risk of accidentally introducing inconsistency between two pieces of code that should match. If a block of code needs to know something about another piece of code, the information can be automatically fetched using Python, without repeating any of that information.

3.2 WiSDM Architecture

WiSDM is a data acquisition system. It can interact with various information systems on the Internet and within its environment. Figure 3.1 shows how data flows within that environment. The WiSDM client, which can be the WiSDM survey interface or the administrator interface, interacts with the WiSDM application server over the Internet. The WiSDM application server communicates with the WiSDM backend database via the Intranet. The External Web Application Server can be Amazon Mechanical Turk on which the WiSDM survey interface is displayed in an iFrame, or another third party application which could be displayed on WiSDM's survey interface in an iFrame. This display using iFrame involves the external web application server and both WiSDM client and server.

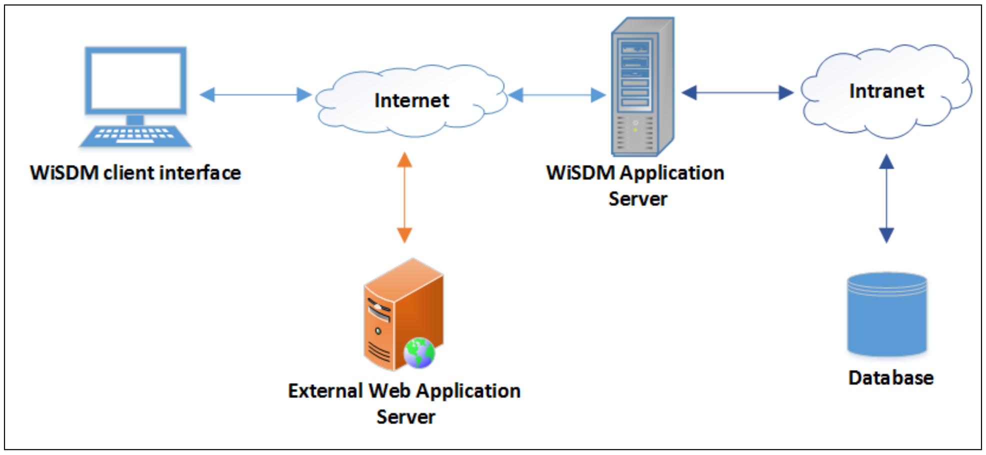


Figure 3.1: WiSDM data flow diagram

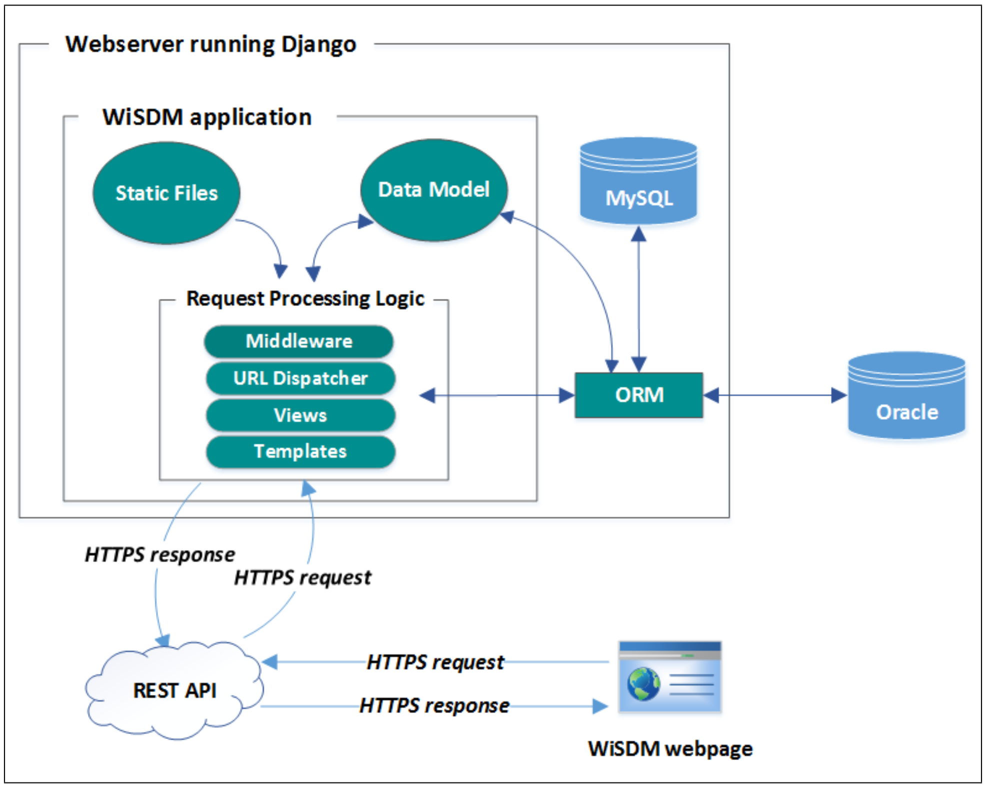


Figure 3.2: A high-level architecture diagram of WiSDM

WiSDM is built using Python 2.7 and Django Framework 1.8.5. There are many advantages to choosing Django as our web framework. It has a built-in administrator interface which has been customized for WiSDM, it has numerous packages and utilities that are ready to be used immediately on installation, and it has good documentation and an active development community. Django also has several built-in security features[59] such as CSRF protection, XSS protection, and protection against SQL injection. It is a time-tested framework that has been around for more than 10 years and is being used as a web framework for large scale sites like Pinterest and Instagram, and highly secure websites such as the official website of NASA.

WiSDM is Cross-Site Request Forgery (CSRF) protected website[8]. WiSDM is also TLS (Transport Layer Security)[26] compliant for added authentication and encryption. The production instances of WiSDM run on HTTPS. The production version of WiSDM is hosted on Apache server, version 2.4.6. The admin interface uses Grappelli (version 2.7.1), a Django module for better look-and-feel. WiSDM currently interacts with both MySQL (Version 15.1 Distrib 5.5.44-MariaDB, for Linux using readline 5.1) and Oracle (Oracle DB 12c EE Release 12.1.0.2.0 - 64Bit Production) databases.

Figure 3.2 is a high-level representation of the internal server architecture of WiSDM. The Oracle database is hosted on a separate server as we are using a shared instance of Oracle. However, the MySQL database runs on the same server as WiSDM. Running RDBMS on the same server as WiSDM reduces connection overhead and bandwidth bottlenecks, if any, in the network connecting the RDBMS with the web server. The use of an ORM (object relational mapper) enables us (developers of WiSDM) to abstract from the relational data model used by the RDBMS, both for storage and for queries. Even the creation and alteration of tables is handled by the ORM.

The web application transfers requests and responses between the client and the server using the HTTPS protocol. The HTTPS request consists of a Uniform Resource Identifier (URI) and optional GET or POST parameters. The HTTPS response consists of a status code and a message body. The WiSDM GUI on the web browser consists of a hierarchy of templates, which consist of HTML code with tags. The views define the structure of the WiSDM application, dictating what information to be sent to the user. They group units of user functionality together and define possible interactions with the model and ORM. The data models (or simply models) describe the logical grouping of data and functionality.

The WiSDM core components include Administrator module, Models that interact with the underlying database, Views which define how a survey will be presented to a responder, and the Reporting module for building ad hoc reports on the data. The component dependencies within the WiSDM Framework are shown in Appendix A. The communication between each middleware is internal to the Django framework and is not depicted in this document.

3.3 How Does WiSDM Work To Fulfill User Requests

Figure 3.3 is control flow diagram of the interactions between various components within WiSDM application. The numbers represent the sequence in which each module is called when an HTTP request is generated. Below is a step-by-step explanation of what happens under the hood.

1. An HTTP request is sent. All HTTP requests and responses go via the Django middlewares, defined in Middlewares.py.
   - Using an Internet browser, the responder enters a survey URL or performs an action, such as responding to a question on an already loaded page of that survey, thus sending an HTTP request to the Apache server hosting WiSDM.
   - The HTTP request from a responder taking a survey contains information about the current survey question that they answered, the session id, their response, and, if applicable, their MTurk HIT id, assignment id, and worker id.
2. The Request Middleware sends the request to the URL dispatcher, which maps the requested URL to a view function.
   - The URL Dispatcher runs through the URL patterns in the Urls.py file, and chooses the first one that matches the URL requested by the user
   - If the URL is successfully mapped, the dispatcher sends the corresponding view information to the Views Middleware.
3. The Views Middleware invokes a view, in Views.py, associated with the found URL pattern.
4. The view function performs the requested action, which typically involves reading from or writing to the database. It may include other tasks, as well.
5. The view class communicates with the models to interact with the database.
   - It uses data models to obtain database data.
   - The models are actually database tables and the relationships between them, as defined as python classes in Models.py
6. The data is contained in a relational database such as Oracle, MySQL or PostgreSQL, though other storage mechanisms are possible as well (XML, Text, LDAP, etc).
   - For WiSDM we used Oracle and MySQL.
   - The database connection details are specified in Settings.py

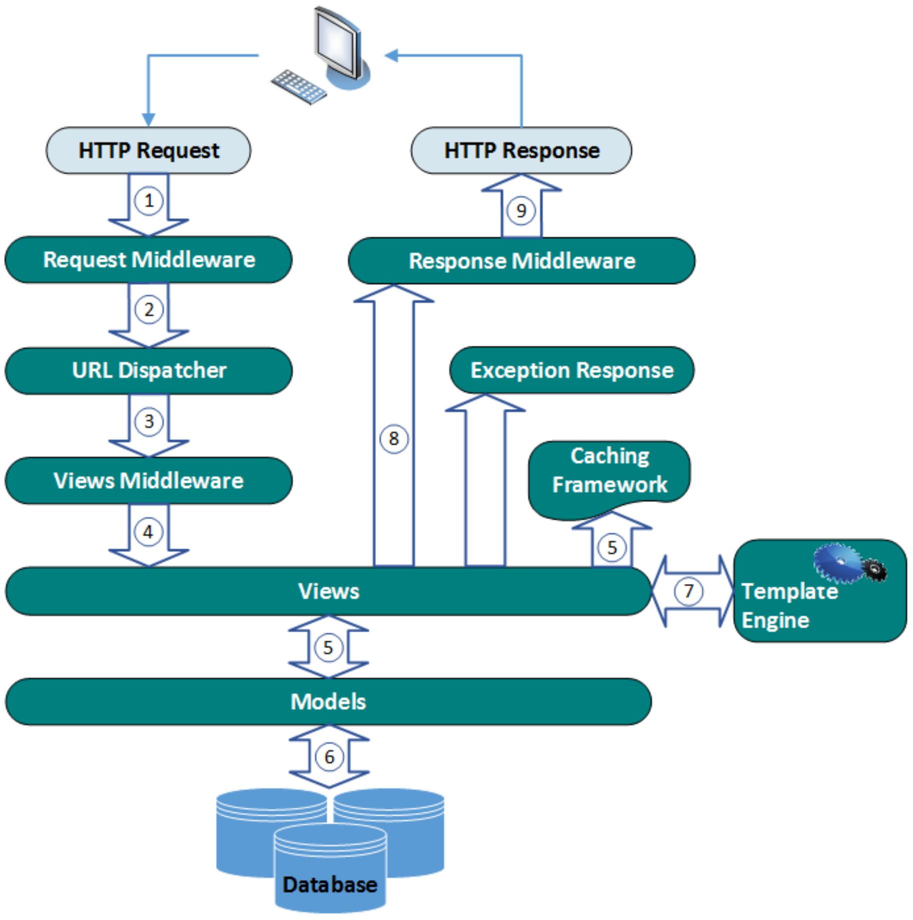


Figure 3.3: A control flow diagram of WiSDM

7. After performing any requested task, the view sends the data along with the template format to the Template engine.
    - The view also passes the context, including the obtained data mapped to template variable names.
    - The Template engine builds an HTML page from the template tags and context data. The engine can also include CSS templates from the Static folder.
8. Finally, the view returns an HTTP response object populated with the rendered template, or an HTTP exception in the event of an error, to the Response Middleware.
9. The Response Middleware then sends the HTTP response to the client.

3.4 Database Design

We have two versions of database tables for WiSDM. The older version is still being used by the WiSDM core. Newer modules in WiSDM use the newer version of the database tables, which are normalized. The older version is easy to maintain and was built to simplify data entry process. However, user responses are stored as lists in the old version, which makes it unfavorable for querying and reporting. So we introduced the new database architecture. Data is loaded into the old tables first, and we have triggers running that normalize the data and populates the tables in the new database version.

3.4.1 Older Version

The first database version had three tables: FormModel, FormFieldModel, FormModelData. Figure 3.4 is a schematic representation of the old database structure.

FormModel

The FormModel table (dynamic_forms_formmodel) contains details about a survey, such as its name, submit URL, templates, preamble, etc. Each survey is assigned an auto-generated unique survey id (field name: id) which is also the primary key. The survey name and submit URL, which is actually the survey URL, are unique fields; these two fields, along with the preamble, are mandatory fields.

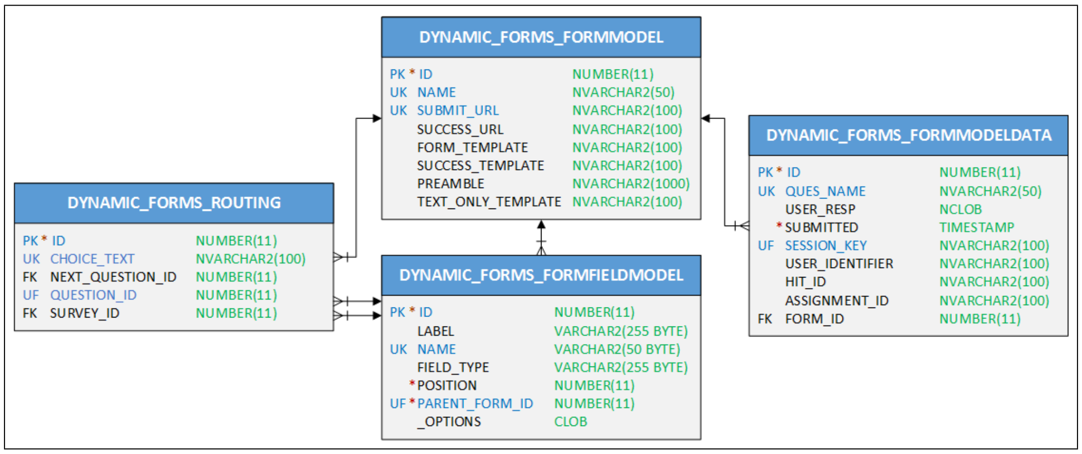

DYNAMIC_FORMS_FORMMODEL
PK * ID NUMBER(11)
UK NAME NVARCHAR2(50)
UK SUBMIT_URL NVARCHAR2(100)
SUCCESS_URL NVARCHAR2(100)
FORM_TEMPLATE NVARCHAR2(100)
SUCCESS_TEMPLATE NVARCHAR2(100)
PREAMBLE NVARCHAR2(1000)
TEXT_ONLY_TEMPLATE NVARCHAR2(100)
DYNAMIC_FORMS_ROUTING
PK * ID NUMBER(11)
UK CHOICE_TEXT NVARCHAR2(100)
FK NEXT_QUESTION_ID NUMBER(11)
UF QUESTION_ID NUMBER(11)
FK SURVEY_ID NUMBER(11)
DYNAMIC_FORMS_FORMFIELDMODEL
PK * ID NUMBER(11)
LABEL VARCHAR2(255 BYTE)
UK NAME VARCHAR2(50 BYTE)
FIELD_TYPE VARCHAR2(255 BYTE)
* POSITION NUMBER(11)
UF * PARENT_FORM_ID NUMBER(11)
_OPTIONS CLOB
DYNAMIC_FORMS_FORMMODELDATA
PK * ID NUMBER(11)
UK QUES_NAME NVARCHAR2(50)
USER_RESP NCLOB
* SUBMITTED TIMESTAMP
UF SESSION_KEY NVARCHAR2(100)
USER_IDENTIFIER NVARCHAR2(100)
HIT_ID NVARCHAR2(100)
ASSIGNMENT_ID NVARCHAR2(100)
FK FORM_ID NUMBER(11)

Figure 3.4: WiSDM Database Version - Old

FormFieldModel

The FormFieldModel table (dynamic_forms_formfieldmodel) contains all the questions and their choices related to a survey. Each question is assigned an auto-generated question id, which also serves as the primary key. The survey id from FormModel is defined as a foreign key here. The choices related to a question and its corresponding help text are appended as a JSON string in the _options field. The survey id and the question name are together defined as unique. The parent_form_id is defined as a foreign key field, which is the primary key 'id' in the FormModel table.

FormModelData

The FormModelData table (dynamic_forms_formmodeldata) contains all the survey responses and responder details (like session key). Like the previous two tables, each user is assigned an auto-generated id which is also the primary key. Responses from each user are appended to a list and stored in the user_resp field. User details like session_key, hit_id and assignment_id are also entered in this table. The ques_name and session_key are together defined as unique in this table. The form_id is defined as a foreign key field, which is the primary key 'id' in the FormModel table.

Routing

This table (dynamic_forms_routing) was added later in the old schema when we introduced question routing in WiSDM. This table determines the order in which the questions will appear when presented to the responder. It has the fields survey_id, choice_text, question and next question. The survey_id is a foreign key related to the FormModel table, and question_id and next_question_id are foreign keys related to FormFieldModel table. The question and its choices (field name: choice_text) are together defined as unique in the table.

3.4.2 New Version

We introduced a new version of the tables because the original version had limitations.

1. Since all choices and responses were entered as lists, even simple querying or building reports proved difficult.
2. Data was not normalized.
3. Maintaining history in the case of the abductive loop would have been difficult.

The schematic representation of the new version is presented in Figure 3.5. The new version normalizes the data. So data from the initial three tables described in previous sections are split across five tables: Surveys, Questions, Choices, Responses, and Sessions. These five tables are being used by the newer modules – Reporting and WiSDM Analytics. We plan to migrate to the new version completely in near future. The mapping from old tables to new tables is shown in Appendix B.

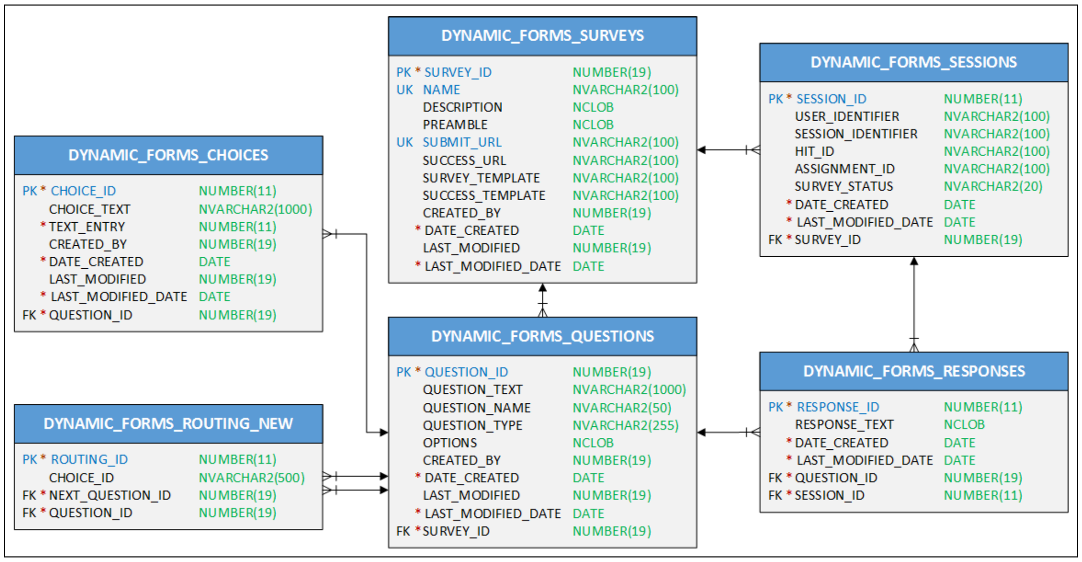

DYNAMIC_FORMS_SURVEYS
PK * SURVEY_ID NUMBER(19)
UK NAME NVARCHAR2(100)
DESCRIPTION NCLOB
PREAMBLE NCLOB
UK SUBMIT_URL NVARCHAR2(100)
SUCCESS_URL NVARCHAR2(100)
SURVEY_TEMPLATE NVARCHAR2(100)
SUCCESS_TEMPLATE NVARCHAR2(100)
CREATED_BY NUMBER(19)
* DATE_CREATED DATE
LAST_MODIFIED NUMBER(19)
* LAST_MODIFIED_DATE DATE
DYNAMIC_FORMS_SESSIONS
PK * SESSION_ID NUMBER(11)
USER_IDENTIFIER NVARCHAR2(100)
SESSION_IDENTIFIER NVARCHAR2(100)
HIT_ID NVARCHAR2(100)
ASSIGNMENT_ID NVARCHAR2(100)
SURVEY_STATUS NVARCHAR2(20)
* DATE_CREATED DATE
* LAST_MODIFIED_DATE DATE
FK * SURVEY_ID NUMBER(19)
DYNAMIC_FORMS_CHOICES
PK * CHOICE_ID NUMBER(11)
CHOICE_TEXT NVARCHAR2(1000)
* TEXT_ENTRY NUMBER(11)
CREATED_BY NUMBER(19)
* DATE_CREATED DATE
LAST_MODIFIED NUMBER(19)
* LAST_MODIFIED_DATE DATE
FK * QUESTION_ID NUMBER(19)
DYNAMIC_FORMS_QUESTIONS
PK * QUESTION_ID NUMBER(19)
QUESTION_TEXT NVARCHAR2(1000)
QUESTION_NAME NVARCHAR2(50)
QUESTION_TYPE NVARCHAR2(255)
OPTIONS NCLOB
CREATED_BY NUMBER(19)
* DATE_CREATED DATE
LAST_MODIFIED NUMBER(19)
* LAST_MODIFIED_DATE DATE
FK * SURVEY_ID NUMBER(19)
DYNAMIC_FORMS_RESPONSES
PK * RESPONSE_ID NUMBER(11)
RESPONSE_TEXT NCLOB
* DATE_CREATED DATE
* LAST_MODIFIED_DATE DATE
FK * QUESTION_ID NUMBER(19)
FK * SESSION_ID NUMBER(11)
DYNAMIC_FORMS_ROUTING_NEW
PK * ROUTING_ID NUMBER(11)
CHOICE_ID NVARCHAR2(500)
FK * NEXT_QUESTION_ID NUMBER(19)
FK * QUESTION_ID NUMBER(19)

Figure 3.5: WiSDM Database Version - New

Surveys

The Surveys table (dynamic_forms_surveys) is similar to the FormModel table with some additional columns: created_by, date_created, last_modified_date, and last_modified.

These fields are added to support survey archiving in a future release. As before, the survey_id is the primary key, and the name is a unique field.

Questions

Unlike FormFieldModel, this table (dynamic_forms_questions) contains only question information. It has a primary key field question_id, foreign key field survey_id, and other fields like question_text, question_name (auto-generated field), question_type, options (which does not contain choices), created_by, date_created, last_modified_date and last_modified.

Choices

The Choices table (dynamic_forms_choices) assigns an auto-generated id to all choices entered in the entire survey questionnaire. The choice_id is the primary key. It also references the question_id from Questions table as the foreign key. It has other fields such as choice_text, text_entry, created_by, date_created, last_modified_date and last_modified. This table is indexed by question_id and choice_id.

Sessions

This table (dynamic_forms_sessions) contains user-related information. It contains a session_identifier field which includes the session key from FormModelData table, date_created, last_modified_date and a survey status field which is by default 'Incomplete'. It will later be changed dynamically based on whether a user has completed a survey. When a survey is responded from Amazon Mechanical Turk, the MTurker's worker id, HIT id, and assignment id go into user_identifier, hit_id, and assignment_id fields respectively. Each user is assigned a session_id, which is also defined as the primary key, and the survey_id from the Survey table is defined as a foreign key.

Responses

This table (dynamic_forms_responses) contains all user responses. It assigns a response id to each user response. The response id is an auto-generated field and is also acts as the primary key. The question_id from Questions table and the session_id from Sessions table are defined as foreign keys. Other fields are response_text, date_created and last_modified_date.

Routing_New

This table (dynamic_forms_routing_new) is similar to Routing table in the previous schema and is designed for the same purpose. However, we replaced choice_text with choice_id, and we removed the relationship with Survey table. We assigned a routing_id to each branching information. No data is being inserted into Routing_New table as of now, as we are still using Routing table.

3.5 WiSDM Web Application Features

WiSDM web framework is a team project. In this section, we discuss only those features that were implemented by me.

3.5.1 Preamble:

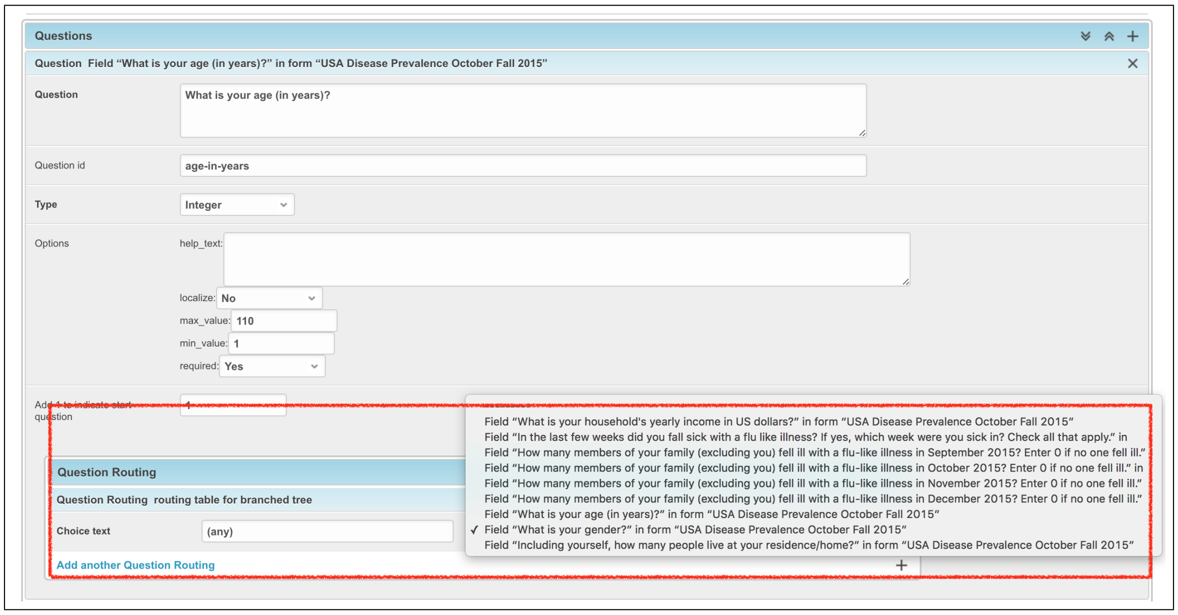
The survey preamble page is the first page in any WiSDM survey. This provides an introduction to the survey and set of rules to be followed by the responder. This is a data-driven HTML page whose contents can be edited in the WiSDM admin page by the survey administrator.

3.5.2 Branched Logic

There have been many studies that say branched, dynamic surveys lead to easier tasks for respondents and better data quality[78, 29]. It is an important step towards improving data quality as it removes irrelevant questions presented to the responders.

The previous version of WiSDM deployed all the questions for a survey on a single page. Introducing branched surveys into WiSDM was a major development effort, especially since implementation was not part of out of the box Django package. It was a major change from the previous version because the previous versions did not support dynamic survey branching. We refer to survey branching in WiSDM as Question Routing. Question Routing has been nested inside the Question section (called 'inline' in Django) in the survey administration page. We integrated a module called Super-inlines (version 0.1.4) to nest Routing inside Question inline.

In the survey admin page, for each question a in the Question section all questions are listed as 'next question' dropdown in question routing section as shown in Figure 3.6. The choice text by default is (*any*) that signifies that, irrespective of any choice selected by the responder, the selected question will follow its parent question. The survey administrator can define a choice option, say x in 'choice text' text box and select a next question b from the 'next


Questions

Question Field “What is your age (in years)?” in form “USA Disease Prevalence October Fall 2015”

Question: What is your age (in years)?

Question id: age-in-years

Type: Integer

Options
help_text:
localize: No
max_value: 110
min_value: 1
required: Yes

Add 1 to indicate start question: 1

Question Routing

Question Routing routing table for branched tree

Choice text: (any)

Add another Question Routing

Field “What is your household's yearly income in US dollars?” in form “USA Disease Prevalence October Fall 2015”
Field “In the last few weeks did you fall sick with a flu like illness? If yes, which week were you sick in? Check all that apply.” in
Field “How many members of your family (excluding you) fell ill with a flu-like illness in September 2015? Enter 0 if no one fell ill.”
Field “How many members of your family (excluding you) fell ill with a flu-like illness in October 2015? Enter 0 if no one fell ill.” in
Field “How many members of your family (excluding you) fell ill with a flu-like illness in November 2015? Enter 0 if no one fell ill.”
Field “How many members of your family (excluding you) fell ill with a flu-like illness in December 2015? Enter 0 if no one fell ill.”
Field “What is your age (in years)?” in form “USA Disease Prevalence October Fall 2015”
✓ Field “What is your gender?” in form “USA Disease Prevalence October Fall 2015”
Field “Including yourself, how many people live at your residence/home?” in form “USA Disease Prevalence October Fall 2015”

Figure 3.6: WiSDM Question Routing in Survey Admin page

question' dropdown. On doing this the admin has defined that the survey will route from question a to question b when a responder chooses x option for question a. The algorithm is defined as Algorithm 3.1. A survey administrator can also design a non-branched survey by selecting (*any*) as the choice text, along with the next question for all the questions in the survey. For the last question in either type of survey, the administrator must not add any routing information.

Algorithm 3.1 Routing condition

```
If respondent chooses 'x' for question #a, then go to question #b
Else if respondent chooses 'y' for question #a, then go to question #c
Else if '(any)' is found as a routing option for question #a, then go to question #d
Else if no match is found, then end the survey.
```

3.5.3 Response Download

The survey administrator can download responses along with the survey information. The download functionality is defined as a WiSDM Administration action, which by design, is best suited for bulk actions. An administrator can choose a survey and trigger the download action. All responses for the survey will be downloaded. But what if the administrator wants to download multiple surveys? In that case, this process becomes tedious. So defining the download functionality as an action lets someone download one or more sets of survey results in bulk. The data can be downloaded in CSV, XLSX or JSON format. The download functionality can also download one or more responses in CSV format.

The download function, defined in Actions.py, gets called when the action is triggered from the admin page, as shown in Figure 2.4 in the previous chapter. First, it performs an inner join between tables Questions and Responses on the join condition Surveys.survey_id = Questions.survey_id . The data produced is then joined with Responses with the join condition on question_id. Finally, this data is joined with the Sessions table on the session_id. The resulting data set contains survey and question information along with user details for each response.

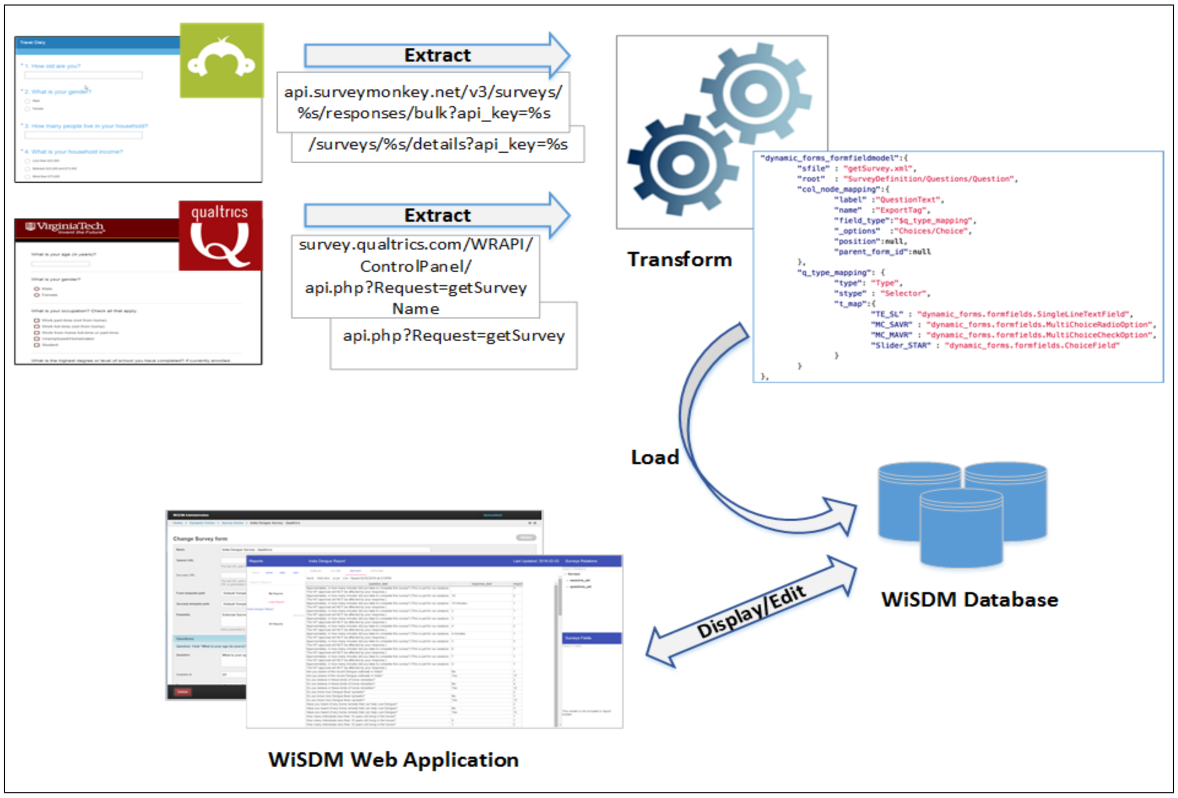
The equivalent SQL query is given below. The <user_input> is the survey or surveys selected by the user before triggering the download action in admin interface.

```
SELECT s.survey_id, s.name, q.question_id, q.question_text, q.question_type,
        r.response_text, u.session_identifier, u.user_identifier, u.hit_id,
        u.assignment_id, r.date_created, r.last_modified_date
FROM
DYNAMIC_FORMS_SURVEYS s,
DYNAMIC_FORMS_QUESTIONS q,
DYNAMIC_FORMS_RESPONSES r,
DYNAMIC_FORMS_SESSIONS u
WHERE s.survey_id = q.survey_id and q.question_id=r.question_id and
        r.session_id=u.session_identifier and s.survey_id =<user_input>;
```

3.6 WiSDM API Adapter

This is an extension of WiSDM, built to integrate ETL tasks with conventional data collection methods, such as other surveys. WiSDM is the only framework, to the best of our knowledge, that integrates data collection with ETL[16]. WiSDM API Adapter can be configured to pull data from external datasources, including – but not restricted to – commercial survey tools using REST APIs, transform the data to conform to WiSDM's database specification, then load the data into the WiSDM database.

The module has been utilized to pull data from Qualtrics and Survey Monkey, both commercial survey tools, and import that data into WiSDM. Since each data source is unique and the API support for each data source differs, the API Adapter needs to be configured to support the source API schema. Most of the configuration details are defined in the Config.json file. The workflow is shown in Figure 3.7 and the class diagram is given in Figure 3.8.


Extract
api.surveymonkey.net/v3/surveys/%s/responses/bulk?api_key=%s
/surveys/%s/details?api_key=%s
Extract
survey.qualtrics.com/WRAPI/ControlPanel/api.php?Request=getSurveyName
api.php?Request=getSurvey
Transform
Load
Display/Edit
WiSDM Database
WiSDM Web Application

```
"dynamic_forms_formfieldmodel":{
        "sfile" : "getSurvey.xml",
        "root"  : "SurveyDefinition/Questions/Question",
        "col_node_mapping":{
                "label" :"QuestionText",
                "name"  :"ExportTag",
                "field_type":"$q_type_mapping",
                "_options"  :"Choices/Choice",
                "position":null,
                "parent_form_id":null
        },
        "q_type_mapping": {
                "type": "Type",
                "stype" : "Selector",
                "t_map":{
                        "TE_SL" : "dynamic_forms.formfields.SingleLineTextField",
                        "MC_SAVR" : "dynamic_forms.formfields.MultiChoiceRadioOption",
                        "MC_MAVR" : "dynamic_forms.formfields.MultiChoiceCheckOption",
                        "Slider_STAR" : "dynamic_forms.formfields.ChoiceField"
                }
        }
},
```

Figure 3.7: Workflow of the API Adapter

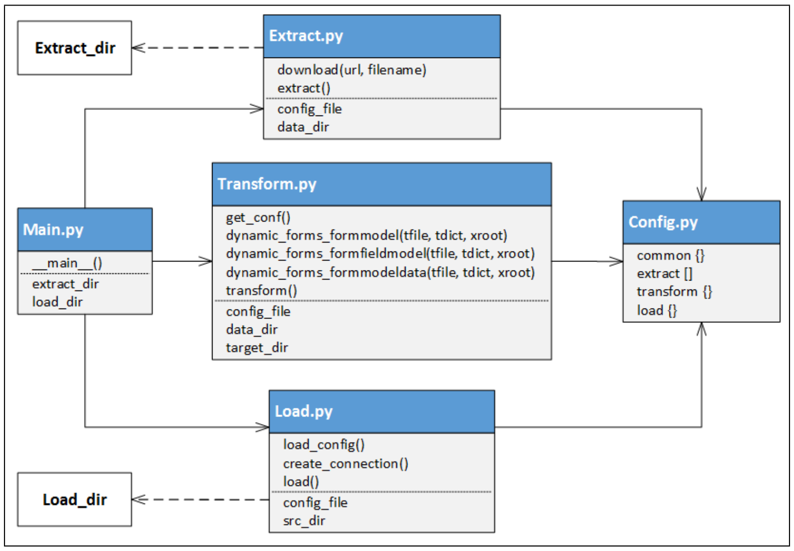


Figure 3.8: API Adapter class diagram

Config.json

The configuration file, Config.json, contains three main keys specific to each class in the module, and a common configuration set. The first section in the 'common' key contains information about the survey tool, the survey name, survey id, and the name of the user under whose account the data is stored.

The next section is 'extract', which is used by Extract.py. It contains information on the APIs from which the data needs to be downloaded, what format the data can be downloaded in, the URL of the data set, and the name of the file where the data will be saved locally. The data will be saved in a subdirectory of the current directory called ext_d.

The 'transform' section contains information on mapping columns between the tables in both data sources. For example for Qualtrics, the data set in "Results/SurveyName" in getSurveyName.xml is to be mapped to the "Name" column in the FormModel table in the WiSDM data set. Columns in WiSDM that could not be mapped to any fields in the third party data set are assigned null or default values in the case of non-nullable fields. This section is used by Transform.py.

The last section, 'load', contains information on which data from which field will be loaded into the columns of which table. Although the tables are somewhat mapped once in the transform section before, a mapping is defined here to maintain modularity among the three stages. It also contains the WiSDM database credentials. These configuration details are used in Load.py.

Extract.py

This submodule extracts data from the third party data source using the APIs specified in Config.json. The API calls also include user information, survey id, and download format, all specified in the configuration file.

Transform.py

This is the most complicated module. This maps the downloaded data to the WiSDM database schema based on the mapping information provided. The transformation type varies for each table and each API. The transformed data is saved as a CSV file. The current implementation assumes a data set that is in XML/JSON format; this may need to be extended for APIs that support different download formats.

Load.py

This module loads the transformed data into the WiSDM database based on the details mentioned in Config.json.

3.6.1 Challenges

Since each data source is disparate and unique, retrieving and ingesting such data set into the WiSDM database has many challenges which need to be addressed prior to a successful load. Here we mention a few challenges that we faced when running the API Adapter on the Qualtrics and Survey Monkey data sources.

- Each data source has a unique data structure. The API Adapter needs to be customized according to each data structure definition.
- The API Adapter also needs to be customized according to the source data format. We have customized code for XML (Qualtrics) and JSON (Survey Monkey).
- Data needs to be cleaned before being transformed into WiSDM adaptable data format.
  - Data contains special characters which can get converted to Unicode when extracted from external sources using their APIs.
  - Strings that contain single or double quotes may throw exceptions during data ingestion in the Load phase. In the INSERT statement, the Python interpreter sometimes wraps the value (string with single quotes) with double quotes. This creates an invalid INSERT statement. In such cases, values need to be inserted as list elements.
  - External data sources which are in XML format may sporadically contain HTML tags which must be cleaned.
- Datatypes, such as matrix, which are not supported by WiSDM need to be handled separately by the API Adapter. Each row in the matrix is treated as a separate question (<original matrix question>_<row string>), and all columns are the choice options for each question.
- WiSDM prevents the same responder from taking a survey more than once in order to reduce inconsistencies in the data set. However, most external tools do not have such constraints. For Survey Monkey, we used '<user ip address>_<response_id>' as user_identifier.
- Date-time conversions need to be handled. WiSDM supports date-time in '%d-%B-%y %I:%M:%S.%f %p' format.

- The length of attributes in external data sources exceed that of WiSDM. The character length of 'label' and 'question_text' attributes in the WiSDM dynamic_forms_questions and dynamic_forms_formfieldmodel tables respectively had to be increased to support Survey Monkey.

- Depending on the external survey tool, multiple APIs may need to be called to fetch actual data. In the case of Qualtrics, we used 3 API calls to get all of the relevant data, but for Survey Monkey the response data are sent as IDs, so additional endpoints (APIs) need to be called to identify the IDs.

- Handling inconsistent data formats across API calls. For example in Qualtrics, the question id in questions is inconsistent with the question id in the responses data; the question id is 4 digit code in questions and 2 digit code in responses.

3.6.2 A Proposed Future Extension

Manual mapping of two data sources is a time consuming and highly error prone task which becomes a key bottleneck in a large-scale information management systems. The fact that data sets come in various formats further aggravates the problem. [28] studied the problem of finding a semantic mapping between two given ontologies. They proposed a framework, GLUE, which employs learning techniques to semi-automatically create semantic mappings. We believe a similar approach can be taken in the near future to map disparate data sources. The paper is discussed here to facilitate future enhancement of WiSDM API adapter. This discussion is completely based on the work of Anhai Doan and his research team and bears no contribution from our end.

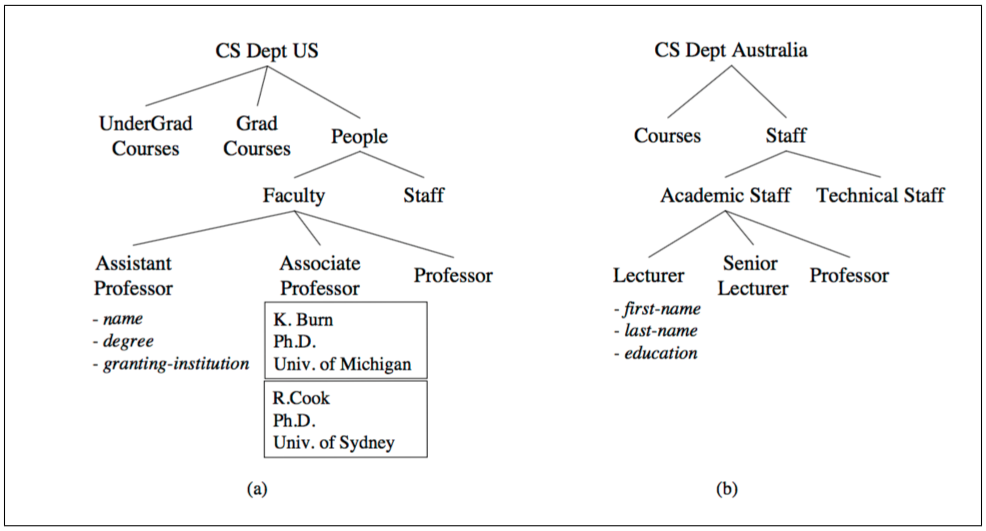


Figure 3.9: Two different ontologies: Computer Science Department in (a)US and (b)Australia

GLUE

Given two ontologies and their associated data instances as given in Figure 3.9, for each node (i.e. concept) in one ontology, we need to find the most similar node in the other ontology, for a pre-defined similarity measure. Identified challenges with any such similarity measure are:

- A well-defined similarity measure is needed.
- The similarity measure should correspond to our intuitive notion of similarity.
- Many similarity measures exist, each being appropriate to only certain situations.

To address these challenges, an architecture named GLUE as shown in figure 3.10 has been developed which applies machine learning techniques to semi-automatically create a semantic mapping between ontologies. The basic architecture of GLUE has 3 parts.

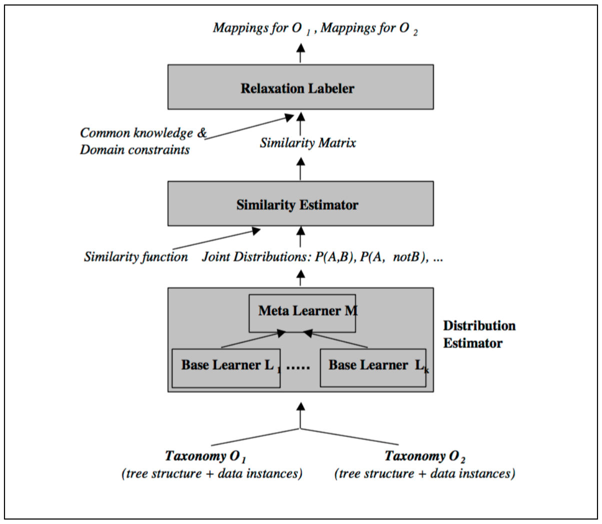

Mappings for O_1, Mappings for O_2
Relaxation Labeler
Common knowledge & Domain constraints
Similarity Matrix
Similarity Estimator
Similarity function
Joint Distributions: P(A,B), P(A, notB), ...
Meta Learner M
Base Learner L_1
.....
Base Learner L_k
Distribution Estimator
Taxonomy O_1 (tree structure + data instances)
Taxonomy O_2 (tree structure + data instances)

Figure 3.10: GLUE Architecture

Distribution Estimator

This computes the joint probability distribution between two taxonomies O_1 and O_2. For any two concepts A $(A \in O_1)$ and B $(B \in O_2)$, the joint distribution consists of 4 probabilities $P(A,B)$, $P\left(A,\overline{B}\right)$, $P\left(\overline{A},B\right)$, $P\left(\overline{A},\overline{B}\right)$ where a term $P\left(A,\overline{B}\right)$ is the probability that an instance in the domain belongs to concept A but not to concept B.

To compute $P(A,B)$, the general assumption made is that the set of instances of each input taxonomy is a representative sample of the instance universe covered by the taxonomy. So $P(A,B)$ can be estimated as,

$$P(A,B) = \left[N\left(U_1^{A,B}\right) + N\left(U_2^{A,B}\right)\right] / \left[N(U_1) + N(U_2)\right] \tag{3.1}$$

where,

U_i = set of instances given for taxonomy O_i ,

$N(U_i)$ = size of U_i ,

$N(U_i^{A,B})$ = number of instances in U_i that belong to both A and B.

The joint probability distribution between A and B is estimated as follows (illustrated in Figure 3.11).

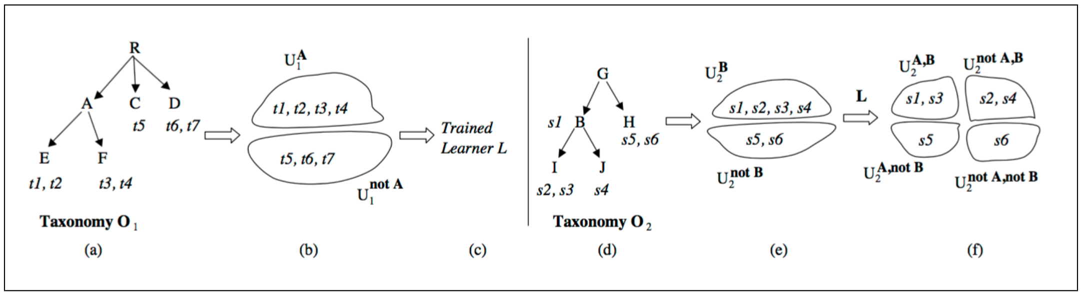


Figure 3.11: Estimating the joint distribution of concepts A and B

1. Partition U_1 ($U_1 \in O_1$) into U_1^A (set of instances that contain A) and $U_1^{\overline{A}}$ (set of instances that does not contain A).
2. Train a classifier L for instances of A, using U_1^A and $U_1^{\overline{A}}$ as a set of positive and negative training examples, respectively.
3. Partition U_2 ($U_2 \in O_2$) into U_2^B (set of instances that contain B) and $U_2^{\overline{B}}$ (set of instances that does not contain B).
4. Apply L to each instance in U_2^B. This partitions U_2^B into $U_2^{A,B}$ and $U_2^{\overline{A},B}$. Similarly, applying L to each instance in $U_2^{\overline{B}}$ results in two sets $U_2^{A,\overline{B}}$ and $U_2^{\overline{A},\overline{B}}$.
5. Repeat steps 1-4, but with roles of O_1 and O_2 reversed to obtain $U_1^{A,B}$, $U_1^{\overline{A},B}$, $U_1^{A,\overline{B}}$ and $U_1^{\overline{A},\overline{B}}$.

6. Finally, $P(A,B)$ can be computed using formula 3.1. The remaining three joint probability distributions can also be computed in a similar manner.

For Step 2, there are a variety of classifiers that we can use since different learners are better at utilizing different types of information. So GLUE takes a multi-strategy learning approach where multiple sets of classifiers $L_1, L_2, .., L_k$, called base learners, are trained with the same instances. In Step 4, their predictions are combined using a meta-learner which assigns weights to each learner (weights indicate how much the meta-learner trusts that learner). Then it combines the base learner's predictions via a weighted sum.

In the WiSDM API Adapter, we can use a distribution estimator concept to compute joint probability between the WiSDM database columns and each element in the third party data source. In the next step, the similarity estimator, each element in the external data set can be mapped to the most similar column in the WiSDM tables using a relevant similarity measure.

Similarity Estimator

The joint probability distribution computed above is the input to this stage, which computes the similarity between two concepts from their joint distribution using any user-defined method. By default, it is Jaccard similarity.

$$Jaccard - sim(A,B) = P(A \cap B)/P(A \cup B) = \frac{P(A,B)}{P(A,B) + P\left(A,\overline{B}\right) + P\left(\overline{A},B\right)} \tag{3.2}$$

Relaxation Labeler

In this final stage, GLUE uses hierarchical information in the ontology to improve matching accuracy. An example of the hierarchical information given in the paper is "if node X matches Professor and node Y is an ancestor of X in the taxonomy, then it is unlikely that Y matches Assistant-Professor". However in our data sources, there exists no hierarchy. So we believe this step can not be implemented with the current WiSDM data structure.

3.7 Conclusion

WiSDM is a data acquisition system that combines data collection functionality with ETL, which makes WiSDM a novel initiative. In this chapter, we discussed the architecture of the WiSDM web application and presented its backend database architecture. We also discussed the WiSDM API Adapter, which can retrieve data from external sources and ingest them into the WiSDM database, and we discussed a possible future extension to that feature. Introducing machine learning into disparate data-structure mapping may reduce manual efforts and inaccuracies to a considerable extent. Successfully implementing WiSDM Adapter with two different survey interfaces proved our third hypothesis which is building a framework that can pull data from external data sources and store it locally.

In the next chapter, we discuss how we load-tested the WiSDM application server to simulate situations when hundreds of responders may participate in online studies hosted on WiSDM.

Chapter 4

WiSDM Application Testing

Web Server Testing is an important step towards stabilizing any web application. Not only do we have to ensure that the application does not crash when subjected to large numbers of concurrent users, but we also must ensure that data consistency is maintained: that is, that data is written and retrieved correctly. Web application responsiveness can be evaluated through performance testing of the server and the database.

Traditional web application testing tools focus mainly on protocol conformance, broken link detection, and HTML validation[6]. WiSDM, a data-driven application, could receive hundreds of requests per second when hosted on Amazon Mechanical Turk (MTurk), depending upon worker response. So scalability testing is crucial for stabilizing the application; therefore, WiSDM was evaluated using Apache JMeter, version 2.13.

4.1 Load Testing

Performance testing encompasses a range of different tests to conduct analysis of various aspects of the system. One of the simplest ways to test the performance of a website is through load testing. Load testing focuses on simulating, and often exceeding, expected production load, throwing many concurrent requests at the site for an extended period of time. This provides information about the behavior of the system when handling specific loads of users, who might be making a number of simultaneous transactions on the system. Load Testing also draws attention to any potential software bottlenecks so that these can be fixed before they become problematic. The main reason for carrying out such testing is to measure and report the behavior of the application under anticipated live load. As the result of the testing, user response times, system state, and CPU and memory statistics are recorded. However, these results will vary depending on external factors, such as users' Internet speed; hence, there could be slight variations in the results between the simulated environment with virtual users and the actual environment with live users accessing the system.

Because load tests evaluate the server environment, it is best to run them against live production servers or a replicated instance of the production servers having identical, or at least similar, configurations. As such, the response times will most accurately reflect what the user will experience. An acceptable response time depends on the action that is being performed[7]. A user is more prepared to wait for a complex analytical calculation that could provide them a wealth of information, say on personal budget, than they are to wait for the second page of a news article to load. Usually, one should aim to keep response times to less than a second. However, response time not only depends on the application server but also hugely on the backend database performance.

4.2 Database Testing

Relational Database Management systems (RDBMS) often contain data which is updated by potentially thousands, if not millions, of users. The data travels from the user interface (UI) to the backend database (DB) and vice versa. Furthermore, the DB implements functionality in the form of database methods (stored procedures, stored functions, and/or triggers) and database objects (code instances). Whenever a certain action is performed on the front end, a corresponding Create, Retrieve, Update, Delete (CRUD) action gets invoked on the backend. With database testing, one needs to verify that the fields/forms in the front end are mapped consistently with the corresponding DB table.

Response Time is a measure of the time that starts with the initiation of a query request from the end user and ends with the query response to the end user, including the time spent at each discrete step in between. Usually, SQL queries are called underneath all requests. So all CRUD operations are tested first for consistency and data integrity during database testing.

4.3 Apache JMeter

Apache JMeter[63] is an open source, pure Java application designed to load-test functional behavior, and measure performance. It can be used as a database performance testing tool that is mainly used for testing the load on the server, as well as testing client applications[84]. JMeter may look like a browser. However, as far as web-services and remote-services are concerned, it neither executes JavaScript found in web pages, nor does it render HTML pages as a browser does. It is a platform-independent tool that performs functional, load and performance testing on many different server types – web (HTTP, HTTPS), database (JDBC, LDAP), mail (POP3), etc.

JMeter uses the Java Swing GUI widget to provide a user interface for building a test plan that simulates users' browsing patterns. The test plan can accommodate a wide variety of scenarios. JMeter stores its test plans in XML format (.jmx extension), which means one can generate a test plan using an XML editor. The tester can specify how many threads to create, and JMeter treats each thread as a user who interacts with the web application, and, in turn, applies load on the server. The tester can also specify how many times each user can loop through the test plan. Cookies can also be enabled for applications that maintain sessions and each user (thread) gets a unique session id which is maintained throughout the lifetime of the user.

JMeter also has a "record-and-reply" feature which captures browser mouse movements to generate a test plan. All JMeter Test Plans can be saved for future use.

4.3.1 Installing And Launching JMeter

Installing JMeter is pretty easy. We installed it on Ubuntu 14.04 LTS using apt-get. The commands to install and fire up JMeter are displayed below.

```
$ sudo apt-get install jmeter
$ cd /home/ananya/Desktop/apache-jmeter-2.13/bin
$ ./jmeter
```

4.3.2 Creating Test Plan

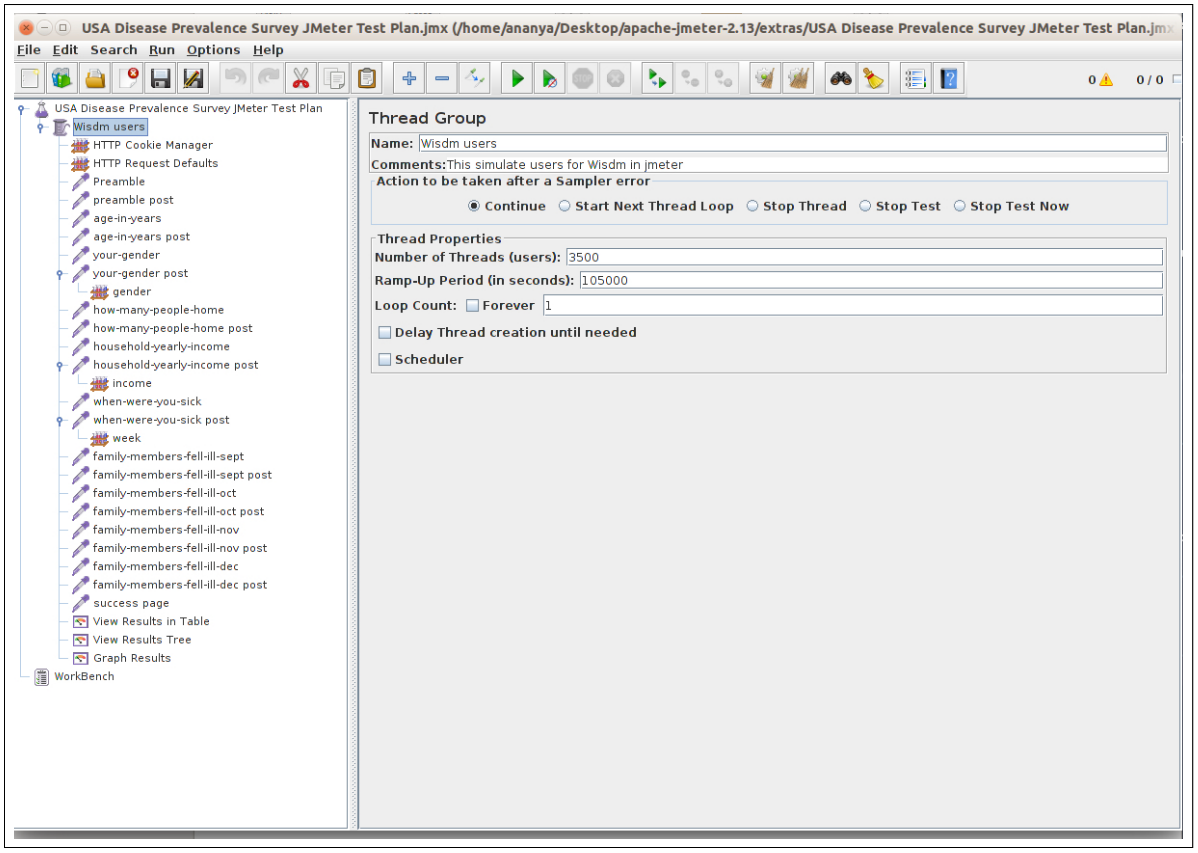
Although we used the JMeter UI to build a test plan[53], as mentioned before, these test plans are plain XML files and can be built using text editors. This section illustrates how to build a test plan for WiSDM. The test plan was built for the survey 'USA Disease Prevalence October Fall 2015'. Figure 4.1 is a screenshot of the JMeter Test Plan. The backend database for this survey is Oracle Database 12c Enterprise Edition.

The JMeter launches with a default Test Plan and workbench which can be modified as required. The test plan for this survey was named 'USA Disease Prevalence Survey JMeter Test Plan'.

- The first step for building a test plan is to add a Thread Group element. The thread group tells JMeter the number of users the tester wants to generate. A thread group can be created by right clicking on Test Plan –> Add –>Threads –> Thread Group. For WiSDM, the thread group properties were:

Name: Wisdm users
Comments: [optional]
Number of Threads: 3000
Action to be taken after a Sample error: Continue
Ramp up Period(in seconds): 9000
Loop Count: 1

We simulated up to 3000 threads with a ramp up period of 9000 seconds. Each new user was spawned in a maximum of 3 seconds. Since in WiSDM we prohibit the same user from taking a survey more than once, the loop count is set to its default value, 1.


USA Disease Prevalence Survey JMeter Test Plan.jmx (/home/ananya/Desktop/apache-jmeter-2.13/extras/USA Disease Prevalence Survey JMeter Test Plan.jm

- USA Disease Prevalence Survey JMeter Test Plan
  - Wisdm users
    - HTTP Cookie Manager
    - HTTP Request Defaults
    - Preamble
    - preamble post
    - age-in-years
    - age-in-years post
    - your-gender
    - your-gender post
      - gender
    - how-many-people-home
    - how-many-people-home post
    - household-yearly-income
    - household-yearly-income post
      - income
    - when-were-you-sick
    - when-were-you-sick post
      - week
    - family-members-fell-ill-sept
    - family-members-fell-ill-sept post
    - family-members-fell-ill-oct
    - family-members-fell-ill-oct post
    - family-members-fell-ill-nov
    - family-members-fell-ill-nov post
    - family-members-fell-ill-dec
    - family-members-fell-ill-dec post
    - success page
    - View Results in Table
    - View Results Tree
    - Graph Results
- WorkBench

Thread Group

Name: Wisdm users

Comments:This simulate users for Wisdm in jmeter

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users): 3500

Ramp-Up Period (in seconds): 105000

Loop Count: ☐ Forever 1

☐ Delay Thread creation until needed

☐ Scheduler

Figure 4.1: Apache JMeter Test Plan

When more than 3000 users were used in the current machine configuration, the JMeter threw an OutOfMemoryError. In the event of an error, JMeter will continue to run the test plan as configured in 'Actions to be taken...'. However, the errors will be written to the log file if any occur. This is so configured because WiSDM is designed to accept incomplete survey responses.

- The next step is to add Default HTTP Request properties. It can be added by right clicking on Thread Group –> Add –> Config Element –> HTTP Request Defaults.

Here one can specify default settings of their HTTP requests. For WiSDM, the properties were:

```
Server Name: epics.vbi.vt.edu/wisdm_analytics
Path: /usaDiseasePrevalenceJmeter/
Protocol: HTTPS
```

The server name is where all the requests will be sent. The path is the survey path. If there are multiple paths for the same survey, this field may be left blank. WiSDM uses HTTPS for its requests. The default for the Protocol parameter is HTTP.

- WiSDM uses cookies to transfer data from one survey page to another. To add cookies support, one simply needs to add an HTTP Cookie Manager to each thread group. This will ensure each thread gets its own cookies and is shared across all HTTP Request objects generated by the thread. To add the cookie manager, right click Thread Group –> Add –> Config Element –> HTTP Cookie Manager.

- In our Test plan, we made two HTTP Requests (GET and POST) for each survey page. Each request must be added as an HTTP Request component in the tree. We noticed that if one request is made for both GET and POST, the HTTP Response message gets lost and it becomes difficult to debug in case of any error. So segregating the request from the response keeps the test report clean. To create an HTTP Request, right click Thread Group –> Add –> Sampler –> HTTP Request. The properties set for the WiSDM test plan are:

```
Name: <question_name>_<get/post>
Method: <get/post>
Path:
Follow Redirects: [unchecked]
Redirect Automatically: [unchecked]
Use KeepAlive: checked
Parameters: [As per page requirements]
```

There are 9 questions in the survey. We added 21 HTTPS requests (2 GET and POST Requests for the Preamble page, 9 GET and 9 POST Requests for each question, 1 GET Request for the survey success page). The convention followed for naming each HTTP request was the question name it is associated with and whether it was a GET or a POST request. Accordingly, the Method parameter also had to be selected. Because WiSDM surveys are built such that the URL doesn't change with the change in survey page/questions, the Path field needs to be left blank for most of the pages. The Test Plan will pick up the default path mentioned in the HTTP Request Defaults set earlier.

The "Follow Redirects" and "Redirect Automatically" option must be unchecked because the GET and POST requests are sent separately in the test plan. However 'Use

KeepAlive' must be checked as this keeps the session alive for the next request. Some of the POST messages have parameters, and, for some, the parameters were to be randomized. The parameters were added in the Parameters section, and random functions or tester-defined variables were added as child nodes to the Request . The parameters can be defined as follows:

- To populate fields of any Request, JMeter functions can be used. A function looks like this

```
${__functionName(var1,var2,var3)}
```

- To add a value from a range, such as for questions like "Including you, how many members are there in your family?", add the expression below to the corresponding parameter.

```
${__Random(1,6,)}
```

  The __Random() function will generate a number between 1 and 6. In WiSDM, the range is chosen between a minimum and maximum accepted number.

  – To randomly select one of the available choices for a single-select option, say for a question like "What is your yearly household income?", 'User Defined Variables' must be added as child nodes to the HTTP Request. It can be added by right-clicking on HTTP Request –> Add –> Config Element –> User Defined Variables. The following code snippet under Variables and Parameters must be added for User Defined Variables and the HTTP Request respectively.

```
Variables:
Name: income_1, Value: '0-24,999'
Name: income_2, Value: '25,000-74,999'
Name: income_3, Value: '75,000 or more'
Parameters:
Name: household-yearly-income,
Value: ${__V(income_${__Random(1,4,)})}
```

- The last HTTP Request that we need to add is for rendering the WiSDM survey success page. This is a GET request where the URL path changes. The Path field in this Request is set to '/usaDiseasePrevJmeterSuccess/'. The Path parameter set in this page will override the Path parameter set before in HTTP Request Defaults. All other fields can be left as it is.

- The final element that needs to be added in the Test Plan is a Listener. This element is responsible for storing results of all the HTTP Requests and presenting a visual model of the data. For this survey, three Listeners were added: View Results in Table, View Results Tree and Graph Results. To add each listener, right click on Thread Group –> Add –> Listener. The output can also be saved to a file.

- A JMeter plugin Graph Generator Listener was added to generate graphs at the end of the test. To do that , a "fake" test was created to trigger a listener on existing file. The Graph Generator listener must be added to the fake test. The Graph Generator is available in JMeter-Extras package.

4.4 Results And Analysis

We ran our tests on a desktop computer with 7.7 GB memory, Intel Core i7 3.4 Ghz CPU, and Ubuntu 14.04 LTS 64-bit Operating System.

WiSDM supports a REST API, so all the POST requests generated to load data into the database were via the Django Application Server. JMeter simulated 13,856 users to generate 89,372 responses for the survey 'USA Disease Prevalence October Fall 2015'. The data was collected through multiple runs of the survey; a maximum of 3000 users was scheduled to respond to each run. The questions asked in the survey are given in Appendix C. The plots show the results at a state when the database was already filled with 80000+ responses. Figure 4.2 shows more than 1200 threads active at one point in time. However, the latency of some of the pages increased exponentially as more data was loaded into the server and the number of hits per second increased.

Figure 4.3 shows a plot of the response time over percentiles, which means at each point (P, V) P percent of the samples were below V ms. The black line on top represents the success page (i.e. the last page in the survey). This page was rendered the slowest, with average latency as high as 22 minutes. With the current database state and manually taking the survey with just one active thread, we noticed the average latency of the success page to be above 3 minutes. The other pages were rendered quickly, with overall latency less than a second. In both manual and JMeter tests, latency is measured from just before sending the request to just after the first response has been received. The time includes all the processing needed to assemble the request.

We tried to reproduce this issue again after a month under the exact same test conditions as before with JMeter. This time, we ran the JMeter test under stricter conditions: 3000 users, ramp up period 8000 seconds. Unfortunately, the issue of high latency in rendering the success page could not be reproduced. The latency had dropped significantly, with the entire test finishing in under 2 hours instead of the several hours it took to run the previous test. We continuously tested the application performance by responding to the same survey

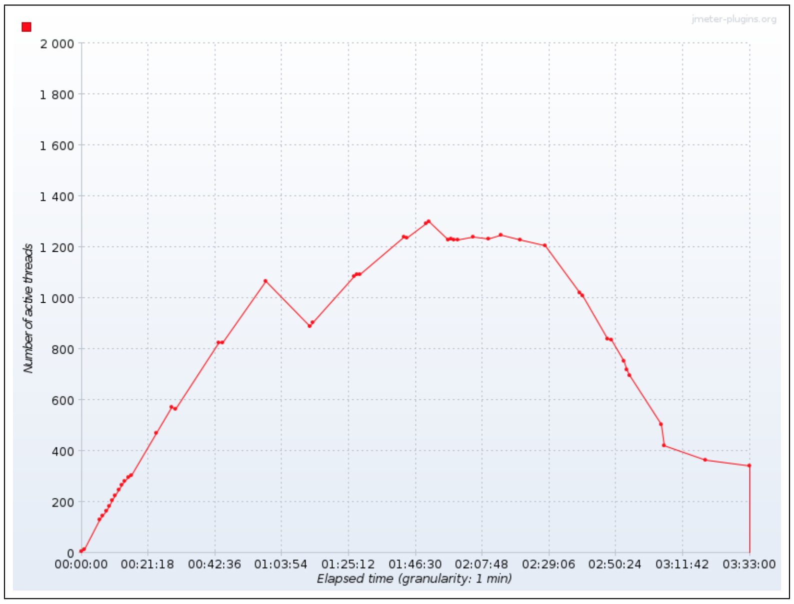


Figure 4.2: Number of Active Threads over Time

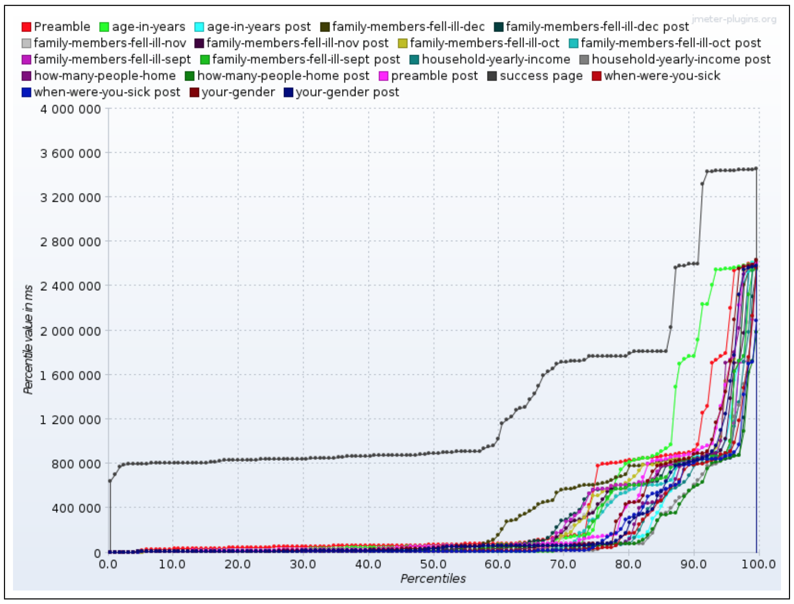


Figure 4.3: Response Time over Percentiles

manually before, during and after the JMeter test. In all of these cases, the WiSDM server rendered each survey page in less than 1 second. We concluded that the recent tablespace increase in the database from 1GB to 4GB might have stabilized the server. In the earlier test conditions, the database tablespace was almost full, which might have triggered higher latencies.

4.5 Quality Assurance

Testing a web application such as WiSDM is a complex task as there are HTTP-level request handling, form validation and processing, database interaction, and template handling. As discussed in this chapter, we used JMeter to test WiSDM server and database. While doing so we communicated with all components of WiSDM web application using HTTP request mechanism. This process not only load tested the server and the database but also the entire communication network within the framework. The test suite communicated with system end-to-end. The forms in the user interfaces were implicitly tested in the process, as they are validated by the framework for every HTTPS request.

WiSDM was subjected to both automation and manual tests. The automation test environment consisted of both deterministic and non-deterministic tests. Deterministic test suites required that the framework subjected to exact test environment iteratively succeeded every time. The non-deterministic test environment covered more areas including boundary conditions because non-deterministic inputs were fed into the system which yielded a different test environment every test run. The non-deterministic inputs are the ones that are not decided beforehand and are chosen uniformly at random from within the boundary conditions. In order to diminish sporadicity in test results, seeds were introduced with non-deterministic inputs. For example in non-deterministic test, response to a question 'What is your age?' can be any number within the defined boundary condition which is $[1, 99]$, both inclusive. The selected response is a non-deterministic input to the system. But in the case of deterministic test, the system has only one computation path and hence the response input, a hard-coded number in the test suite, will be same for every test run. The performance reports in Figure 4.2 and Figure 4.3 are results of non-deterministic tests.

Under manual test conditions, WiSDM was tested for functionality validations. Functional Testing was done both under staging and production environment. Staging site is a replica of production instance and was used to test and review newer versions before being pushed to the production instance. For example, we manually tested question routing by mapping the survey flow in WiSDM survey page with the question routing flow defined in WiSDM admin page and the routing information stored in WiSDM database. Incidentally, in this example along with functional testing of question routing, we are also doing a Semantic Testing of the data displayed across three components of WiSDM. Semantic Testing in WiSDM was also done in automation testing. In case of any discrepancy in the data that was being communicated within the framework, error message would be displayed and the test case would fail. In WiSDM, the software instance used by the client is treated like a production instance.

Invocations in WiSDM application or for that matter most web services are asynchronous by nature because service providers are capable of accepting requests from clients without prior notice. But apart from that, WiSDM is a synchronous web application since web based surveys are by design synchronous[5]. WiSDM uses HTTPS, which is a transport for synchronous operations, for web service communications. Transports that support asynchronous web message exchanges inherently support the correlation of response messages to request messages. WiSDM can be integrated with a job queue broker such as Celery to make the application asynchronous. However the nature of task that WiSDM currently performs, necessity to build an asynchronous application is trivial.

WiSDM closes databases connections after every request is fulfilled and thus does not retain any idle connection to database or overload database with numerous open threads. WiSDM under this configuration has been successfully tested with little more than 1200 concurrent requests which is the observed load when studies are conducted on MTurk. But if the WiSDM application server encounters a much larger concurrent request load and becomes unresponsive, a number of connections will remain open till the database terminates them. Depending on the number of idle requests, the database performance can be severely affected and can potentially lead to a race condition if the application server is rebooted while the database still retained idle connections. This is a non-deterministic situation and hence the application and the database server must be configured based on the anticipated/observed load.

Our primary objective of testing WiSDM application has been to ensure that it satisfies the design and implementation of use cases from the perspective of a survey administrator and a responder. We tested following key functionalities.

- Admin interface,
- Survey interface,
- Branched logic,
- Download action,
- Reporting,
- Accurate and consistent data flow among web interface, server and backend database,
- Data correctness between old database structure and new normalized database structure, and
- Ability to handle hundreds of concurrent HTTP requests.

Nevertheless there are certain areas of testing that has not been not covered in the current implementation of WiSDM. For example, white-box testing techniques such as Unit Testing. WiSDM being a Django application, 'unittest' module in Python is supported in the framework. This module uses assert functions to verify expected results. To run tests with other unit test framework Django provides APIs that can be integrated with those frameworks.

4.6 Conclusion

From the results in section 4.4, it is noticeable that the application server performance deteriorated under some irreproducible condition which we infer was due to low tablespace in Oracle. We noticed that the server performs significantly better now that the tablespace has been increased. This test should suffice that WiSDM application server is robust enough to handle hundreds of concurrent actions, a necessary feature when we use human sensors for data acquisition and in MTurk, the accept assignment rate can be quite high. So 3000 responders is a reasonable test.

Chapter 5

WiSDM: Synthetic Data Generator

Synthetic data sets can be useful in a variety of situations, including providing realistic data to test inference models. Researchers, engineers, and software developers can test against a safe data set without affecting or even accessing the original data, insulating them from privacy and security concerns, as well as letting them generate larger data sets when required. But to be adequate substitutes for real data, the quality of the synthetic data sets needs to conform as much as possible to realistic data.

We introduce an agent-based modeling platform, Synthetic Data Generator, that utilizes the Network Dynamics and Simulation Science Laboratory's (NDSSL's) synthetic Global Population[10] and their various demographic attributes to simulate virtual responders who can "take" surveys to answer behavioral/non-behavioral questions. We used data from one of our recently conducted surveys to train the classification model; every question and corresponding choices in the survey are assigned predefined probabilities to direct how the virtual responders will respond to the survey. We analyzed our training data to determine these probabilities empirically.

The synthetic population is then classified into multiple behavioral categories based on demographic information. To classify the synthetic population, we used decision tree algorithm[94]. The global synthetic population was then modeled to "respond" to the same survey based on their demographics and the pattern recognized by the trained classifier.

In this section, we give an overview of the design and architecture of the Synthetic Data Generator, which is implemented as an external module of WiSDM that uses the synthetic population to generate survey responses. The specific goal is to design and develop a system to explore the feasibility of synthesizing data sets for testing the effectiveness of an inference model. The data generator can be used not only in the evaluation and testing of inference models but also in testing the WiSDM application server and the WiSDM Analytics module.

5.1 Objective

Our vision is of an experimental platform that can generate data sets containing user-defined epidemic scenario information as well as background information, and that readily permits the imposition of additional data structures and conditions on data attributes. Our synthetic generation design is based on the use of Decision trees to assign behavior to the members of the synthetic population. Decision trees have long been used in the field of epidemiology to identify individuals at higher risk of developing certain diseases[66], or to identify the target population groups for health interventions[108]. For this project, decision trees have been used as a model for a sequential decision problem under uncertainty.

A key aspect of our objective to synthesize survey data is to provide a scalable design that can be adapted to improve data quality, assign/remove behaviors, and add other constraints on the attributes dictated by the epidemiologists.

5.2 Related Work

An important issue in evaluating inference models is the availability of representative data. When real-life data are hard to obtain, or when their properties are hard to modify for testing various algorithms, synthetic data becomes an appealing alternative. Due to the dearth of suitable data, along with privacy and security concerns, [47] discusses a general approach for building synthetic data. When simulating human agents, they restricted their approach to behaviors that are measurable; for example, the number of times per day a real user copies files to a removable drive. A similar approach is taken by [74], who built rules around behaviors that govern the generation of data elements. These rules in a specific situation are represented by semantic graphs in XML format. This approach is particularly interesting to our problem domain because we are trying to generalize a rule-based system towards behavior definition. [116] closely resembles our work on generating realistic test data. Their test is done in several phases; the kind of data set to be generated depends on the threat and the environment. Once the threat data and the environment data is generated, both are blended to assemble the final data set. Our approach differs from this because we plan to instill risk attributes into the synthetic population defining their behavior in a threat situation; their reactions to a certain situation will be used to generate our synthetic data.

Tree-based synthesizers are widely adapted for generating partially synthetic data. [93, 32] evaluate classification and regression trees to generate partially synthetic data from observed data. Examples of the generation of synthetic data through various means for testing purposes are available in many other areas of research. To test cursive hand-writing recognition systems, synthetic handwriting data has been generated using random perturbations of actual handwriting[110]. Alternatively, the US Census Bureau utilizes synthetic data-technics as a means of protecting the confidentiality of the public while maintaining statistical quality[62]. In the literature of software testing, many methods have been studied on the automation of test data generation[67]. Because of the controlled environment that synthetic data provides, several research areas are paying more attention to developing data generation systems aimed at systematically generating synthetic data for numerous applications.

5.3 Synthetic Population

NDSSL has constructed a Synthetic Population[10] of 800+ million people from 13 countries. The U.S. data is generated from the 2009 version of the American Community Survey. The synthetic people predominantly represent the urban populations, detailed at the individual level, including their activities that last for longer and shorter periods of time, as well as what other individuals make up their households and what contacts they have throughout the day.

The data sets are available for academic purposes from NDSSL OpenData and are released under a Creative Commons Attribution Noncommercial ShareAlike license.

5.4 Decision Tree Algorithm

Decision trees (DTs) are a supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

DT is a classification procedure that recursively partitions a data set into smaller subdivisions on the basis of a set of tests defined at each branch (or node) in the tree (Figure 5.1). The tree is composed of a root node, a set of internal nodes (splits), and a set of terminal nodes (leaves). Each node in a decision tree has only one parent node and two or more descendant nodes. The set of nodes at a given level in the tree, i.e., where all of the "leaves" are at the same "distance" from the root, is called a layer. Each leaf node is assigned a class label (in the figure, [A, B, C]).

A DT defines a relationship between the values of input fields (known as predictors) and an output field (known as the target) where the relationship takes the form of a tree. Internal nodes of the tree consist of decisions that depend on the value of predictor field(s), and, based on that, select a particular sub-branch of the node. The leaf nodes of the tree represent predictions about the target value. For a classification tree model, the target field contains a fixed set of outcomes (for example, "gender"), and the leaf nodes of the decision tree specify the most likely outcome from the set of possible outcomes.

In particular, decision trees are strictly non-parametric and do not require assumptions regarding the distributions of the input data. In addition, they handle non-linear relations between features and classes, allow for missing values, and are capable of handling both numeric and categorical inputs in a natural fashion[35, 85]. Finally, decision trees have significant intuitive appeal because the classification structure is explicit and therefore easily interpreted.

However, biased trees are created in the case that some classes dominate. It is therefore recommended that the data set is balanced prior to fitting the decision tree. It also tends to overfit on data with a large number of features.

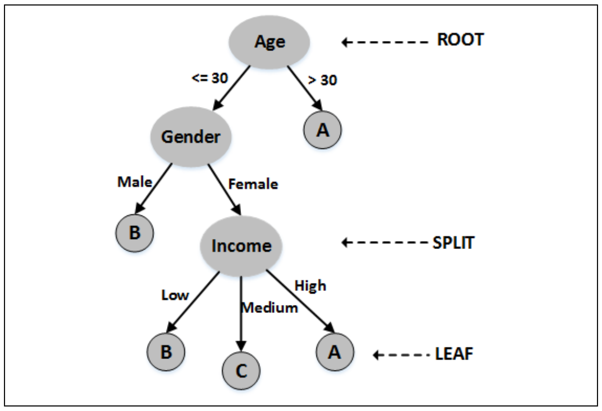


Figure 5.1: Decision Tree

5.4.1 Extra-Trees Classifier

The scikit-learn package[85] includes an averaging algorithm based on randomized decision trees: the extra-trees method. This creates a diverse set of classifiers by introducing randomness to the classifier construction. The prediction of the ensemble is given as the average prediction of the individual classifiers. Each tree in the ensemble is built from a sample drawn with replacement. A random subset of candidate features is used, and thresholds are drawn at random from each candidate feature; the best of these randomly-generated thresholds is picked as the splitting rule. This usually reduces the variance of the model while slightly increasing the bias. The function to measure the quality of a split is “gini” for the Gini impurity[90] and “entropy” for the Information Gain[90].

DTs are high variance models. The extra-trees classifier uses bagging to decrease variance in the model. In bagging, multiple versions of the training data set are created by randomly selecting data instances with replacement, creating a new data set each time. The DT is then run on each version of the data set, creating a different tree every time. This is called bootstrapping. So, if there are 50 versions of the data set, 50 trees are created. The DTs are then run on the test data set, and the most popular output from all these trees become the output for the classifier. We used this ensemble version of DT as our classification algorithm to reduce overfitting.

5.4.2 Complexity

The time complexity of a balanced decision tree is $O\left(n_{samples} n_{features} \log\left(n_{samples}\right)\right)$ and query time $O\left(\log\left(n_{samples}\right)\right)$. In a more practical case where the subtrees are approximately balanced, the cost of the tree is $O\left(n_{features} n_{samples}^{2} \log\left(n_{samples}\right)\right)$.

Scikit-learn pre-sorts the feature over all relevant samples and retains the running label count, reducing the complexity at each node to $O\left(n_{samples} n_{features} \log\left(n_{samples}\right)\right)$.

5.4.3 Data

The observed data set used to fit our decision tree model was collected from a survey data set 'USA Disease Prevalence October Fall 2015' conducted last year via Amazon Mechanical Turk. The questions asked in the survey contain demographic information, as shown in Appendix C. To fit our decision tree model, we used the following features: age, gender, household income, and number of family members. The household income and gender attributes are categorical. For the test data set, we used a synthetic population with same features as the responders to our live survey. However, household income in the original test data set was nominal, which we converted to categorical before feeding it to the model.

We collected a sample of 100 respondents for our training data and a sample of 500 synthetic people for the unseen test data. These 500 people were taken randomly from five states: 100 people each from Alabama, Virginia, California, Washington and Kansas in our synthetic population database. Only one person per household was chosen i.e. no two people were allowed to have same household id and all participants are above 18 years of age.

For this data set, the training data was manually categorized into 'high', 'medium' or 'low' risk bins based on the following rules:

- Responders who live in a household where no one is sick: low
- Responders who are or have family members who are sick: high
- Responders who are not sick and in the lowest income bracket with no additional family member: medium

The income brackets (in USD) used in both data sets are:

- 0-24,999: low income

- 25,000 - 74,999: medium income
- 75,000 or more: high income

5.5 Design And Implementation

The Synthetic Data Generator has been designed to classify synthetic people into 'low', 'medium' or 'high' risk bins depending on their background information. The information used to classify the population is parameterized. Also, the model used by the Data Generator for classification is plug-and-play, meaning that, if needed, the model can be changed (defined in DecisionTree.py) without breaking any other part of the code.

The Synthetic Data Generator uses the categorized synthetic people as agents who take surveys hosted on the WiSDM application server, in turn generating synthetic responses. The survey details can be configured in the Config.json file. Currently, the data generated is for the 'USA Disease Prevalence October Fall 2015' survey.

The process flow diagram is depicted in Figure 5.2 and its implementation is described below. The class diagram is given in Figure 5.3.

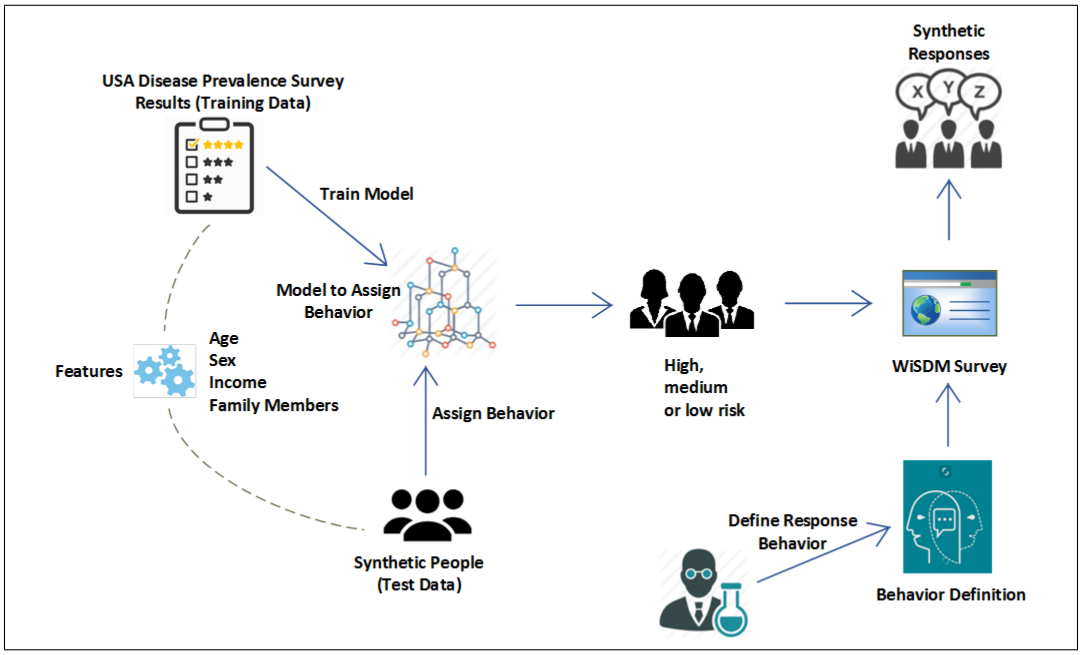


Figure 5.2: Synthetic Data Generation Process

1. Connect to the synthetic population Oracle database and pull the synthetic population data.

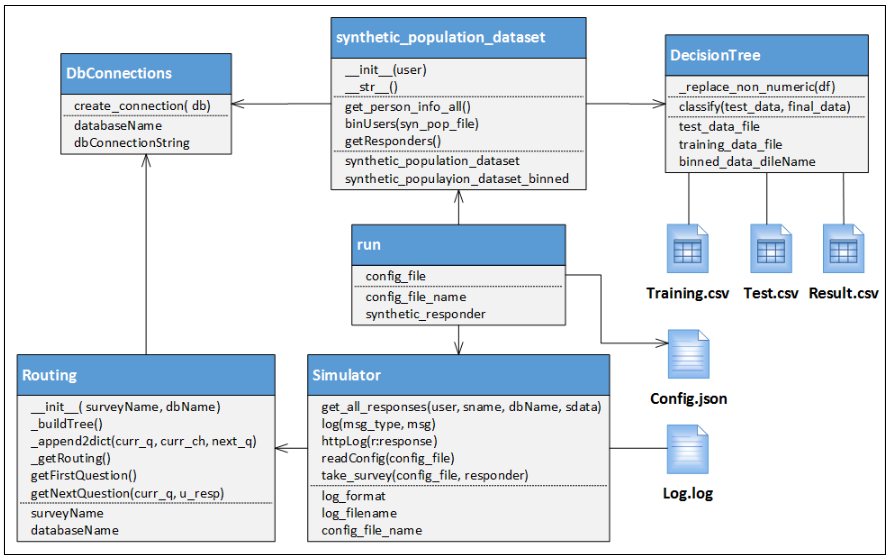


Figure 5.3: Synthetic Data Generator class diagram

(a) The synthetic people details are obtained for five states by joining <state>_PERSON_2009_1, <state>_PERSON_INFO_2009_1 and <state>_HOUSEHOLD_2009_1 tables.

2. The synthetic people collected have demographic information, including agep (age), sex (gender), hincp (household income), persons (number of family members), home_zipcode, HID (household id) and PID (person id).

   (a) Only the agep, sex, hincp, and persons values are added per synthetic person to the test data set in the classification model. This is because our test data set had only these demographic features.

3. Now the classification model, the Decision Tree classifier, is instantiated.

   (a) Since the decision tree implementation in scikit-learn uses only numerical features, the classifier converts categorical inputs into predefined integers. For example in sex, 'male' is set to 1 and 'female' is set to 2.

   (b) The missing values are replaced by the mean value to avoid the data set getting affected by missing values.

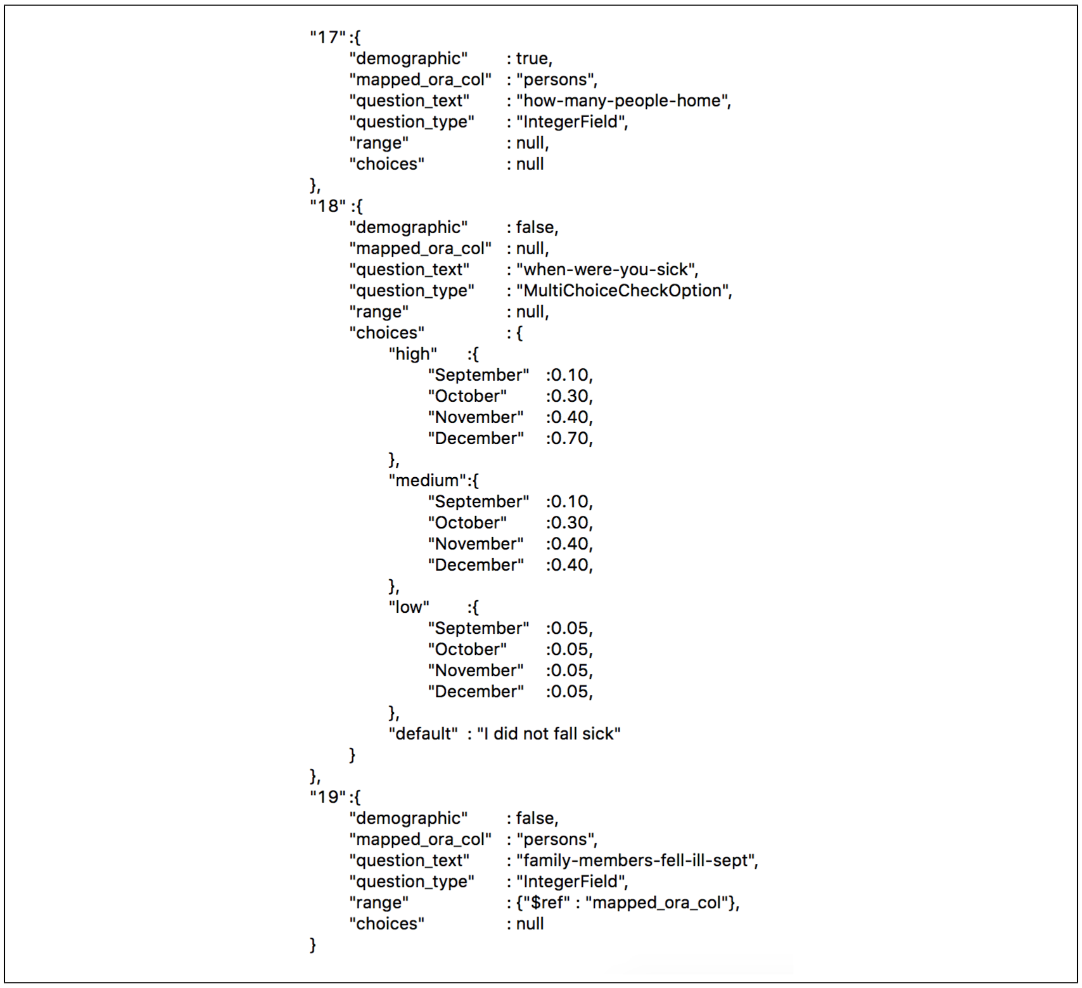
   (c) The data is also cross-validated before fitting the model.

4. When the test data is loaded, the categorical fields are first converted into integer labels, and each synthetic person is classified as 'high', 'low' or 'medium' risk.

5. Each bin is assigned some behavior. The behaviors are quantified as probabilities in the Config.json file. A sample is shown in Figure 5.4.

(a) For example in a “high risk” category, a question with choices ’yes’ and ’no’ may have a probability distribution of 90:10. The same question in the “medium risk” category could have a distribution of 50:50 and in “low risk” it could be 10:90.

(b) The probability distribution varies depending on the question types. For example, probabilities assigned to all options in a single choice question add up to 1. But probabilities assigned to each option in a multi-choices question add up to 1.



```
"17":{
      "demographic"      : true,
      "mapped_ora_col"   : "persons",
      "question_text"    : "how-many-people-home",
      "question_type"    : "IntegerField",
      "range"            : null,
      "choices"          : null
},
"18" :{
      "demographic"      : false,
      "mapped_ora_col"   : null,
      "question_text"    : "when-were-you-sick",
      "question_type"    : "MultiChoiceCheckOption",
      "range"            : null,
      "choices"          : {
            "high"    :{
                  "September"   :0.10,
                  "October"     :0.30,
                  "November"    :0.40,
                  "December"    :0.70,
            },
            "medium":{
                  "September"   :0.10,
                  "October"     :0.30,
                  "November"    :0.40,
                  "December"    :0.40,
            },
            "low"     :{
                  "September"   :0.05,
                  "October"     :0.05,
                  "November"    :0.05,
                  "December"    :0.05,
            },
            "default"  : "I did not fall sick"
      }
},
"19" :{
      "demographic"      : false,
      "mapped_ora_col"   : "persons",
      "question_text"    : "family-members-fell-ill-sept",
      "question_type"    : "IntegerField",
      "range"            : {"$ref" : "mapped_ora_col"},
      "choices"          : null
}
```

Figure 5.4: A sample of Config.json file

6. Now the Simulator fetches one person at a time and answers the demographic questions in the survey based on the synthetic person’s demographic information.

7. For behavioral questions, the simulator runs a biased randomizer based on the probabilities assigned to the person’s bin for each question.

   (a) The randomizer chooses one of all available options in the event of a single-select question.

   (b) For multi-select questions, the randomizer runs individually for each given option, either selecting or rejecting that option.
   (c) For questions with a range, the limit of which is determined by the responder's demographic information, the randomizer selects any value applicable to the user's demographic input as the upper limit.

8. Based on the survey branched logic and the person's response to the current question, the next question is chosen.
9. These responses are sent to the WiSDM application server as HTTP Responses via REST APIs.
   (a) The Simulator maintains a session for each responder so that the next request to be sent is correctly determined by WiSDM.
   (b) Each responder is assigned a unique session id.
10. All responses, along with the user's session id and PID, are saved to the database.

The pictorial representation of the decision tree model is shown in Figure 5.5. It can be noticed in Figure 5.5 that only two attributes, 'persons' and 'agep', are dominant. This is because, from the training data, the classifier found only those two features to be informative. The feature importance is calculated by Gini as illustrated below, and the corresponding plot is shown in Figure 5.6.

1. feature 0 (0.798120) : AGEP
2. feature 3 (0.201880) : PERSONS
3. feature 2 (0.000000) : HINCP
4. feature 1 (0.000000) : SEX

5.6 Additional Usages

The Synthetic Data Generator also has some additional applications:

- It can be used as a functional testing tool that tests the WiSDM application by sending a variety of accepted responses. These responses can be configured.
- It can be used as a tool for testing WiSDM Analytics. Since the data generated is not completely random, one can perform near-real analytics testing with huge amounts of data.

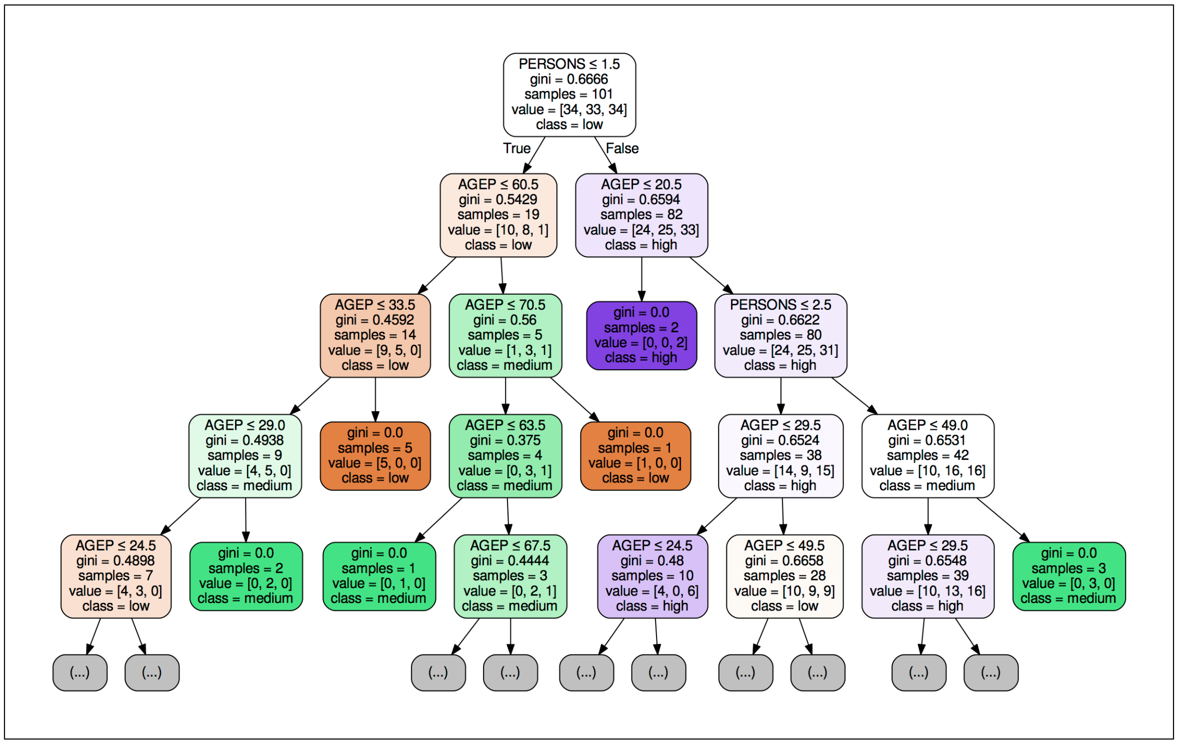


Figure 5.5: A pictorial representation of the trained tree

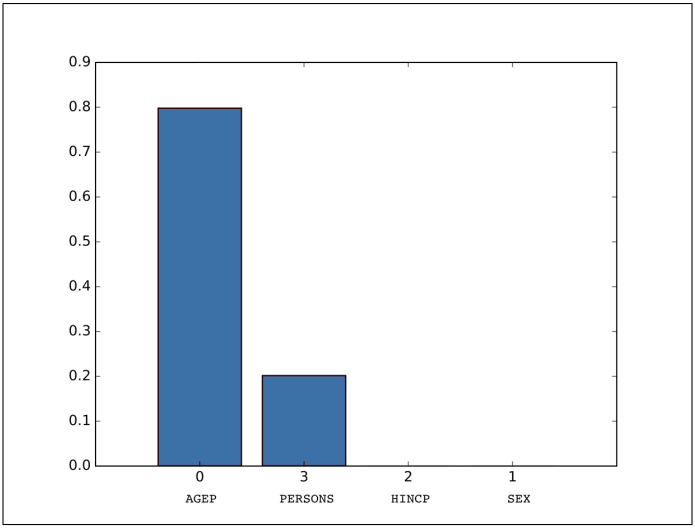


Figure 5.6: Feature Importance

5.7 Conclusion

In this chapter, we introduced a synthetic data generation module that simulates human behavior to answer survey questions related to a threat situation. Their behavior is defined by a classification model and a degree of uncertainty associated with each question in the survey. We built this module in order to generate near real data that can be used to evaluate inference models. We incorporated a flexible design approach keeping most of the components pluggable and configurable in order to maximize customizations for researchers. The Synthetic Data Generator is just the first step towards building a data generator. The Synthetic Population generated at NDSSL is very detailed and contains many features. Including a bigger subset of those features may overwhelm a decision tree, but a more sophisticated model like SVM[18] can leverage the available information stored in a bigger subset of features.

Chapter 6

WiSDM Analytics

Analyzing any survey, online or traditional, consists of a number of interrelated processes that are intended to summarize, arrange, and transform data into information. In order to get insight into the data collected through surveys, we must transform raw data into information in such a way that will allow us to draw patterns, build correlations, identify responders' behavior, and paint a clear picture for the researchers using our system.

Good visualization, a key component of analytics, means communicating clearly a message based on the results of the data analysis. But different messages may be most relevant for different audiences, so it is necessary to 1) think about who the audiences are, 2) work out how to format the data effectively for analyzing results or messages from the data set most relevant to that audience, and 3) to design visualizations which best communicate those particular messages based on the audience's roles and needs.

Using an example from WiSDM, results from the 'USA Disease Prevalence Fall 2015' survey would most likely be relevant to epidemiologists who are interested in flu transmission over multiple age ranges. This message will be more clearly communicated graphically rather than viewing the raw data in a tabular view.

6.1 Objective

Literature on survey results analysis is mostly limited to removing biases in the collected data and drawing correlations among the results using statistical models like ANOVA[69]. While we plan to incorporate such advanced analysis in WiSDM in the future, first we must build a suitable data model on which those analyses can be done. However, there are few articles that discuss how to format survey data for analysis.

In this chapter, we propose a data model that we think is best suited for analytics on survey data, and we also discuss how our survey data can be presented for visualization. This chapter is divided into two sections:

1. How can survey data be prepared so that data analysis is simple while still maximizing performance efficiency?
2. What visualization tools can make the most sense of our data, and how can visualizations be created around that output?

6.2 Data Modeling

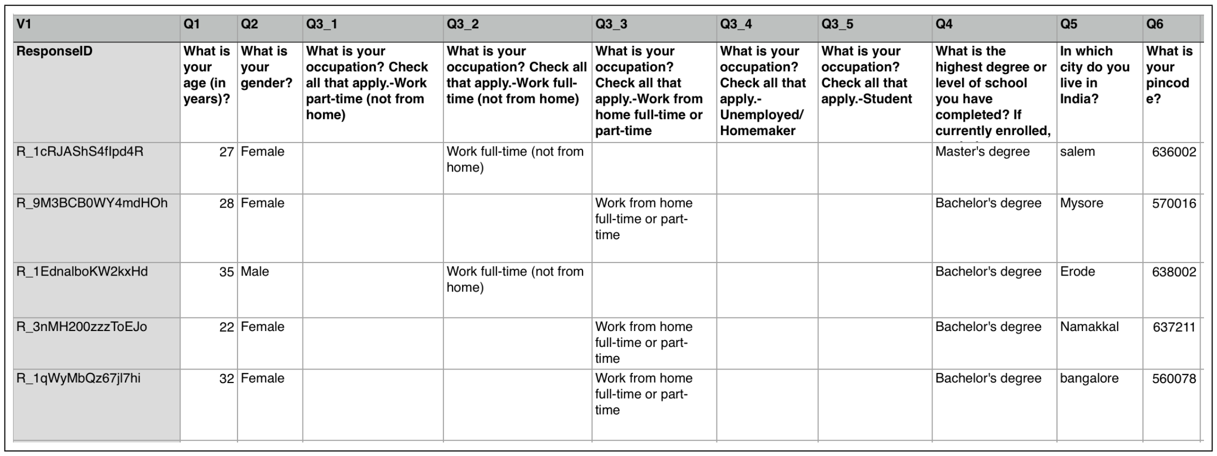
There is no single state-of-the-art or best practice in analytics around survey-oriented data. The biggest impediment is converting the source data into a format that is conducive to analytics. Before we discuss our proposed data model, let's look at some typical survey data formats.

6.2.1 Survey Data Format For Analysis

While most survey tools maintain data in a normalized form in their internal databases, they typically render data in a one-row-per-respondent format for analysis, with columns corresponding to questions in the survey. The complexity in analytics arises when surveys contain multiple multi-response questions. Multi-response questions can generally be organized into one of the following three formats for analysis.

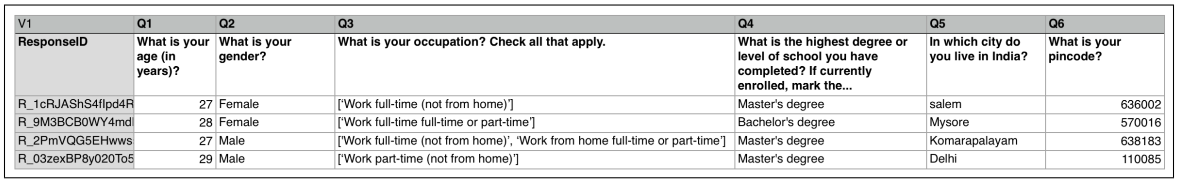
1. The most common format is to provide as many columns as there are question-choice combinations[79], a convention followed by Qualtrics (Figure 6.1) and many other commercially available survey tools. [115] also conducted a pen-and-paper longitudinal study for which analysis was performed on the data in the mentioned format.
2. The second method is to group all selected responses as a list. For example, column Q3 in Figure 6.2.
3. The third format is to maintain a separate table for the multi-response questions[79]. Each row in the table will have the person, question, and corresponding responses. So, there will be as many rows of data as there are responses, and there are no entries for the person who gives no response.

It is easier to do analytics when all responses for a question are stored in a single column, as compared to the format described in method 1, and it will also be more performance efficient. With the second method, additional computation needs to be performed to extract data from a list. The third format is better in terms of data organization, but in this approach, one would be required to know or identify question types (single response or multi-response types) before querying appropriate tables to answer analytical queries.



V1	Q1	Q2	Q3_1	Q3_2	Q3_3	Q3_4	Q3_5	Q4	Q5	Q6
ResponseID	What is your age (in years)?	What is your gender?	What is your occupation? Check all that apply.-Work part-time (not from home)	What is your occupation? Check all that apply.-Work full-time (not from home)	What is your occupation? Check all that apply.-Work from home full-time or part-time	What is your occupation? Check all that apply.-Unemployed/ Homemaker	What is your occupation? Check all that apply.-Student	What is the highest degree or level of school you have completed? If currently enrolled,	In which city do you live in India?	What is your pincode?
R_1cRJAShS4flpd4R	27	Female		Work full-time (not from home)				Master's degree	salem	636002
R_9M3BCB0WY4mdHOh	28	Female			Work from home full-time or part-time			Bachelor's degree	Mysore	570016
R_1EdnalboKW2kxHd	35	Male		Work full-time (not from home)				Bachelor's degree	Erode	638002
R_3nMH200zzzToEJo	22	Female			Work from home full-time or part-time			Bachelor's degree	Namakkal	637211
R_1qWyMbQz67jl7hi	32	Female			Work from home full-time or part-time			Bachelor's degree	bangalore	560078

Figure 6.1: Survey Data extracted from Qualtrics: The columns Q3_1 to Q3_5 represent the data that was gathered from the 'check all that apply' survey question 'What is your occupation?'.



V1	Q1	Q2	Q3	Q4	Q5	Q6
ResponseID	What is your age (in years)?	What is your gender?	What is your occupation? Check all that apply.	What is the highest degree or level of school you have completed? If currently enrolled, mark the...	In which city do you live in India?	What is your pincode?
R_1cRJAShS4flpd4R	27	Female	['Work full-time (not from home)']	Master's degree	salem	636002
R_9M3BCB0WY4mdl	28	Female	['Work full-time full-time or part-time']	Bachelor's degree	Mysore	570016
R_2PmVQG5EHwws	27	Male	['Work full-time (not from home)', 'Work from home full-time or part-time']	Master's degree	Komarapalayam	638183
R_03zexBP8y020To5	29	Male	['Work part-time (not from home)']	Master's degree	Delhi	110085

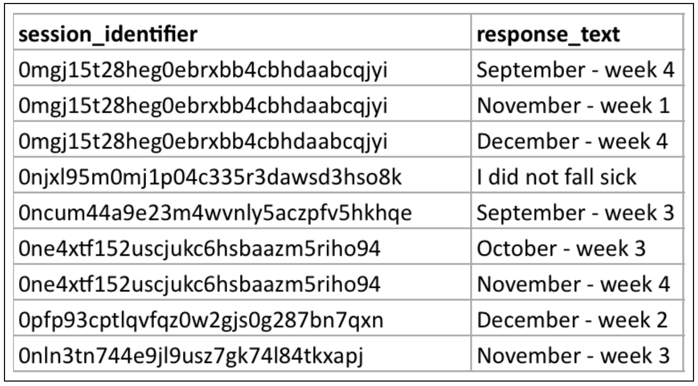
Figure 6.2: Example of data storage where multiple-select options are stored in a list.

6.2.2 WiSDM Analytics Data Model

Key-Value (KV) is a data management methodology designed for high read and write access patterns; this approach has the data stored in key and value format. Every key consist of an alphanumeric identifier and its associated values are stored in a simple string format. The values may be simple text or more complex lists and sets. Keys with no values are usually not stored in this data store; hence, a sparse data set is highly unlikely. As a simple data model, key-value stores are ideally suited to retrieve information in a very fast, available and scalable way. Amazon makes extensive use of its own Key-Value system, DynamoDB[101], in its shopping cart. DynamoDB is used in some of Amazon‘s core services to provide highly available and scalable distributed data stores.

For WiSDM, our vision was to build a data model that provides a simple and generic approach to doing analytics, while at the same time eliminating the creation of a sparse data set. The key-value data management system fits our needs perfectly. To the best of our knowledge, this model has never been used to structure survey results for analytics.

In this data model, survey data is split into multiple smaller tables. Each table represents one question and contains only two columns: <session_identifier> as a key and <response_text> as value. Session_identifier corresponds to a responder's session key, and response_text corresponds to responses for that question. So, if a survey is comprised of eight questions, there will be eight tables. In the case of multi-response questions, each response will be split into multiple rows in the table. An example of this format is displayed in Figure 6.3.



session_identifier	**response_text**
0mgj15t28heg0ebrxbb4cbhdaabcqjyi	September - week 4
0mgj15t28heg0ebrxbb4cbhdaabcqjyi	November - week 1
0mgj15t28heg0ebrxbb4cbhdaabcqjyi	December - week 4
0njxl95m0mj1p04c335r3dawsd3hso8k	I did not fall sick
0ncum44a9e23m4wvnly5aczpfv5hkhqe	September - week 3
0ne4xtf152uscjukc6hsbaazm5riho94	October - week 3
0ne4xtf152uscjukc6hsbaazm5riho94	November - week 4
0pfp93cptlqvfqz0w2gjs0g287bn7qxn	December - week 2
0nln3tn744e9jl9usz7gk74l84tkxapj	November - week 3

Figure 6.3: Example of a key-value format for WiSDM Analytics data model. Responses for question "When did you fall sick?" is shown in this table.

6.2.3 Advantages

This model has several advantages.

1. Data can be sliced-and-diced easily across any combination of criteria in order to answer analytical questions.
2. It allows for a survey-independent approach to data management.
3. It is easy to query on response texts, as all question types are organized in the same format and there is no separate format for multi-response questions.
4. Since each table will contain only a subset of the whole survey data, there is a significant reduction in the number of reads per transaction, which is essential as analytics is read-oriented.

6.3 Data Visualization

According to [40], data visualization is necessary to communicate information clearly and effectively through graphical means. It is a crucial step in data analysis because displaying data in a variety of visual formats makes it easy to see patterns and identify correlations among the result sets. In this section, we introduce Versa, a python framework for visualization, to do analytics on data collected through WiSDM.

6.3.1 Versa

Versa[51] is a data visualization and data analytics framework designed and developed at Network Dynamics and Simulation Science Laboratory (NDSSL) at Virginia Tech. It comes with APIs to perform data manipulation, analytics, and visualization operations. Using these APIs, users can run analytics as a series of python statements. But, unlike other python frameworks, Versa offloads computation to the database, and, hence, can scale to a data set much larger than the available main memory.

Each API call in Versa is converted into Relationally Mapped Objects (RMO), which are python data structures mapped to tables in the database. Subsequent API calls are merged into a single RMO. The RMO objects are then handed over to SQLAlchemy, which communicates with the underlying database. The RMO objects are *lazily executed,* implying that they get executed only when terminal APIs are called. Terminal APIs take an RMO instance but return non-RMO instances, such as lists.

Versa launches database instances on supercomputing nodes. When a job is launched, Versa reserves a compute resource and launches a database engine on a port. The compute node is identified as server IP address and the port as the server port. Computations are offloaded to the relational data engine, empowering the framework to handle data sets much larger than the main memory would allow. The details of a data engine connection are encapsulated in the API 'build_session', and all later interactions with the database are mediated through this handle. The Versa framework comprises of super computing nodes, a relational data engine, and SQLAlchemy, which is an object relational mapper (ORM).

SQLAlchemy[9] is an open source SQL toolkit and ORM for the python programming language. It is a high-level python interface which associates user-defined python classes to database tables. It includes both a database server-independent SQL expression language, and an Object Relational Mapper which allows the user to persist application objects automatically through SQL. It confines all data manipulation code to a specific database session that controls the life cycle of every object in that session. The SQLAlchemy Object Relational Mapper presents a method for associating user-defined python classes with database tables and instances of these classes with rows in their corresponding tables. It synchronizes all changes in state between objects and their related rows. It also has a system for expressing database queries in terms of these classes, along with their defined relationships between each other.

6.3.1.1 Related Work

Versa acts as a bridge between the SQLAlchemy ORM that communicates with the database and a python developer who wants to communicate with the multiple data sources and perform analytics on the data without writing SQL statements. Atlas[96], a data warehouse of biological data provides an easy storage and retrieval infrastructure for data from multiple sources using SQL queries. The data is first downloaded as data files from disparate data sources. Atlas loaders, which build instances of the relational model in the local database, loads these files into MySQL relational data models. Data can be accessed directly from the MySQL database, with SQL statements implemented underneath C++, Java and Perl APIs. Similarly, Versa allows developers to seamlessly interact with different data sources by calling python APIs, but, unlike Atlas, Versa is also equipped with APIs that enable analytics and visualizations on the data.

6.3.2 WiSDM With Versa

To analyze the data collected by WiSDM Data Acquisition system and to analyze the data generated by Synthetic Data Generator, we need an analytical tool that supports advanced analytical features to build sophisticated visualizations. WiSDM integrates with Versa to achieve this objective. WiSDM Analytics, an external module of WiSDM, imports Versa APIs to generate graphs on the data stored in WiSDM database. In this work, we built only a few basic charts but the approaches shown here can be extended to use more features of Versa. In near future, we plan to integrate this module with WiSDM Data Acquisition tool so that users can invoke Versa APIs and build visualizations from WiSDM admin interface.

In the current approach as an external module, we start out by creating a denormalized table, WISDM_SURVEY_DATA_ALL, with a limited set of columns (e.g. survey_name, session_identifier, question_name, question_text, response_text) that are most relevant for analytics; this table is created by combining Survey, Questions, Responses and Session tables from the WiSDM application database. This table can be populated on a routine basis.

In Versa, we first create an Oracle session to retrieve data from WISDM_SURVEY_DATA_ALL table. Using Versa APIs, we split the data as per our key-value data model and load it into a PostgreSQL database. This process, known as Pivoting, converts the single container data set into multiple, smaller, column-oriented (key-value) tables. The pseudocode for Pivoting is given in Algorithm 6.1.

The tables created in PostgreSQL persist as long as the life of the session which is created when a job to run a database instance is launched. This means that, for each survey, temporary tables are created only for the duration of the analysis being performed.

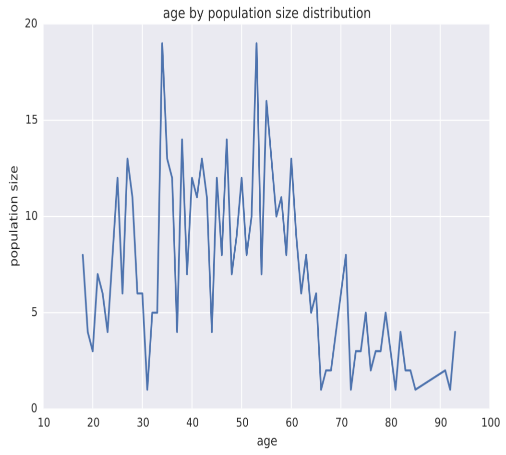
Versa APIs communicate with PostgreSQL tables and run user-defined standard data manipulation functions, such as joins, aggregation, filters, sort, etc. It also has APIs to build plots on the data. If some graphical plots are not supported in Versa, they can be easily integrated with other python modules to build those charts.

Below are the sets of analysis we did with Versa on two different surveys. The first survey was a data set of about 3000 responses with a respondent set of 500, and the second survey contains approximately 90,000 responses from a respondent set of 14,000. The code for these visualizations are included in Appendix D.

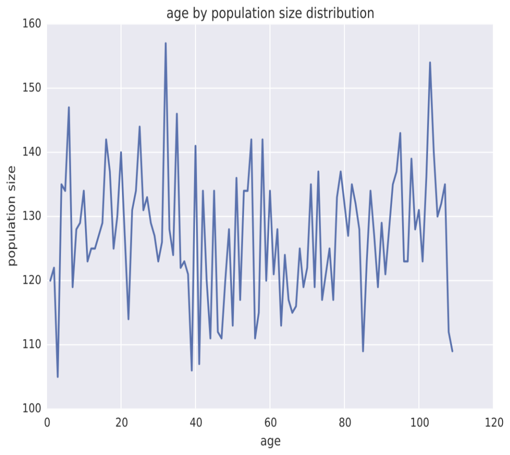
Algorithm 6.1 Pivoting

```
mport versa_api as vapi
import versa_impl as vi
from <source table>_model import *

# create a session with the source db
source_db_session = vi.init(<source db session>)

# create rmo on all values of the attribute in the source table
q_c = vapi.proj(source_db_session, <source table>, <attribute>)
# create rmo on all distinct values in q_c
d_q_c = vapi.distinct(source_db_session, q_c, <attribute>)
# retrieve distinct values in d_q_c as a list
d_res = vapi.scan(source_db_session, d_q_c)

# create a session with the target db
target_db_session = vi.init(<session name>)

# for each item dim in d_res
for dim in d_res:
    # filter source table when attribute=dim[0]
    s1 = vapi.filterEQ(source_db_session, <source table>, <attribute>, dim[0])
    # attribute value becomes the table name in target db
    tbl_name = str(dim[0])
    # define distinct column names in the target table
    target_col_names = []
    for each attr in <distinct attributes in target table>:
            target_col_names.append(attr)

    # define table with the table name and column names
    [table, cls, rmo] = vapi.create_table_and_rmo(
                        target_db_session, table = tbl_name,
                        cls = tbl_name, attrs=target_col_names,
                        types= [<datatype of each column>],
                        primary_keys= [<set primary key for columns as True/False>],
                        indexes = [<set columns to be indexed as True/False>]
                        )

    # create target table with name as
    tbl_name vapi.write_rmo_def(target_db_session, rmo, tbl_name, tbl_name, tbl_name)
    # retrieve data from s1 as a list
    d_s1 = vapi.scan(source_db_session, s1)
    # for each data row s in d_s1
    for s in d_s1:
        # instantiate target table class
        newTable = cls()
        # insert respective values from s into newTable's attributes.
        for i in range(target_col_names):
            setattr(newTable, target_col_names[i], s[0])
            target_db_session.add(newTable)

    target_db_session.commit()
```

Figure 6.4 represents the age distribution for a respondent size of 500 and an approximate population size of 14,000 for the second survey.



(a) Respondent Size 500



(b) Respondent Size 14,000

Figure 6.4: Age by Population Size Distribution

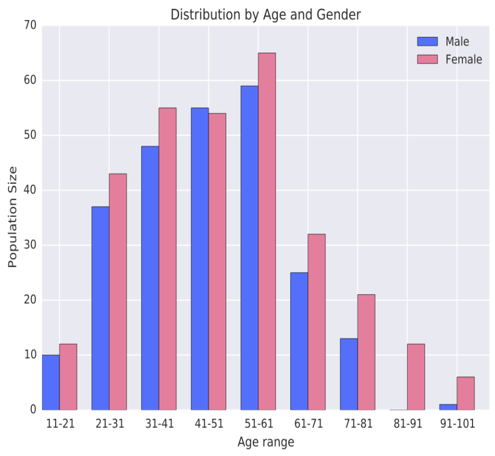
Figure 6.5 represents a grouped bar chart of age distribution by gender.

In addition, Figure 6.6 shows a bar chart of the age distribution by sick population for the first survey.

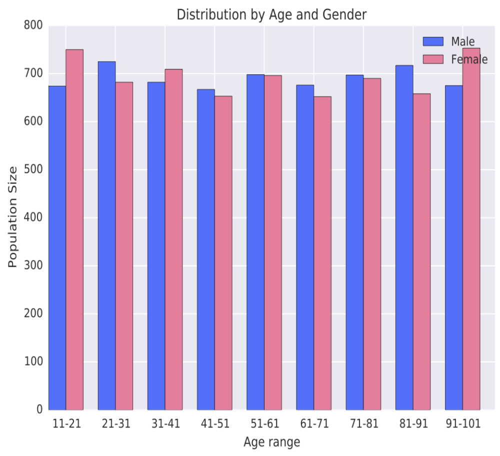
6.3.3 Performance Comparison Of Versa Between Oracle And PostgreSQL

We did a performance comparison between Oracle and PostgreSQL by running Versa on both the databases. We tested on a data set of 1,171,389 rows in wisdmdev instance in Oracle. This data set was created by running a script to import random responses for the 'USA Disease Prevalence Fall 2015' survey hosted on WiSDM directly into the Oracle database.

1. First, we performed Pivoting, where we split the data set into multiple smaller tables. Time taken by the two databases:
   (a) PostgreSQL: 0:05:41.215573
   (b) Oracle: 0:03:53.845529
2. Next, we performed an analysis of gender by age distribution in the data set. Two tables were joined here – 'age-in-years' : 112,488 rows, and 'your-gender' : 112,381 rows. Time taken by the two databases.



(a) Respondent Size 500



(b) Respondent Size 14,000

Figure 6.5: Age by Gender Distribution

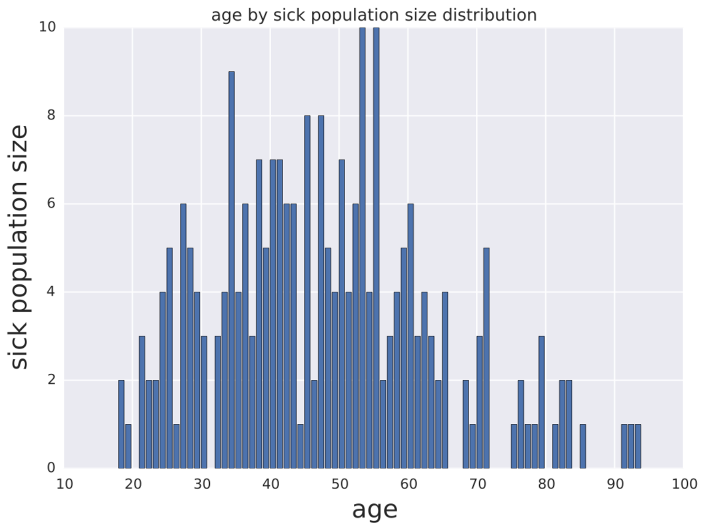


Figure 6.6: Age by Sick Population Size Distribution

(a) PostgreSQL: 0:00:00.583687

(b) Oracle: 0:00:01.864170

6.3.4 Advantages Of Versa

There are many advantages to using Versa.

1. Versa offloads computation to supercomputing data nodes. So it can handle data sets much larger than the main memory available.
2. Versa allows users to perform database operations like join, filter, aggregation, etc. as python function calls. This simplifies the database interactions for a developer not proficient in SQL.
3. Versa can seamlessly interact with multiple databases. We communicated with the Oracle database and PostgreSQL while doing analytics with the WiSDM data set.
4. The biggest advantage of Versa is that it is an in-house resource and can be enhanced as needed.

6.4 Conclusion

In this chapter, we introduced a data model based on key-value concepts. This data model is generic and provides an intuitive structure to run analytical queries. We also discussed how Versa can be used to build the key-value data model and showed a few examples of visualizations performed on the data, using Versa APIs.

Chapter 7

Conclusions And Directions For Future Work

Online data acquisition platforms have emerged over the last few years as highly convenient tools for conducting research. These platforms enable researchers to create and deliver surveys to subjects/participants in a convenient manner, and they produce results rapidly. Many researchers use them to conduct online studies, analyze results, re-evaluate their hypothesis and sometimes conduct the study again. But most of these tools do not support seamless integration to crowdsourced platforms that are cost and time effective resources to acquire research data. These tools also do not provide simple interfaces to build intelligent surveys. These were the primary motivations behind developing WiSDM. WiSDM is an advanced data acquisition system that uses human sensors to acquire behavioral information. It can build intelligent surveys that can adapt according to responders' behavior and can be integrated with crowdsourcing platforms like MTurk through iFrames for faster situation awareness. It can also integrate external applications in iFrames in its web template to increase interactivity with the responder.

However, we did not limit WiSDM to data acquisition. WiSDM can also operate as an ETL tool (known as the WiSDM API Adapter), which is a novel way to leverage off of other systems that may support special features that lie outside of WiSDM's capabilities. We also presented the Synthetic Data Generator as an external module of WiSDM that can generate large-scale synthetic responses by generating survey results by using a synthetic population in conjunction with a classification model to simulate near-real data for testing inference algorithms. We also showed how survey results can be formatted so that analytics on the data is simple. We proposed a (key, value) data management system that can be used to model survey data. It requires splitting the entire data set into multiple, smaller, non-sparse tables. Finally, we showed how we could do analytics on WiSDM data using Versa, a python-based framework for data analytics and visualization.

With all of these extensions, WiSDM comes full circle. However, WiSDM is still under active development, and there are a number of additional features that would further enhance its value:

- Adding more advanced logic to surveys, such as an ability to add branched logic on multiple variables and not just the previous question. For example, if responses to question #1 is X and question #2 is Y, then jump to question #5.
- Adding simple data validation techniques, such as geocode detection, screeners, minimum response duration per user, etc. may partially filter out untruthful data, detect inattentive respondents, and mitigate survey speeding issues.
- Gamification of surveys to improve data quality[55, 31, 54]. Although gamification of online surveys does not guarantee significant positive results, it does increase user engagement. Not all types of surveys can be gamified, but they can definitely have features like badges and stages that may increase user motivation.
- Building native mobile apps for WiSDM to leverage on mobile crowdsensing. Although WiSDM in its current implementation can be accessed as a mobile browser app, a native app provides many additional advantages apart from ease of use such as providing push notification to remind a participant to input information. Native apps can also implicitly collect user data such as user's current location and help detect context against which a relevant question can be asked[41].
- Tighter integration of the API Adapter to the WiSDM web interface. It would be better if a user interface could be built around the API Adapter and also provide the ability to schedule data transfers. A UI wrapper will help users easily specify parameters and constraints required to extract data from external data sources.
- Development of a more sophisticated classification model for the Synthetic Data Generator. A sophisticated model such as SVM (Support Vector Machine)[18] can leverage the information available, provided the training data to support those features. Using a data processing technique like Principal Component Analysis (PCA) with classification may also improve classification accuracy when dealing with high dimensional data[60].
- Parallelizing Synthetic Data Generator so that multiple virtual agents can take the survey simultaneously. This will improve the Synthetic Data Generator as a functional/load testing tool.
- Integrating WiSDM Analytics into the WiSDM web application. Developing a web form where users can enter question names, filter conditions, and chart specifications to represent data in a graphical format would make this system more accessible to researchers. The web form will internally call Versa APIs to build visualizations. It would also allow users without any programming background to use WiSDM Analytics.

WiSDM, as a data acquisition system that also supports data simulation, data analytics, and ETL, is a novel idea which is generic enough to support research outside of epidemiology. It can be integrated with surveillance systems such as Eyes on the Ground to collect ground information for better situational awareness. WiSDM is designed and developed at NDSSL and is a result of collaborative efforts of various members of the NDSSL lab.

Bibliography

[1] Geochat. Innovative Support to Emergencies Diseases and Disasters (InSTEDD). [Online; accessed 20-June-2016].

[2] Vivek Akupatni, Richard J. Beckman, Keith R. Bisset, Jiangzhuo Chen, Bryan L. Lewis, Achla Marathe, Madhav V. Marathe, Vullikanti S. Anil Kumar, and Amanda Wilson. Widely internet sourced distributed monitoring. Technical report, NDSSL, 2015.

[3] Atocha Aliseda-Llera. Seeking explanations: Abduction in logic, philosophy of science and arti cal intelligence. 1997.

[4] Dorine Andrews, Blair Nonnecke, and Jennifer Preece. Electronic survey methodology: A case study in reaching hard-to-involve internet users. *International journal of human-computer interaction*, 16(2):185–210, 2003.

[5] Dorine Andrews, Blair Nonnecke, and Jennifer Preece. Conducting research on the internet: Online survey design, development and implementation guidelines. 2007.

[6] Anuja Arora and Madhavi Sinha. Web Application Testing: A Review on Techniques, Tools and State of Art. *International Journal of Scientific & Engineering Research*, 3(2):1, 2012.

[7] Scott Barber. How fast does a website need to be, 2010.

[8] Adam Barth, Collin Jackson, and John C Mitchell. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 75–88. ACM, 2008.

[9] Michael Bayer. Sqlalchemy - the database toolkit for python. *http://www.sqlalchemy.org/*, 2012.

[10] Keith R. Bisset, Karla Atkins, Christopher L. Barrett, Richard J. Beckman, Stephen Eubank, Achla Marathe, Madhav V. Marathe, Henning Mortveit, Paula Stretz, and Anil Vullikanti. Synthetic Data Products for Societal Infrastructures and Proto-Populations: Data Set 2.0. Technical report, NDSSL Technical Report, 2006.

[11] Joaquin A Blaya, Hamish SF Fraser, and Brian Holt. E-health technologies show promise in developing countries. *Health Affairs*, 29(2):244–251, 2010.

[12] Michael Buhrmester, Tracy Kwang, and Samuel D Gosling. Amazon's Mechanical Turk a new source of inexpensive, yet high-quality, data? *Perspectives on psychological science*, 6(1):3–5, 2011.

[13] Jared Cechanowicz, Carl Gutwin, Briana Brownell, and Larry Goodfellow. Effects of gamification on participation and data quality in a real-world market research domain. In *Proceedings of the First International Conference on Gameful Design, Research, and Applications*, pages 58–65. ACM, 2013.

[14] Andre Charland and Brian Leroux. Mobile application development: web vs. native. *Communications of the ACM*, 54(5):49–53, 2011.

[15] Georgios Chatzimilioudis, Andreas Konstantinidis, Christos Laoudias, and Demetrios Zeinalipour-Yazti. Crowdsourcing with smartphones. *IEEE Internet Computing*, 16(5):36–44, 2012.

[16] Timothy Chee, Lee-Kwun Chan, Min-Hooi Chuah, Chee-Sok Tan, Siew-Fan Wong, and William Yeoh. Business intelligence systems: state-of-the-art review and contemporary applications. In *Symposium on Progress in Information & Communication Technology*, volume 2, pages 16–30, 2009.

[17] Rumi Chunara, Clark C Freifeld, and John S Brownstein. New technologies for reporting real-time emergent infections. *Parasitology*, 139(14):1843–1851, 2012.

[18] Corinna Cortes and Vladimir Vapnik. Support vector machine. *Machine learning*, 20(3):273–297, 1995.

[19] Mick P Couper. Review: Web surveys: A review of issues and approaches. *The Public Opinion Quarterly*, 64(4):464–494, 2000.

[20] Mick P Couper and Peter V Miller. Web survey methods introduction. *Public Opinion Quarterly*, 72(5):831–835, 2008.

[21] Mick P Couper, Michael W Traugott, and Mark J Lamias. Web survey design and administration. *Public opinion quarterly*, 65(2):230–253, 2001.

[22] Dana Cuff, Mark Hansen, and Jerry Kang. Urban sensing: out of the woods. *Communications of the ACM*, 51(3):24–33, 2008.

[23] Edith D De Leeuw. To mix or not to mix data collection modes in surveys. *Journal of official statistics*, 21(2):233, 2005.

[24] Edith D De Leeuw and Joop J Hox. Internet surveys as part of a mixed-mode design. *Social and Behavioral research and the internet*, pages 45–76, 2011.

[25] Anind K Dey, Gregory D Abowd, and Daniel Salber. A context-based infrastructure for smart environments. In *Managing Interactions in Smart Environments*, pages 114–128. Springer, 2000.

[26] Tim Dierks. The transport layer security (tls) protocol version 1.2. 2008.

[27] Don A Dillman. Mail and internet surveys: The tailored design method, volume 2. Wiley New York, 2000.

[28] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Ontology matching: A machine learning approach. In *Handbook on ontologies*, pages 385–403. Springer, 2004.

[29] Sara Dolnicar, Bettina Grün, and Venkata Yanamandram. Dynamic, interactive survey questions can increase survey data quality. *Journal of Travel & Tourism Marketing*, 30(7):690–699, 2013.

[30] Steven Dow, Anand Kulkarni, Brie Bunge, Truc Nguyen, Scott Klemmer, and Björn Hartmann. Shepherding the crowd: managing and providing feedback to crowd workers. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*, pages 1669–1674. ACM, 2011.

[31] Theo Downes-Le Guin, Reg Baker, Joanne Mechling, and Erica Ruyle. Myths and realities of respondent engagement in online surveys. *International Journal of Market Research*, 54(5):1–21, 2012.

[32] Jörg Drechsler and Jerome P Reiter. An empirical evaluation of easily implemented, nonparametric methods for generating synthetic datasets. *Computational Statistics & Data Analysis*, 55(12):3232–3243, 2011.

[33] Bobby Duffy, Kate Smith, George Terhanian, and John Bremer. Comparing data from online and face-to-face surveys. *International Journal of Market Research*, 47(6):615, 2005.

[34] Thomas Eiter and Georg Gottlob. The complexity of logic-based abduction. *Journal of the ACM (JACM)*, 42(1):3–42, 1995.

[35] Usama M Fayyad and Keki B Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine learning*, 8(1):87–102, 1992.

[36] Ernst Fehr and Simon Gächter. Cooperation and punishment in public goods experiments. *Institute for Empirical Research in Economics working paper*, (10), 1999.

[37] Robert J Fisher. Social desirability bias and the validity of indirect questioning. *Journal of consumer research*, pages 303–315, 1993.

[38] Harry G Frankfurt. Peirce's notion of abduction. *The Journal of Philosophy*, 55(14):593–597, 1958.

[39] Clark C Freifeld, Rumi Chunara, Sumiko R Mekaru, Emily H Chan, Taha Kass-Hout, Anahi Ayala Iacucci, and John S Brownstein. Participatory epidemiology: use of mobile phones for community-based health reporting. *PLoS Med*, 7(12):e1000376, 2010.

[40] Vitaly Friedman. Data Visualization and Infographics in Graphics. *Monday inspiration*, 2008.

[41] Jon Froehlich. Sensing and Feedback of Everyday Activities to Promote Environmentally Sustainable Behaviors. In *Doctoral Colloquium in the Adjunct Proceedings of UbiComp, Orlando, Florida, USA*, 2009.

[42] Jon Froehlich, Mike Y Chen, Sunny Consolvo, Beverly Harrison, and James A Landay. Myexperience: a system for in situ tracing and capturing of user feedback on mobile phones. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 57–70. ACM, 2007.

[43] Jon Froehlich, Joachim Neumann, and Nuria Oliver. Sensing and Predicting the Pulse of the City through Shared Bicycling. In *IJCAI*, volume 9, pages 1420–1426, 2009.

[44] Lutz Frommberger and Falko Schmid. Mobile4d: crowdsourced disaster alerting and reporting. In *Proceedings of the Sixth International Conference on Information and Communications Technologies and Development: Notes-Volume 2*, pages 29–32. ACM, 2013.

[45] Raghu K Ganti, Fan Ye, and Hui Lei. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine*, 49(11):32–39, 2011.

[46] Huiji Gao, Geoffrey Barbier, Rebecca Goolsby, and Daniel Zeng. Harnessing the crowdsourcing power of social media for disaster relief. Technical report, DTIC Document, 2011.

[47] Joshua Glasser and Brian Lindauer. Bridging the gap: A pragmatic approach to generating insider threat data. In *Security and Privacy Workshops (SPW), 2013 IEEE*, pages 98–104. IEEE, 2013.

[48] Joseph K Goodman, Cynthia E Cryder, and Amar Cheema. Data collection in a flat world: The strengths and weaknesses of Mechanical Turk samples. *Journal of Behavioral Decision Making*, 26(3):213–224, 2013.

[49] Samuel D Gosling, Simine Vazire, Sanjay Srivastava, and Oliver P John. Should we trust web-based studies? A comparative analysis of six preconceptions about internet questionnaires. *American Psychologist*, 59(2):93, 2004.

[50] Bin Guo, Zhiwen Yu, Xingshe Zhou, and Daqing Zhang. From participatory sensing to mobile crowd sensing. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pages 593–598. IEEE, 2014.

[51] Sandeep Gupta. Dicex User Manual and Cookbook, 2016.

[52] Peter Haddawy, Lutz Frommberger, Tomi Kauppinen, Giorgio De Felice, Prae Charkratpahu, Sirawaratt Saengpao, and Phanumas Kanchanakitsakul. Situation awareness in crowdsensing for disease surveillance in crisis situations. In *Proceedings of the Seventh International Conference on Information and Communication Technologies and Development*, page 38. ACM, 2015.

[53] Emily H Halili. Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites. Packt Publishing Ltd, 2008.

[54] Juho Hamari, Jonna Koivisto, and Harri Sarsa. Does gamification work? – A literature review of empirical studies on gamification. In *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, pages 3025–3034. IEEE, 2014.

[55] Johannes Harms, Stefan Biegler, Christoph Wimmer, Karin Kappel, and Thomas Grechenig. Gamification of online surveys: Design process, case study, and evaluation. In *Human-Computer Interaction–Interact 2015*, pages 219–236. Springer, 2015.

[56] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, and Paul Webster. The anatomy of a context-aware application. *Wireless Networks*, 8(2/3):187–197, 2002.

[57] Robert M. Hauser. Investments in Longitudinal Surveys, Databases, Advanced Statistical Research, and Computation Technology. *http://www.ncbi.nlm.nih.gov/books/NBK220826/*, 2002.

[58] Adrian Holovaty and Jacob Kaplan-Moss. The django book. *http://www.djangobook.com*, 2007.

[59] Adrian Holovaty and Jacob Kaplan-Moss. The definitive guide to Django: Web development done right. Apress, 2009.

[60] Tom Howley, Michael G Madden, Marie-Louise O'Connell, and Alan G Ryder. The effect of principal component analysis on machine learning accuracy with high-dimensional spectral data. *Knowledge-Based Systems*, 19(5):363–370, 2006.

[61] Janet Ilieva, Steve Baron, and Nigel M Healey. Online surveys in marketing research: Pros and cons. *International Journal of Market Research*, 44(3):361, 2002.

[62] Ron S Jarmin, Thomas A Louis, and Javier Miranda. Expanding the role of synthetic data at the US Census Bureau. *Statistical Journal of the IAOS*, 30(2):117–121, 2014.

[63] Apache JMeter. Apache software foundation, 2010.

[64] Ashish Kapoor, Nathan Eagle, and Eric Horvitz. People, Quakes, and Communications: Inferences from Call Dynamics about a Seismic Event and its Influences on a Population. In *AAAI Spring Symposium: Artificial Intelligence for Development*, 2010.

[65] Mike Keith and Merrick Schnicariol. Object-relational mapping. In *Pro JPA 2*, pages 69–106. Springer, 2009.

[66] Trace S Kershaw, Jessica Lewis, Claire Westdahl, Yun F Wang, Sharon Schindler Rising, Zohar Massey, and Jeannette Ickovics. Using clinical classification trees to identify individuals at risk of STDs during pregnancy. *Perspectives on sexual and reproductive health*, 39(3):141–148, 2007.

[67] Bogdan Korel. Automated software test data generation. *Software Engineering, IEEE Transactions on*, 16(8):870–879, 1990.

[68] Robert Kraut, Judith Olson, Mahzarin Banaji, Amy Bruckman, Jeffrey Cohen, and Mick Couper. Psychological research online: report of Board of Scientific Affairs' Advisory Group on the Conduct of Research on the Internet. *American psychologist*, 59(2):105, 2004.

[69] Sameer Kumar and Promma Phrommathed. *Research methodology*. Springer, 2005.

[70] Florian Laws, Christian Scheible, and Hinrich Schütze. Active learning with amazon mechanical turk. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1546–1556. Association for Computational Linguistics, 2011.

[71] Avraham Leff and James T Rayfield. Web-application development using the model/view/controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*, pages 118–127. IEEE, 2001.

[72] Yaliang Li, Jing Gao, Chuishi Meng, Qi Li, Lu Su, Bo Zhao, Wei Fan, and Jiawei Han. A Survey on Truth Discovery. *arXiv preprint arXiv:1505.02463*, 2015.

[73] Ulf Liebe, Klaus Glenk, Malte Oehlmann, and Jürgen Meyerhoff. Does the use of mobile devices (tablets and smartphones) affect survey quality and choice behaviour in web surveys? *Journal of choice modelling*, 14:17–31, 2015.

[74] Pengyue J Lin, Behrokh Samadi, Alan Cipolone, Daniel R Jeske, Sean Cox, Carlos Rendon, Douglas Holt, and Rui Xiao. Development of a synthetic data set generator for building and testing information discovery systems. In *Information Technology: New Generations, 2006. ITNG 2006. Third International Conference on*, pages 707–712. IEEE, 2006.

[75] Greg Little, Lydia B Chilton, Max Goldman, and Robert C Miller. Turkit: tools for iterative tasks on Mechanical Turk. In *Proceedings of the ACM SIGKDD workshop on human computation*, pages 29–30. ACM, 2009.

[76] Winter Mason and Siddharth Suri. Conducting behavioral research on Amazon Mechanical Turk. *Behavior research methods*, 44(1):1–23, 2012.

[77] Sheila A McIlraith. Logic-based abductive inference. *Knowledge Systems Laboratory, Technical Report KSL-98-19*, 1998.

[78] Kent L Norman. Implementation of conditional branching in computerized self-Administered questionnaires. 2003.

[79] The University of Reading Statistical Services Centre. Approaches to the Analysis of Survey Data. *Biometrics Advisory and Support Service to DFID*, 2001.

[80] Daniel M Oppenheimer, Tom Meyvis, and Nicolas Davidenko. Instructional manipulation checks: Detecting satisficing to increase statistical power. *Journal of Experimental Social Psychology*, 45(4):867–872, 2009.

[81] Robin Wentao Ouyang, Lance Kaplan, Paul Martin, Alice Toniolo, Mani Srivastava, and Timothy J Norman. Debiasing crowdsourced quantitative characteristics in local businesses and services. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, pages 190–201. ACM, 2015.

[82] Gabriele Paolacci, Jesse Chandler, and Panagiotis G Ipeirotis. Running experiments on Amazon Mechanical Turk. *Judgment and Decision making*, 5(5):411–419, 2010.

[83] David C Parkes, Andrew Mao, Yiling Chen, Krzysztof Z Gajos, Ariel Procaccia, and Haoqi Zhang. Turkserver: Enabling synchronous and longitudinal online experiments. In *Proceedings of the Fourth Workshop on Human Computation (HCOMP'12)*. AAAI Press, 2012.

[84] Bhoomit Patel, Jay Parikh, and Rushabh Shah. A Review Paper on Comparison of SQL Performance Analyzer Tools: Apache JMeter and HP LoadRunner. 2014.

[85] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[86] Yun Peng and James A. Reggia. Abductive Inference Methods for Diagnostic Problem-Solving, 1990.

[87] Andy Peytchev, Mick P Couper, Sean Esteban McCabe, and Scott D Crawford. Web survey design paging versus scrolling. *Public opinion quarterly*, 70(4):596–607, 2006.

[88] James E Pitkow and Colleen M Kehoe. Emerging trends in the WWW user population. *Communications of the ACM*, 39(6):106–108, 1996.

[89] Question Pro. Survey Software. *online, http://www.questionpro.com*, 2005.

[90] Laura Elena Raileanu and Kilian Stoffel. Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41(1):77–93, 2004.

[91] Ulf-Dietrich Reips. The Web experiment method: Advantages, disadvantages, and solutions. *Psychological experiments on the Internet*, pages 89–117, 2000.

[92] Ulf-Dietrich Reips. Standards for Internet-based experimenting. *Experimental psychology*, 49(4):243, 2002.

[93] Jerome P Reiter. Using CART to generate partially synthetic public use microdata. *Journal of official statistics-stockholm*, 21(3):441, 2005.

[94] Lior Rokach and Oded Maimon. Decision trees. In *Data mining and knowledge discovery handbook*, pages 165–192. Springer, 2005.

[95] Kirusnapillai Selvarajah, Michael P Craven, Adam Massey, John Crowe, Kavita Vedhara, and Nicholas Raine-Fenning. Native Apps versus Web Apps: which is best for healthcare applications? In *International Conference on Human-Computer Interaction*, pages 189–196. Springer, 2013.

[96] Sohrab P Shah, Yong Huang, Tao Xu, Macaire MS Yuen, John Ling, and BF Francis Ouellette. Atlas–a data warehouse for integrative bioinformatics. *BMC bioinformatics*, 6(1):1, 2005.

[97] Joshua Wolf Shenk. What makes us happy? *The Atlantic*, 2009.

[98] Amit Sheth. Citizen sensing, social signals, and enriching human experience. *IEEE Internet Computing*, 13(4):87, 2009.

[99] Sickweather. Sickweather. [Online; accessed 20-June-2016].

[100] Robert Simpson, Kevin R Page, and David De Roure. Zooniverse: observing the world's largest citizen science platform. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 1049–1054. ACM, 2014.

[101] Swaminathan Sivasubramanian. Amazon dynamoDB: a seamlessly scalable non-relational database service. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 729–730. ACM, 2012.

[102] Christine B Smith. Casting the net: Surveying an Internet population. *Journal of Computer-Mediated Communication*, 3(1):0–0, 1997.

[103] Jon Snow and Myntillae Nash. Qualtrics survey software: handbook for research professionals, 2013.

[104] Siddharth Suri and Duncan J Watts. Cooperation and contagion in web-based, networked public goods experiments. *PLoS One*, 6(3):e16836, 2011.

[105] Melanie Swan. Crowdsourced Health Research Studies: An Important Emerging Complement to Clinical Trials in the Public Health Research Ecosystem. *Journal of Medical Internet Research*, 14(2):e46, 2012.

[106] Natasha L Tilston, Ken TD Eames, Daniela Paolotti, Toby Ealden, and John Edmunds. Internet-based surveillance of Influenza-like-illness in the UK during the 2009 H1N1 influenza pandemic. *BMC Public Health*, 10(1):650, 2010.

[107] Vera Toepoel, Marcel Das, and Arthur van Soest. Design of Web questionnaires: A test for number of items per screen. 2005.

[108] Hana Tomaskova, Hana Slachtova, and Anna Splichalova. Using methods of decision tree for identification of target population groups for health intervention: ISEE-176. *Epidemiology*, 14(5):S37–S38, 2003.

[109] Jos WR Twisk. Applied longitudinal data analysis for epidemiology: A practical guide. Cambridge University Press, 2013.

[110] Tamás Varga and Horst Bunke. Generation of synthetic training data for an HMM-based handwriting recognition system. In *ICDAR*, volume 3, pages 618–622. Citeseer, 2003.

[111] Panos Vassiliadis, Alkis Simitsis, and Spiros Skiadopoulos. Conceptual modeling for ETL processes. In *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP*, pages 14–21. ACM, 2002.

[112] Lee Ventola. Mobile devices and apps for health care professionals: uses and benefits. *Pharmacy and Therapeutics*, 39(5):356, 2014.

[113] Pew Research Center Washington D.C. A Comparison of Results from Surveys by the Pew Research Center and Google Consumer Surveys, 2012. [Online; accessed 16-July-2016].

[114] Pew Research Center Washington D.C. Internet Surveys, 2014. [Online; accessed 16-July-2016].

[115] Victoria M White and Emily Bariola. Australian Secondary School Students' Use of Tobacco, Alcohol, and Over-the-counter and Illicit Substances in 2011: Report. National Drug Strategy, Department of Health and Ageing, 2012.

[116] Mark A Whiting, Jereme Haack, and Carrie Varley. Creating realistic, scenario-based synthetic data for test and evaluation of information analytics software. In *Proceedings of the 2008 Workshop on BEyond time and errors: novel evaLuation methods for Information Visualization*, page 8. ACM, 2008.

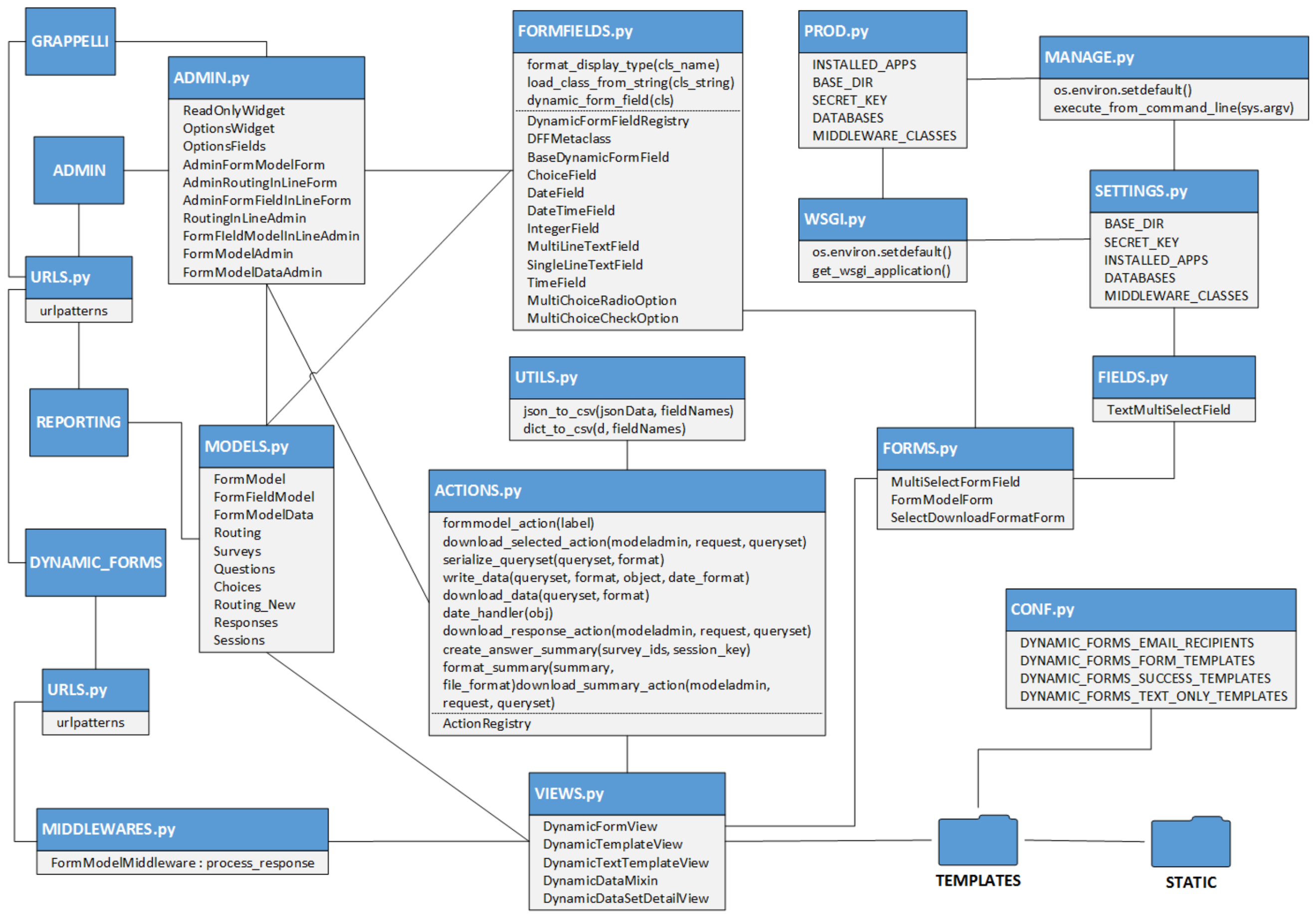
[117] Kevin B Wright. Researching internet-based populations: Advantages and disadvantages of online survey research, online questionnaire authoring software packages, and web survey services. *Journal of Computer-Mediated Communication*, 10(3):00–00, 2005.

[118] Gi Woong Yun and Craig W Trumbo. Comparative response to a survey executed by post, e-mail, & web form. *Journal of Computer-Mediated Communication*, 6(1):0–0, 2000.

[119] Arkady Zaslavsky, Prem Prakash Jayaraman, and Shonali Krishnaswamy. Sharelikescrowd: Mobile analytics for participatory sensing and crowd-sourcing applications. In *Data Engineering Workshops (ICDEW), 2013 IEEE 29th International Conference on*, pages 128–135. IEEE, 2013.

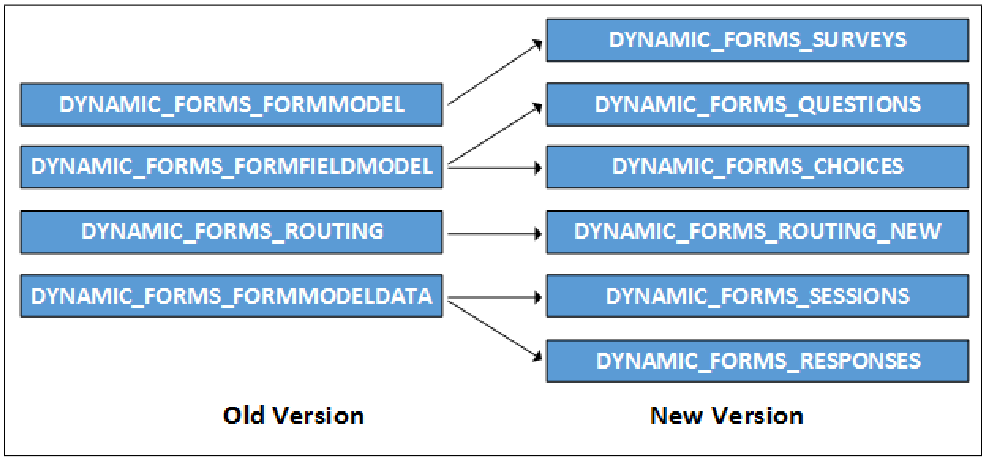
Appendix A

Components Of WiSDM And Their Dependencies



Appendix B

Mapping From WiSDM Old Database Version To New Database Version



Appendix C

USA Disease Prevalence Survey Questions

Questions asked in USA Disease Prevalence October Fall 2015 Survey.

USA Disease Prevalence October Fall 2015

1. What is your age (in years)?
2. What is your gender?
   (a) Male
   (b) Female
3. Including yourself, how many people live at your residence/home?
4. What is your household's yearly income in US dollars?
   (a) 0 - 24,999
   (b) 25,000 - 74,999
   (c) 75,000 or more
5. In the last few weeks did you fall sick with a flu like illness? If yes, which week were you sick in? Check all that apply.
   (a) September - week 1
   (b) September - week 2
   (c) September - week 3

(d) September - week 4
(e) September - week 5 or October - week 1
(f) October - week 2
(g) October - week 3
(h) October - week 4
(i) October - week 5
(j) November - week 1
(k) November - week 2
(l) November - week 3
(m) November - week 4
(n) December - week 1
(o) December - week 2
(p) December - week 3
(q) December - week 4
(r) December - week 5
(s) I did not fall sick

6. How many members of your family (excluding you) fell ill with a flu-like illness in September 2015? Enter 0 if no one fell ill.

7. How many members of your family (excluding you) fell ill with a flu-like illness in October 2015? Enter 0 if no one fell ill.

8. How many members of your family (excluding you) fell ill with a flu-like illness in November 2015? Enter 0 if no one fell ill.

9. How many members of your family (excluding you) fell ill with a flu-like illness in December 2015? Enter 0 if no one fell ill.

Appendix D

WiSDM Data Visualization Code

age by population size distribution

```
session = vi.init('wisdm')

popsz_by_age = vapi.aggregate(session, what_your_age, '
    what_your_age_response_text', 'pop_sz')
popsz_by_age_ordered = vapi.ascending(session, popsz_by_age, '
    what_your_age_response_text')
popsz_by_age_as_array = vapi.build_array(session, popsz_by_age_ordered, ['
    what_your_age_response_text', 'pop_sz'])
restype = { 'x_attr' : 'array_what_your_age_response_text', 'y_attr' : '
    array_pop_sz'}
restype['plot'] = {'title' : 'age by population size distribution',
                          'xlabel' : 'age',
                          'ylabel' : 'population size',
                          'ylog' : 'False'
                }

# line
vapi.plot_XY(session=session, stmt=popsz_by_age_as_array, plot_info=restype)
```

age by sick population size distribution

```
age_sick = vapi.fuseEQ(session, what_your_age, are_you_sick, '
    what_your_age_session_identifier', 'are_you_sick_session_identifier')
    age_sick_yes = vapi.filterEQ(session, age_sick, '
    are_you_sick_response_text', 'Yes')
popsz_by_age = vapi.aggregate(session, age_sick_yes, '
    what_your_age_response_text', 'pop_sz')
popsz_by_age_ordered = vapi.ascending(session, popsz_by_age, '
    what_your_age_response_text')
popsz_by_age_as_array = vapi.build_array(session, popsz_by_age_ordered, ['
    what_your_age_response_text', 'pop_sz'])
arr = vapi.scan(session,popsz_by_age_as_array)

#histogram
vapip.plot_histogram(age, pop_sz, 'age', 'sick population size', '
    age_by_sick_population_size_distribution_bar', 'age by sick population
    size distribution')
```

age range by gender distribution

```
#create a table/class/rmo for age_bins
bins_class = vgsa.build_ranges(session, 'age_bins', range_lb='bin_begin',
    range_ub='bin_end', start=11, step=10, end=100)
bins_rmo = vapi.select(session, bins_class)

#join fact table with bins- it will group the records by age_bins
age_by_bin = vapi.fuseRange(session, what_your_age, bins_rmo, '
    what_your_age_response_text', 'bin_begin', 'bin_end')
bins_by_gender = vapi.fuseEQ(session, age_by_bin, what_your_gender, '
    what_your_age_session_identifier', 'what_your_gender_session_identifier')

#now you can aggregate by bins
popsz_by_bins_gender = vapil.aggregate(session, bins_by_gender, ['bin_begin',
    'bin_end', 'what_your_gender_response_text'], aggLabel='bin_gender_popsz')
popsz_by_bins_gender_ordered = vapi.ascending(session, popsz_by_bins_gender, '
    bin_begin')
arr = vapi.scan(session, popsz_by_bins_gender_ordered)

# plot grouped bar chart
import collections
import matplotlib.pyplot as plt
import numpy as np
dict = collections.OrderedDict()
for row in arr:
        r = str(row[1]) + '-' + str(row[2])
        try:
                dict[r][row[3]] = row[0]
        except:
                dict[r] ={}
                dict[r][row[3]] = row[0]

#print dict
range = [key for key in dict.keys()]
male = [val['Male'] if 'Male' in val else 0 for val in dict.values()]
female = [val['Female'] if 'Female' in val else 0 for val in dict.values()]
N = 9
ind = np.arange(N)

# the x locations for the groups
width = 0.35

# the width of the bars
fig, ax = plt.subplots()

men = male rects1 = ax.bar(ind, men, width, color=(0.2588,0.4433,1.0))
women = female rects2 = ax.bar(ind + width, women, width, color=(1.0,0.5,0.62)
    )

# add some text for labels, title and axes ticks
```

```
ax.set_ylabel('Population Size')
ax.set_xlabel('Age range')
ax.set_title('Distribution by Age and Gender')
ax.set_xticks(ind + width)
ax.set_xticklabels(range)
ax.legend((rects1[0], rects2[0]), ('Male', 'Female'))

def autolabel(rects):
    # attach some text labels
    for rect in rects:
        height = rect.get_height()
       ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,
               '%d' % int(height),
               ha='center', va='bottom')

plt.savefig('gender_by_age_population_size_distribution.pdf')
```