A Framework for Mapping User-designed Forms to Relational Databases

A Thesis

Submitted to the Faculty

of

Drexel University

by

Ritu Khare

in partial fulfillment of the

requirements for the degree

of

Doctor of Philosophy

December 5, 2011

Office of Graduate Studies

Dissertation / Thesis Approval Form

This form is for use by all doctoral and master's students with a dissertation/thesis requirement. Please print clearly as the library will bind a copy of this form with each copy of the dissertation/thesis. All doctoral dissertations must conform to university format requirements, which is the responsibility of the student and supervising professor. Students should obtain a copy of the Thesis Manual located on the library website.

Dissertation/Thesis Title: A Framework for Mapping User-designed Forms to Relational Databases

Author: Ritu Khare

This dissertation/thesis is hereby accepted and approved.

Signatures:

Examining Committee

Chair _____

Members _____

_____

_____

_____

_____

Academic Advisor _____

Department Head _____

Office of Graduate Studies • 3141 Chestnut St. • Randell Hall 240 • Philadelphia, PA 19104
Tel.: 215-895-0366 • **Fax**: 215-895-0495 • **Email**: graduate@drexel.edu • **Web**: www.drexel.edu/provost/graduatestudies

Dedications

To *Sunny* and *Meethi*.

To *Dadi*'s faith and *Nani*'s prayers.

Acknowledgments

This dissertation is by no means an individual effort, and would not be possible without the collaboration of many.

First and foremost, I want to express my gratitude toward *Dr. Yuan An*, my committee Chair. I can't thank him enough for devoting his precious time to develop and train me as a researcher. Under his guidance, I have developed several critical research skills, including problem identification, writing, presentation, experiment design, conduction of user studies, and out-of-the-box thinking. His vision and expertise in data integration and healthcare have been pivotal in shaping my entire dissertation work. I am delighted to have been his first Ph.D. student.

I want to thank *Dr. Il-Yeol Song* for greatly contributing to this dissertation work. Dr Song helped design the gold standard databases used for evaluating the mapping algorithms proposed in this work. With his expertise and experience in conceptual modeling, and systems analysis, he has helped in improving the literature review and the result dissemination style of this work. He has also advised me in terms of effectively planning out the entire dissertation work in a span of four years. I also want to thank Dr Song for being my foremost inspiration in pursuing a Ph.D. degree.

I want to thank *Dr. Jason Li* for enhancing this work to a large extent. Dr. Li, with his expertise in data mining and machine learning, gave several suggestions in choosing an appropriate learning model for designing the term annotation module. This work has greatly benefited from the insightful and intelligent questions, raised by Dr. Li during the candidacy examination and the proposal defense meeting.

I want to thank *Dr. Christopher Yang* for helping improve the work in many aspects. With expertise in artificial intelligence and information retrieval, Dr. Yang has provided many useful comments that helped in formalizing the research focus, enhancing the metrics and terminology, and finding motivational examples in defining the problems.

I would like to thank *Dr. Min Song* for agreeing to serve on the committee as an external member.

Dr Song's expertise in information extraction and Hidden Markov models has been instrumental in devising and improving the form understanding approach proposed in this work. Also, I have used Dr Song's thesis report as a reference in writing this dissertation.

I have been fortunate enough to participate in several other projects that helped in widening the horizon of my dissertation work. For this, I want to thank *Dr. Tony Hu*, *Dr. Weimao Ke*, *Dr. Prudence Dalrymple*, *Dr. Michelle Rogers*, and *Dr. Susan Gasson*. I also want to thank my friends, colleagues, and staff members at the iSchool for the excellent support system.

Finally, I would like to thank God for being with me all the while, and for blessing me with a beautiful family without which I could not even harbor the Ph.D. dream. Words are hardly enough to thank my loving husband *Siddharth* for staying by my side in all kinds of weather; my *Didi* for being my ultimate source of strength and creativity; my *Papa* for teaching me important lessons in time-management and organization; and my *Mummy* for her empathetic listening and unconditional love.

Table of Contents

List of Tables

List of Figures

Abstract

A Framework for Mapping User-designed Forms to Relational Databases
Ritu Khare
Yuan An, Ph. D.

In the quest for database usability, several applications enable users to design custom forms using a graphical interface, and forward engineer the forms into new databases. The path-breaking aspect of such applications is that users are completely shielded from the technicalities of database creation. Despite this innovation, the process of automatically integrating a new form into an existing database remains unexplored. At large, databases continue to remain unusable. This dissertation focuses on investigating the problem of mapping multiple forms into an existing relational database. We seek a framework that automatically detects and merges the semantically matching elements between forms and databases. Upon encountering the unmatched form elements, the framework creates new database elements and integrates them with the underlying database. The technical goal is to ensure that the resultant database is compliant with "high quality" principles defined in terms of form semantics. The usability goal is to ensure minimalism in the user interventions required to discover correspondences.

We introduce a model, the *form tree*, to represent the user's semantic intentions represented by a form. We design two anchor approaches to extract the form tree from an arbitrarily-designed form, and to further disambiguate the form semantics by annotating its terms using standard concepts. Thereon, we formulate the following mapping solution: (i) Leverage linguistics and semantics to discover and validate semantic correspondences between the form tree and the existing database. (ii) Devise mapping algorithms that create a new high-quality database for a given form tree, and merge it with the existing database while fusing the semantically equivalent elements. We evaluate the entire framework by developing a prototype, and experimenting in the healthcare domain. We collect 52 clinical forms from different medical institutions, and map them to 6 databases of varying scales. The anchor approaches generate form trees with 98% accuracy, and annotate the form terms

with a precision of 0.89. The framework helps in producing up to 74% principle compliant databases in terms of compactness, and in reducing user interventions by 61%. The experiment results imply that the use of annotation helps in improving the quality of the evolved database.

Part I

Introduction

Chapter 1: Motivation

Research in "database usability" continually strives to bridge the gap between users and databases[1]. A popular aspect of database usability involves the information retrieval (IR) algorithms that enable users to "search"[2–4] and "query"[5–10] across complex databases. Another aspect that involves enabling users to "build" databases has only lately gathered some attention[11]. Recently, certain Do-It-Yourself (DIY)[12] and What You See Is What You Get (WYSIWYG)[13] paradigms have been developed for enabling the non-technical users to build databases on their own. Applications in these paradigms are form-driven, i.e., they leverage the facts that knowledge of data-entry forms is already ingrained in the users[14;15], and that the forms contain important information about databases[14;16]. These applications enable users to design custom forms, and automatically translate forms into underlying databases while shielding users from the technical details for database creation and code generation. Example applications include Formassembly[17], Zoho Creator[18], Jotform[19], and Wufoo[20].

Despite this innovation in database birthing, most databases remain inflexible with respect to the changing needs of the users[21], and hence remain largely unusable from an integration point of view. The following example from the healthcare domain illustrates the problem.

Example 1.0.1 *Clinicians are dependent on the health information technologies (HIT) in their daily activities such as collecting customized data related to patients, diseases, treatments, etc. Figure 1.1 shows a form and an associated back-end healthcare database. The application maintains a mapping between the form and the back-end database.*

Suppose a new form as in Figure 1.2, reflecting a new data collection need, is designed to collect data into the same database. The aim is to collect multiple kinds of needs into a holistic model that would provide an integrated view of the clinical information to the users. A technical developer would first link the Name, Sex, Date of Birth, *and* Marital Status *items on the form to the existing* Patient *table in the database. She would then extend the existing database properly to collect the new data*

items under the Social Activities *group on the new form.*

Materialization of this problem entails the following: (i) the ***building*** *of new forms, wherein a technical developer collaborates with the domain experts (i.e., the clinicians) in order to understand the new needs. (ii) the* ***integration*** *of new forms over the existing back-end database, wherein a technical developer directly accesses the database system; studies the existing, possibly complex, schema; and writes the appropriate application code.*

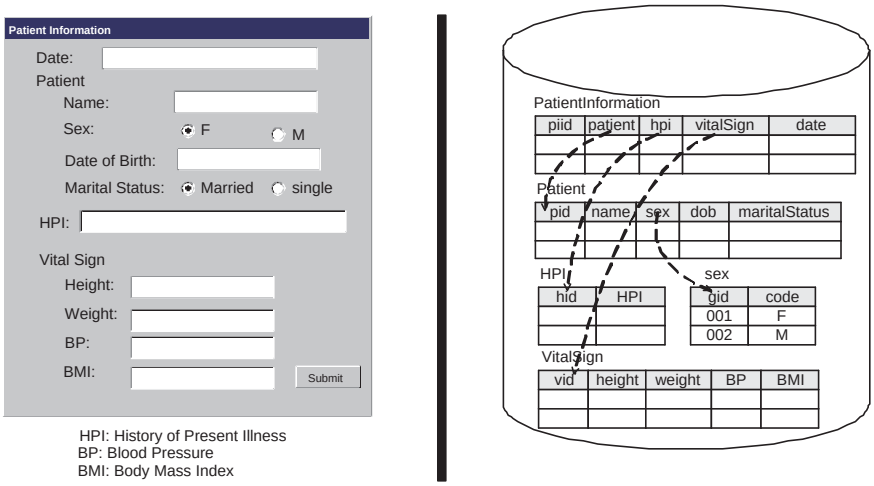


Figure 1.1: A Form and the Associated Database

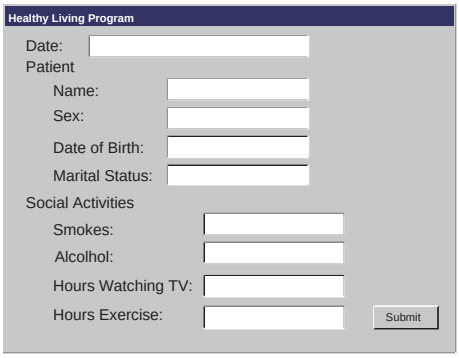


Figure 1.2: A New and Interrelated Form

■

Let us study the scenario from the usability standpoint. The form building task can be improved by using the existing DIY solutions that enable non-technical users for creating forms and databases

while leveraging the users' intuitive knowledge of data entry forms. This would also help reduce the impedance mismatch between the clinicians' needs and the back-end databases that may exist as a result of the miscommunication between the non-technical and technical users[21]. Improving the integration task, however, is complicated. There is a lack of appropriate solutions to make this more usable to the non-technical users, primarily, because these users do not possess the necessary skills to understand a database schema and write a suitable application code to extend the schema. Even for a technical expert, the integration process is quite tedious, error-prone, and time-consuming[22] and often leads to unintended consequences[21;23;24].

This motivation makes it desirable to develop a system that automatically maps the user-designed forms to the underlying databases. Upon encountering the form elements that do not correspond to any elements in the database, the system should automatically create new elements and merge them with the existing database. **This dissertation focuses on investigating the problem of automatically mapping and integrating multiple forms into an existing structured database.**

Chapter 2: Research Problems

We are interested in building a start-to-finish framework that allows users to build forms on their own, and that automatically maps and merges the user-designed form into the hidden relational databases. Since many DIY solutions exist that allow users to build complicated forms, we assume that a user-designed form is already acquired, and that it needs to be mapped to an existing, possibly complex, relational database. As we walk through the sketches of possible solutions, we come across the following problems in that order.

2.1 Form Understanding

The first step in developing a solution to the mapping and integration process is to understand the semantics of user-designed forms. A form contains a sequence of form elements, i.e., text-labels and input elements (textbox, selection list, etc.). We assume that the data-entry form is in a popular machine readable format such as the HTML. Some example forms are shown in the Figures 1.1 and 1.2. A form is primarily designed for human understanding and reflects the semantic intentions of the designer, in particular, the hierarchical parent-child relationships. For instance, the element *Height* falls under the *Vital Sign* category in the Figure 1.1.

While humans easily perceive a data-entry form based on past experiences and visual cues, machine processing of a form is challenging. There is an infinite number of possible form layout patterns[25], and the forms collected from different sources, possibly designed by different designers, are even more diverse (See Figures 2.1, 2.2, 2.3). Given this diversity, a machine can only read the **syntactic** structure of a form, which includes the elements, their sequential order, and the associated formatting. There is no standard convention that associates a certain layout pattern with a certain semantic intention. Since mapping is a semantic problem, we are interested in the **semantic** structure that captures the semantic intentions associated with the form elements. This semantic structure, however, is not directly readable as it is not captured in a form's source code.

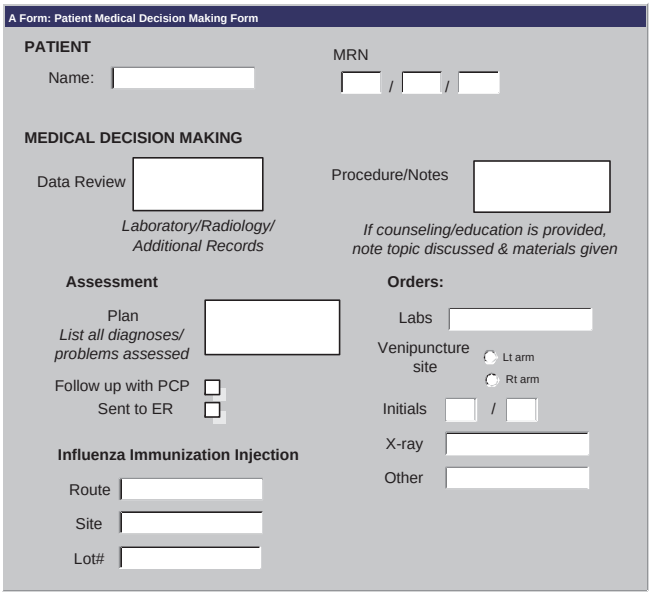


Figure 2.1: A Form from Source A

Figure 2.2: A Form from Source B

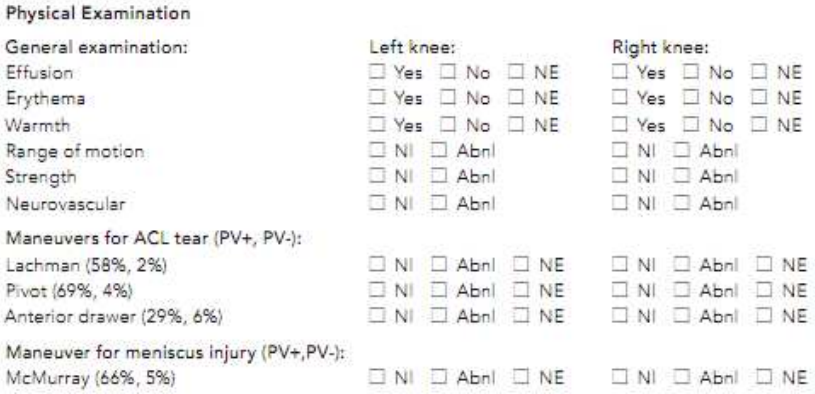


Figure 2.3: A Form from Source C

2.2 Correspondence Discovery

Once the form's semantic structure has been extracted, the next step is to link the form elements to the corresponding semantically matching elements in the existing database. We refer to this as the correspondence discovery problem.

At first glance, the problem of correspondence discovery may appear as yet another flavor of the longstanding problems of schema and ontology matching problems[26–30]. One may wonder whether existing mapping techniques are sufficient to discover the correspondences between forms and databases. The main roadblock in doing so is that so far no fully automatic solutions are available for the schema mapping problem. If we simply adopt a semi-automatic solution to the form to database mapping problem, then the system would require users to examine the intermediate results. The examination would in turn need technical knowledge and background. Such a requirement would diminish the value and usefulness of the DIY form to database mapping tool that we have envisioned. Also, there are certain conceptual differences between schema mapping and form to database mapping problems. We briefly point out these differences as we enlist the challenges associated with the discovery problem.

- *Heterogeneity:* Unlike schema mapping, form to database mapping is a case of mapping two heterogeneous structures. Semantically merging two heterogeneous resources is a complex problem[31].

- *Term Variations:* Different users may use different terms to specify the same semantic concept on the form. Reconciliation of this term variety can be accomplished by using existing linguistic similarity based techniques such as exact match, substring, tokenize, stemming, thesaurus, abbreviation, synonyms, etc. Some form specific challenges include handling multi word terms, handling long terms, and identification of relevant concepts from the long terms.

- *Correspondence Combinations and Validation:* The results of schema mapping are relationships between elements in different schemas, while mappings between forms and databases involve not only schema elements but also data values. This just leads to more number of pos-

sible combinations of correspondences, and hence more complications. The initially identified correspondences fall under the following categories.

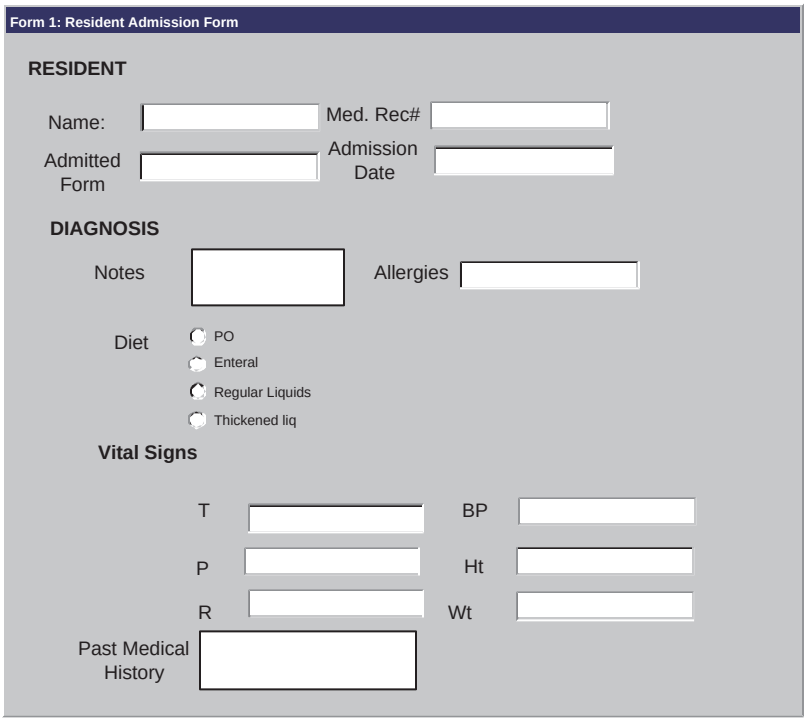


Figure 2.4: Resident Admission Form

- 1-1: A form element could match with a single database element. However, certain form terms may match linguistically but may differ semantically from one another, e.g., the form element *Vision* in Figure 2.5 linguistically matches the with database element, column Eyes in Figure 2.6 but this correspondence is not valid. Therefore, a validation procedure is needed to eliminate the semantic mismatches.
- 1-M: A form element could match with multiple database elements. In this case, either none or one of the correspondences could be valid. An example of 1-M correspondence is the form element R (as in "respiratory") in Figure 2.4 that matches with the database elements, the column RR and the table Respiratory.
- M-1: Several form elements could match with a single database element. In this case, none or many correspondences could be valid as certain elements are repeated in the

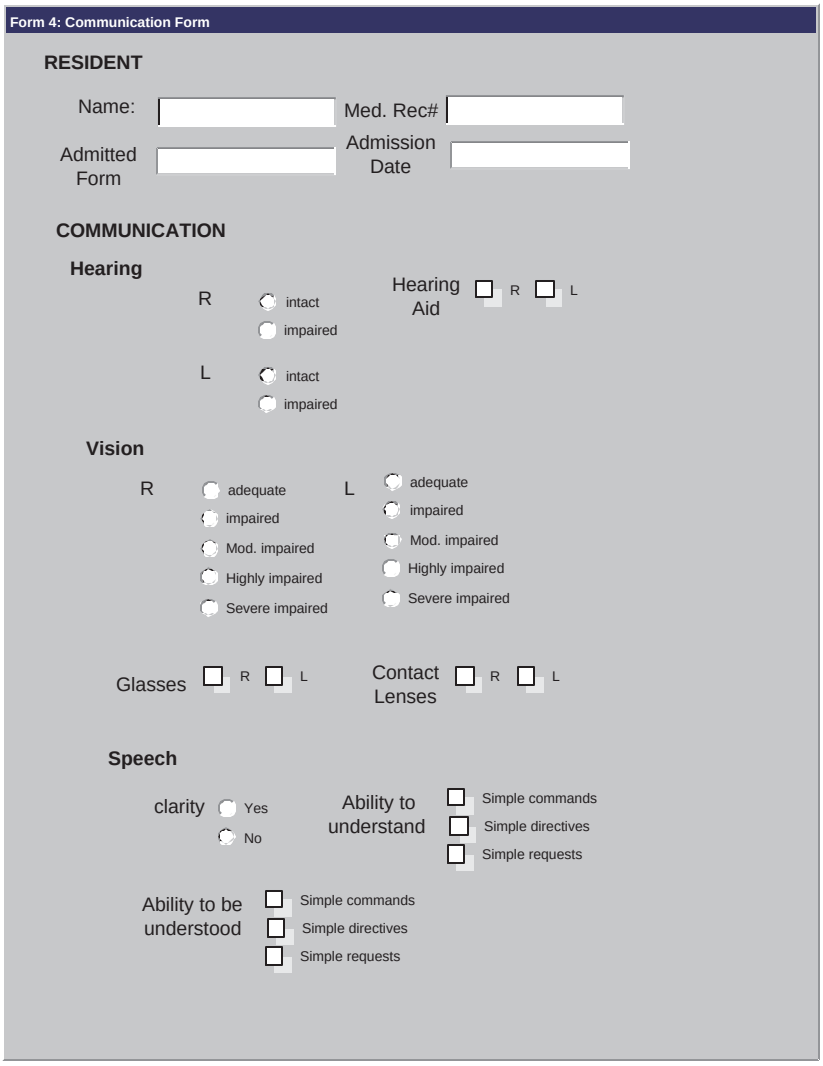


Figure 2.5: Communication Form

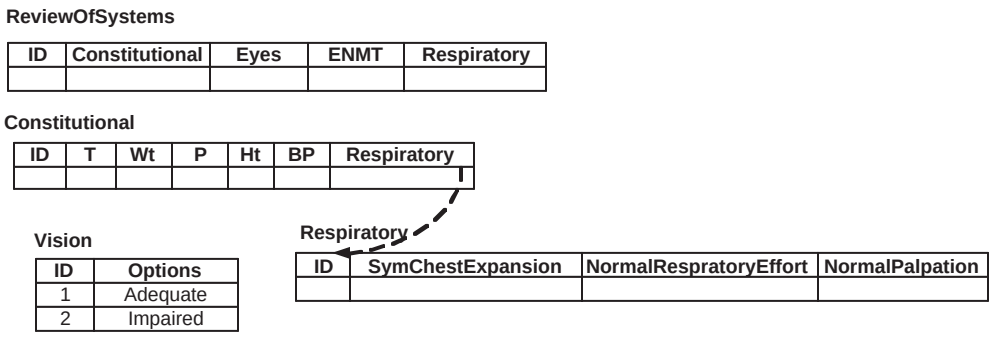


Figure 2.6: An existing relational database

form. For example, both the form elements labeled as *adequate* in the Figure 2.5 match with the database element Adequate in the Figure 2.6 and both the correspondences are semantically valid.

In sum, once the initial correspondences are identified, using certain linguistic techniques, the next challenge is to eliminate the invalid correspondences possibly by analyzing the structures of both the forms and the databases. The heterogeneity between the structures makes this more challenging.

- *Evolution Requirement:* Schema mapping, in itself, does not consider the problem of schema and database evolution when there are elements that cannot be matched. Form to database mapping is more sophisticated in that the resultant correspondences are used for evolving the database for the unmapped elements in the form, and the mapping discovery process has to consider this requirement well in advance.

2.3 Form Integration

Once the semantic correspondences between form elements and database elements have been discovered, the next problem is to integrate the form elements to the database, i.e., to physically merge the mapped form elements with the semantically matching database elements, and to evolve the database with respect to the new, i.e., unmapped, form elements.

2.3.1 Merging into an Existing Database

We merge the matching form elements with the corresponding database elements so that the same concept is not duplicated in the database, and the database remains compact. The problem at hand is to merge a form into a database given the set of validated semantic correspondences. When a semantic correspondence is created between a form element and a database element, the problem is to merge the elements along with the respective local structures. Merging challenges are similar to the database integration challenges[32]. However, since the local structures are heterogeneous, existing solutions can only provide a guideline and cannot be directly applied. Merging a form into an existing database often leads to some conflicting structures[26]; these are described next.

1. *Table Merging Conflicts:* These conflicts occur when a form element is mapped to a database table. The Figure 2.7 demonstrated two of such scenarios. In the Form A, the element *Resident* needs to be merged with the table Patient. The form element comes with its own context, i.e., the fields *Name*, *Med. Rec. #*, etc. Hence the question is whether to create a separate table for the Resident element, and make the 4 child elements as its columns, or to merge all the form elements into the existing Patient table. The first option would lead to duplication of the same concept, and the latter option would however lead to a larger number of NULL values due to the past mappings. Another merging situation is demonstrated for the Form B wherein the element *Vision* corresponds to the database table Vision. However, the form element is associated with more specialized semantics (like the elements R and L) as compared to the one reflected in the existing database. Accommodating such elements is challenging.

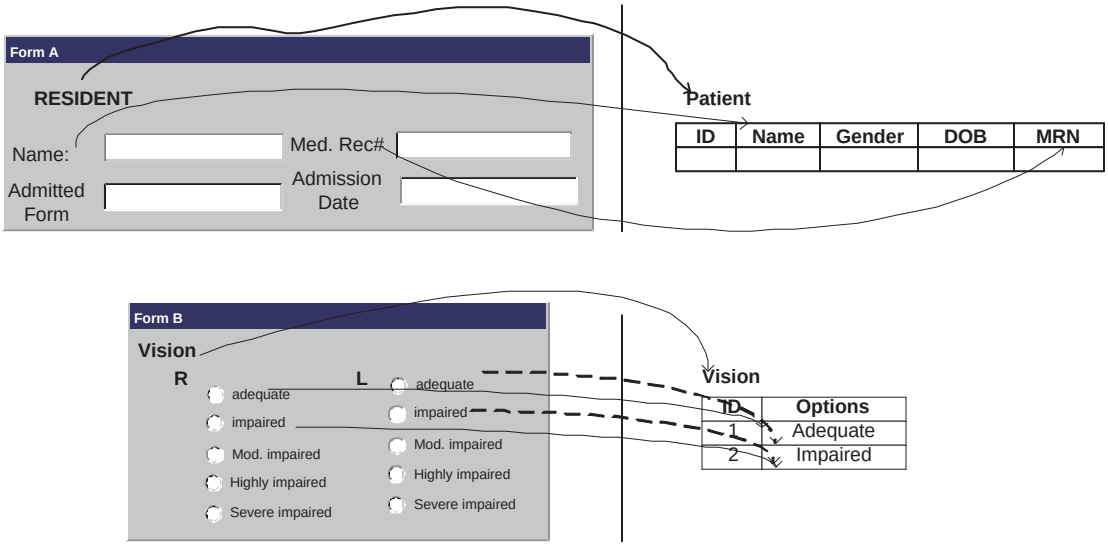


Figure 2.7: Table Merging Conflicts

The goal of the merging process is to retain the contextual information associated with the form element. A challenge is how to merge the elements while accurately escalating all the semantic form element information into the database. In traditional schema integration, it is suggested that a higher abstraction level should be chosen when there is a conflict in the representation of a concept [32].

2. *Column Merging Conflicts:* These conflicts occur when a form element is mapped to a database table column. The Figure 2.8 presents a couple of such scenarios. In the Form C, the element *Past Medical History* corresponds to the database column History.PastMedHistory. The challenge is to accommodate the parent form element, i.e., *Diagnosis*; i.e., whether to merge and connect the matching elements through foreign keys, or to duplicate the elements in separate tables. Similarly, for the case of Form D, the question is how to reflect the form elements associated with *Memory* in the database while retaining the association between the form element *Memory*, and the database column Psych.Memory. Overall, similar to the previous conflict,

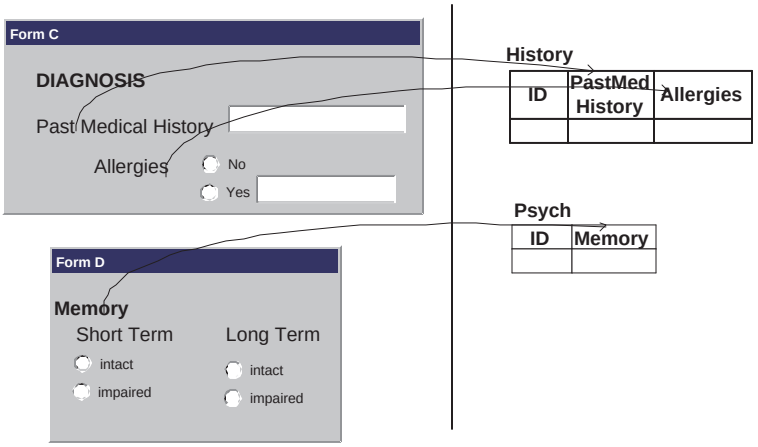


Figure 2.8: Column Merging Conflicts

the goal is to resolve the difference in structures while accurately reflecting the form semantics in the final integrated database.

3. *Structure Conflicts:* Also known as geometric conflicts, such conflicts arise due to the structural differences between the form and the database. A special case of such conflicts are the cardinality conflicts. Consider the database in the Figure 2.9, the referential integrity constraint suggests that a physician can be associated with multiple patients. In contrast, the Form E clearly states that a patient can have multiple physicians associated with her. To resolve this, we cannot simply remove the foreign key as it may affect the previous mappings. One solution is to create a join table that reconciles both the structures. While this is a simple example, the structural conflicts are likely to get very complicated as the database scales up. In database

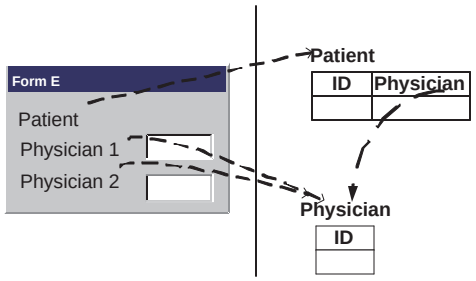


Figure 2.9: Structure Conflicts

integration, such conflicts are resolved by taking the less constraining structure as the final structure[32], however this is not feasible in integrating forms into databases, because all the structures need to be stored in the database, other it would disrupt the previous mapping.

In addition, there are some data-level conflicts[33] and term conflicts that are beyond the scope of the problem in focus. Overall, the goal of conflict resolution is (i) to reconcile the differences between the local structures of both the mapped elements, (ii) to maintain the original semantics of both the form element and database element in the evolved database, and (iii) to do so while considering the database design principles like minimize null values, and avoid duplicating the same concept.

2.3.2 Birthing a New Database

During form integration, some form elements are likely to not match with any of the existing database elements. This may happen when either the form to be mapped is unrelated or partially related, or when the database is empty. Given this, the form to database mapping framework should be able to merge the matching elements, and also to extend the hidden database for the unmatched elements on the form. We call the process of generating new database elements out of the form elements as *birthing*. Birthing comes with its own set of challenges such:

1. How to automatically derive the basic ingredient of relational database design, i.e., the functional dependencies, among the form elements?

2. How to automatically derive cardinalities among the form elements?

3. How to handle the complex design patterns on forms such that one shown in the Figure 2.10.

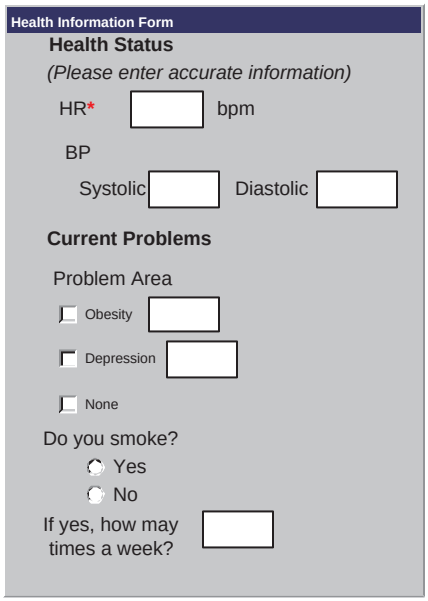


Figure 2.10: Some Complex Form Patterns: *Please enter ...* represents a miscellaneous label, *BP* is a subcategory of *Health Status*, *Obesity* is an extended radiobutton option, *Do you smoke...* and *If yes* are conditionally related fields

4. For a given form pattern, how to evaluate multiple design alternatives and choose one?

2.4 Research Questions and System Goals

Given the challenges associated with the aforementioned problems, the key research questions are:

- Form Understanding:
  - What could be an appropriate model to accurately capture the semantic structure of a given user-designed form?
  - How to automatically extract this semantic model from an arbitrarily designed form?
- Correspondence Discovery:
  - How to represent the form and the database into equivalent and compatible structures such that correspondences can be built?
  - How to determine the semantically equivalent database elements for the given form elements?

  - How to incorporate the form database evolution requirement during the correspondence discovery process?

- Form Integration:

  - How to resolve the merging conflicts while maintaining the original semantics of forms in the database?

  - How to automatically derive a relational database corresponding to an arbitrary pattern of form elements?

In this thesis, we seek the answers to these research questions through the development of a system that automatically maps a user-designed form to an existing database. The first goal of the system is to ensure that the resultant database accurately maintains the semantics of the mapped form, and is principled from the context of the classic database design theories. The second goal of the system is the ensure the minimality of the required user intervention during the process of correspondence discovery.

Chapter 3: Research Methods

To answer the research questions associated with the mapping problem, we propose several algorithms leveraging, in particular, the rich semantics embedded in user-designed forms. We adopt a systems-based empirical approach to evaluate the entire framework. In this chapter, we conceptualize the terminology adopted in the subsequent sections, throw some light on the proposed approaches, and describe the empirical methods.

3.1 Formalism

We now formally describe the terminology and concepts that were briefly touched upon in the previous sections and that will be adopted in the subsequent sections.

3.1.1 Form

The primary object of this study, a form, is formally defined in Definition 3.1.1. A form consists of a collection of form elements laid out in a particular way. Example form elements include text label, text box, radio buttons, select list, and check boxes. The data type of an element can be extracted from the source code of a form, for example, Date for calendar input. Furthermore, the source code of a form often provides constraint information, e.g., whether an input is required or optional. In the subsequent sections, we adopt a semantic representation of a form known as the *form tree*, formally described in Definition 3.1.2. The form tree is a hierarchical tree structure, e.g., Figure 3.1 shows a form and its corresponding form tree. A form tree captures the semantic intentions of the designer. Each node in the tree represents a form element. Graphically, we use different shapes to represent the different types of nodes. Each edge in the tree represents a semantic association between any two form elements.

Definition 3.1.1 (Form) *A Form is a sequence of form components where each component is either a text-label or a form-input such that*

1. *A form-input is one of the following* formats: textbox, textarea, radiobutton group, checkbox group, *or* dropdown list *similar to their specification in HTML 4.0.*

2. *A text-label is one of the following:*

   - field, *the label associated with one or more form-inputs.*
   - category, *the group label that semantically covers one or more* fields *and their associated* formats.
   - subcategory, *the sub-group label that is laid under the semantic scope of a* category *and that further contains (*sub-*)*fields *and associated* formats.

Definition 3.1.2 (Form Tree) *A form tree is defined as a labeled, directed and ordered tree,* $\mathcal{FT} = (N, E, <_{sib}, root)$*, where* $N = \mathcal{I} \cup \mathcal{E} \cup \mathcal{V}$ *is a finite set of nodes,* E *is a finite set of edges,* $<_{sib}$ *is the next-sibling relation between children of a tree node, and* $root \in N$ *is the root of the tree. Moreover,*

- $\mathcal{I}$ *is a finite set of input elements (or inputs), where items of* $\mathcal{I}$ *are drawn from the following set of inputs:* {text box, text area, radio buttons, check boxes, drop-down list, calendar} *;*
- $\mathcal{E}$ *is a finite set of logical elements. Each* $e \in \mathcal{E}$ *has a label* $l = \lambda(e)$*, a data type* $t = \tau(e)$*, and a constraint* $k = \kappa(e)$*, where the function* $\lambda(e)$ *returns the label* l *of* e*, the function* $\tau(n)$ *returns the data type* t *of* e*, and the function* $\kappa(e)$ *returns the constraint* k *of* e*.*
- $\mathcal{V}$ *is a finite set of values;*
- *For an edge* $(n_i \longrightarrow n_j) \in E$*,* $n_i, n_j \in N$*,* n_i *is called* **parent** *and* n_j *is called* **child**.

3.1.2 Database

Definition 3.1.3 (Database) *A (relational) database* $\mathcal{D} = (I, R, \Sigma)$ *is a 3-tuple, where*

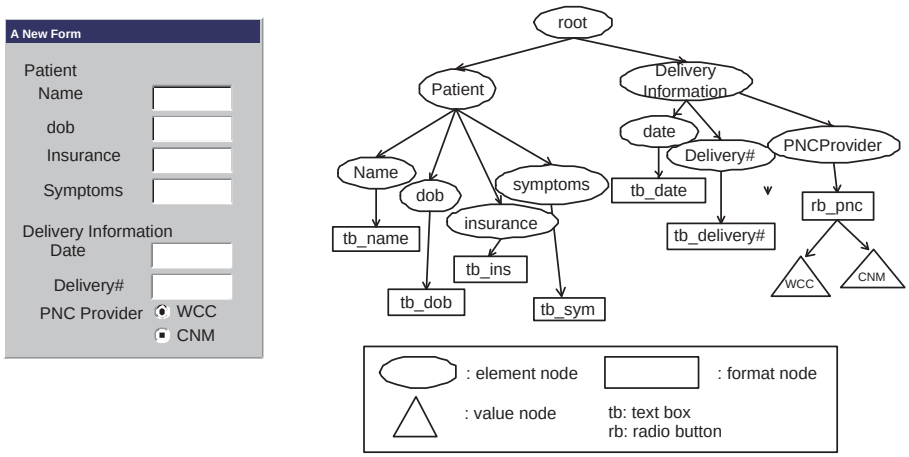


Figure 3.1: A Form and its Associated Form Tree

- *I is a set of relations. A relation T consists of a set of tuples conforming to its schema. $t_T in T$ represents a tuple in table T and $t_T.A_i$ refers to all the values under the attribute A_i of table T and $t_T.v_{A_i}$ refers to a specific value v under the attribute A_i.*

- *R is the schemas of the relations. The schema for a relation specifies the name of the relation, the name of each column (or attribute or field), and the type of each column. We use the notation $T(A_1, A_2, ..., A_n)$ to represent the schema of a relation T with attributes $A_1, A_2, ..., A_n$, and we use the notation $T.A_i$ to refer to the attribute A_i of the relation T.*

- *Σ is a set of integrity constraints imposed on the relations. The integrity constraints impose conditions that the tuples in relations must satisfy. Here, we consider the* key *and* foreign key *constraints. A key in a relation is a subset of the attributes of the relation that uniquely identifies a tuple. A foreign key in a relation T is a set of columns F that* references *the key of another table T', and imposes a constraint that the projection of T on F is a subset of the projection of T' on the key of T'.*

3.1.3 Mapping

Definition 3.1.4 (Mapping) *A (transformation) mapping $\mathcal{M}$ from a form tree $\mathcal{FT} = (N, E, <_{sib}, root)$ to a database $\mathcal{D} = (I, R, \Sigma)$ is a set of correspondences and predicates such that*

- *a correspondence* $\mathcal{FT}$*:P/e* $\rightsquigarrow$ $\mathcal{D}$*:D.d relates a node* $e \in N$ *of the form tree reached by a simple path* P *to an element* d *in the database component* D *where*
  - *A simple path* P *is always relative to the root of the form tree, in which "/" is used to represent a parent/child relationship.*
  - *A database component could be either a tuple in a table* T *or the schema of the table.*
- *Based on various mapping scenarios, we identify two kinds of correspondences:*
  - *Regular: the element node* $e \in \mathcal{FT}.\mathcal{E}$ *maps onto a table* T *in the database, or onto to the attribute* $T.A_i$ *in the database, or onto a value* v_{A_i} *of an attribute* $T.A_i$ *in the database.*
  - *Data Binding: the element node* $e \in \mathcal{F}$ *maps onto the attribute* $T.A_i$ *such that the node* e *is a container for the values stored in* $T.A_i$*.*
- *A predicate specifies the reference edges in the database graph* $\mathcal{DG}$*. Each predicate is an edge* $(m_i, m_j) \in \mathcal{DG}.D$*,* $m_i \in \mathcal{C}, m_j \in \mathcal{T}$ *in the database graph, such that the column node* m_i *reflects a foreign key column in a referencing table in the database, and* m_j *reflects the referenced table in the database.*

While studying the mapping problem, we realize that there are certain common principles that need to be taken into account to effectively carry out the processes of form understanding, correspondence discovery, and form integration. We formally present the principles in the form of the mapping principles. Overall these principles ensure that the resultant mapping and the database are high quality and optimized so that eventually the data collected through these forms is also high quality[34]. We borrow the ideas from the standard database textbooks (e.g.,[35]) contain rich content about designing normalized databases with "good" properties including avoiding logical inconsistency and update anomaly. These are summarized next.

Quality: A database expert, while designing databases, aims for certain "good" properties of a database as recommended by the standard database textbooks (e.g.,[35–37].). In the same lines, we summarize the properties for deriving mappings in the following manner.

- $\mathcal{P}1$ **Correctness**. *Given a form tree* $\mathcal{FT} = (N, E, <_{sib}, root)$ *and a database* $\mathcal{D} = (I, R, \Sigma)$. *The mapping* $\mathcal{M}$ *from form to database is correct, if and only if each correspondence* $\mathcal{FT}$:P/e $\rightsquigarrow$ $\mathcal{D}$:D.d *is correct, i.e., both elements* P/e *and* D.d *contextually represent the same concept in the application domain. Correctness can be assessed based on label matching and context matching. The context of a node refers to the connections, i.e., edges, among various form elements in the tree/graph. Overall, the correctness of a database generated out of a form tree stipulates that the structure of the form tree is accurately maintained in the database.*

- $\mathcal{P}2$ **Completeness**. *Given a form tree* $\mathcal{FT} = (N, E, <_{sib}, root)$ *and a database* $\mathcal{D} = (I, R, \Sigma)$. *The mapping* $\mathcal{M}$ *from form to database is complete, if and only if for each data item collected on the form, there is a correspondence in a mapping* M_i *which maps the data item to a database element.*

- $\mathcal{P}3$ **Compactness**. *Given a form tree* $\mathcal{FT} = (N, E, root)$. *A database* $\mathcal{D} = (I, R, \Sigma)$ *is compact in terms of storing data collected on the form, if and only if there is at most one correct mapping* M: $\{\mathcal{FT}$:P/e $\rightsquigarrow$ $\mathcal{D}$:D.d$\}$ *for any node* $e \in N$ *of the form tree.*

- $\mathcal{P}4$ **Normalization**. *We say the database* $\mathcal{D}$ *is a normalized database with respect to the form tree* $\mathcal{FT}$ *if the database* $\mathcal{D}$ *is in 3NF with respect to all functional dependencies associated with the* $\mathcal{FT}$.

Optimization: Given a form, there could be several mapping alternatives that comply with the aforementioned quality principles. A professional designer selects the alternative that minimizes any potential query processing issues in the evolved database. Along the same lines, we describe some optimization principles[38] as presented next.

- $\mathcal{P}5$ **Minimize foreign-key NULL values**. This principle stipulates that the mappings should minimize the possibility of having NULL values in a foreign key column of the database to avoid any loss of information while performing the JOIN queries.

- $\mathcal{P}6$ **Minimize non-key NULL values**. This principle stipulates that the mappings should minimize the possibility of having NULL values in a any non-key column, particularly a numeric

column, to avoid retrieving inapplicable results while performing queries involving aggregate functions such as SUM and AVERAGE.

- $\mathcal{P}7$ **Minimize database elements and joins**. In case of multiple mapping alternatives each satisfying the above principles, $\mathcal{P}1$-$\mathcal{P}6$, the alternative that leads to fewer number of tables, columns, or foreign key references should be selected.

3.2 Approaches

We propose a framework that maps an arbitrarily designed data-entry form into an existing relational database. The steps involves and the approaches proposed are described next.

1. *Form Understanding:* The first step is to understand the designer's semantic intentions that are implicitly embedded in the form. To represent the intentions, we introduce a semantic structure known as the form tree. We develop a machine learning technique based on layered Hidden Markov Models(HMMs). Using this, the system is able to simulate the process through which a human being understands the form semantics, and thus able to extract the form tree from an arbitrarily designed form.

2. *Form Tree Annotation:* To overcome the challenges encountered in the subsequent steps, we further disambiguate the semantics of the terms, i.e, the *label elements*, in the form. Once the form tree is retrieved, the framework annotates each node with an appropriate standard concept from a popular terminology. To perform the annotation, we develop a machine learning technique based on Naive Bayes Classifier that exploits the semantic structure of the form to derive a unique standard concept for a given tree node. The annotated node represents the precise semantics of the contained term, thereby overcoming many challenges associated with correspondence discovery and form integration.

3. *Correspondence Discovery:* The next step is to discover the correspondences between the form tree elements and the existing database elements. To accomplish this, we adopt a hybrid technique leveraging both the term linguistics and the semantic annotations of the form and the database elements.

4. *Correspondence Validation:* Once the correspondences are discovered, the next step is to ensure their correctness. A naive method is to present all the correspondences to the user for further validation. However, this would lead to a large number of user interventions especially when mapping to a large scale database. Hence, we devise a validation algorithm, that automatically validates or eliminates certain correspondences. The algorithm is designed in a decision tree fashion that evaluates the correspondences based on the local structure of the form tree and the database. The correspondences that are yet unvalidated are forwarded for user intervention.

5. *Database Birthing:* After the form tree has been derived and the correspondences have been discovered, the framework generates a new database corresponding to the form tree to reconcile the structural differences between the form and the database. The birthing algorithm is designed while considering various quality and optimization mapping principles, i.e., $\mathcal{P}1$ through $\mathcal{P}7$. The algorithm is based on the assumption that the parent-child association in the semantic form tree accurately reflects the functional dependency information required to design a normalized database. The birthing algorithm works in an incremental manner while translating each form pattern into equivalent database elements, and eventually generates a complete database.

6. *Database Evolution:* Finally, the discovered correspondences are transferred to the new database, and are used as anchors for merging the new database with the existing database. We design a merging algorithm to accomplish the final integration step. The algorithm merges the equivalent elements together in the final database while establishing a configurable trade-off between the compactness principle $\mathcal{P}3$ and the minimize NULL-value principle $\mathcal{P}6$.

3.3 Evaluation

We adopt an empirical approach to evaluate the framework. The modular design of the framework gives an opportunity to design multiple experiments for assessing various aspects. We develop a functional prototype of the framework and perform experiments with forms currently being used to collect clinical data in various healthcare institutions. Through the experiments, we measure how

each part of the framework facilitates in accomplishing the primary goals of principle compliance and user intervention minimization. In particular, we design the following experiments.

1. *Automatic Form Understanding:* This experiment is conducted to measure the performance of the HMM-based form understanding module. We measure the accuracy of the generated form trees by comparing with the "gold" form trees that capture the actual semantic intentions of the designer. The results contribute to the compliance of the final results with the correctness $\mathcal{P}1$, the completeness $\mathcal{P}2$, and the normalization $\mathcal{P}4$ principles.

2. *Concept Annotation of Form Terms:* This experiment is designed to measure the performance of the concept annotation module. We design multiple variations of this module with different combinations of semantic structural information and linguistic information of the form elements. We select the variation that produces the best performance in terms of recall and precision, and use it for further experiments. The annotation module helps in extracting more precise semantics from the individual form elements, and thus also contributes to the overall correctness (i.e., principle $\mathcal{P}1$) of the resultant database.

3. *Form to Database Mapping:* This set of experiments is designed to measure the performance of the entire mapping framework. The experiment begins with a given form tree and an existing database, followed by correspondence discovery and validation, and database birthing and evolution. We design three variations of the experiments: (i) when the form tree is raw, i.e., unannotated, and the correspondence discovery is performed using linguistic match (ii) when the form tree is concept annotated, and the correspondence discovery is performed using exact concept matching, (iii) when the form tree is concept annotated, and the correspondence discovery is performed using hybridization of concept and linguistic matching techniques. We measure several aspects of the framework, including the number of user interventions, relevance of the intervention screens, percentage of approved mergers, extent of annotation, etc. Overall, we measure the compliance to design principles, in particular, the compactness principle, $\mathcal{P}3$, and the number of required user interventions across different variations.

Chapter 4: Contributions

This thesis work makes several contributions including discovering new research problems, devising successful solutions, drawing implications for further research, and identifying the limitations of the proposed methods and experiments.

4.1 New Problems

There are many existing tools that enable users to design forms and forward engineer the forms to new relational databases. However, the problem of integration and mapping of new forms to existing databases has never been proposed or studied in the past. In this thesis, we formally conceptualize the problem of mapping user-designed forms into databases. This problem is comparable and yet different from the longstanding schema matching problem in several aspects. A solution to this problem would enable the users to automatically integrate and induce new needs into the existing databases, and thereby evolve the databases on their own.

In the process of developing a solution to the mapping problem, we discover another novel problem of annotating form terms, instinctively supplied by the users, using standard terminology concepts. We focus on the healthcare domain and study the problem of finding an equivalent SNOMED CT concept for a given user-defined form term. There have been several works on annotating the clinical artifacts such as free-form notes written by nurses. However, the problem of standardizing form terms has never been studied before. We believe that a solution to this problem is likely to facilitate the process of integrating new forms to relational databases.

4.2 New Solutions

While addressing the mapping problem, we identify several issues arising in the development of a viable solution. We elucidate the possibilities and limitations, and develop the following novel approaches.

- Form Tree Extraction Algorithm: To address the much explored problem of automatic form understanding, we develop a novel solution to extract a semantic structure, i.e., the form tree, from an arbitrarily designed data-entry form. The solution is a machine learning based approach that comprises a layered Hidden Markov Model and certain tree design rules.

- SNOMED CT Annotation Algorithm: To annotate a given form using SNOMED CT concepts, we develop a machine learning based solution that leverages the semantic structure as well as linguistic of a form term to derive the equivalent concept in the SNOMED CT. The key component is a Naive Bayes classifier that analyzes the semantic structure of a form and classifies each term with respect to a semantic category in the SNOMED CT.

- Correspondence Validation Algorithm: We develop a heuristic-based solution to automatically validate the initial discovered correspondences between forms and databases and minimize the need for user intervention. Given a form term and a database element, the approach analyzes the local semantic structure of the form term and the database element and decides whether the correspondence is plausible or not.

- Birthing Algorithm: Given a semantic form tree, we develop a birthing solution to develop a new database corresponding to the form tree. The significance of the algorithm is that it is inspired by the quality and optimization principles. For any given form pattern, the algorithm ensures the compliance of the resultant database with these principles.

- Merging Algorithm: Given the new database corresponding to a given new form, an existing database, and the discovered correspondences, we develop a merging solution to merge the two databases while establishing a trade-off between the quality and the optimization principle.

4.3 Results and Implications

We conduct extensive experiments in the healthcare domain using 52 highly complex data-entry forms collected from 6 medical institutions. These forms are mapped to evolve 6 databases of varying scales, with at least 35 and at most 450 tables. The empirical analysis makes the following contributions and implications.

- The form extraction solution accomplishes up to 98% accuracy in representing any arbitrarily designed form. The algorithm takes less than a second to derive a semantic tree of average scale, i.e., with 135 edges.

- The concept annotation solution achieves an average precision of up to 0.89 and an average recall of up to 0.76. It takes at least 1 and at most 11 seconds to annotate a given form tree. Compared to an existing linguistic-based annotation solution, the proposed approach achieves 43% improvement in terms of precision, and 29% improvement in terms of recall.

- We devise 3 versions of the validation algorithm leading to up to 18 situations with different databases. The algorithm reduces the user interventions by an average 61% for all the situations. The 3 versions required 10, 8, and 13 interventions per form, respectively, in discovering and validating the correspondences.

- On being compared with the expert-designed databases corresponding to 3 datasets, the birthing solution produces databases that are 84.5% identical or superior to the expert-designed ones, wherein, superiority is defined in terms of the compliance with the quality and the optimization principles. The merging algorithm preferred compactness over optimization, and merged the semantically matching elements, in at least 70% of the merging scenarios, in 11 out of the 18 cases. The final version of the framework helped the merging algorithm in ensuring the compactness of the evolved database in 74% of the merging scenarios.

Overall, we draw the following key implications for further research.

- In order to achieve a high annotation performance, it is desirable to design hybrid annotation methods that leverage both the semantic structure as well as the linguistic properties of the form elements.

- The use of annotation has a far-reaching impact on the quality, in particular, the compactness, of the resultant database. The use of annotation led to at least 19% improvement in identification of merging situations and at least 13% improvement in compactness. This however came at the cost of increasing the number of required user interventions.

- Both the validation and birthing algorithms could be improved by further expanding the observation set of form patterns and mapping scenarios. This is likely to further improve the database compactness, and minimize user interventions.

The proposed approaches do contain certain limitations. Both the proposed form understanding and the form annotation approaches are based on supervised learning techniques, i.e., require manual tagging of forms for training the employed machine learning models. Also, other machine learning models, such as Support Vector machines, Classification Association rules, that could also be used for designing the algorithms have not been tested in this study. The evaluation is based on the assumption that the user-supplied correspondences are 100% correct, which may be far from reality. The gold results used for evaluating the experiment results are only an approximation of the ideal results. Moreover, the compliance of the birthing algorithm with the design principles is yet to be theoretically verified.

Chapter 5: Thesis Organization

The rest of the thesis is organized into the following parts.

- Part II presents the literature review. Chapter 6 reviews the literature related to understanding the form semantics. Chapter 7 reviews the literature related to form-driven database design. Chapter 8 reviews the literature related to database integration.

- Part III presents our solutions for the framework proposed in this dissertation. Chapter 9 describes the overall approach. Chapter 10 presents the proposed form understanding techniques. Chapter 11 presents the mapping discovery and validation approach. Chapter 12 presents the algorithms proposed for database design and evolution.

- Part IV presents the evaluation of the proposed framework in the healthcare domain. Chapter 13 describes the objectives of the evaluation. Chapter 14 describes the data and gold standards. Chapter 15 describes the experiment prototype and settings. Chapter 16 presents the experimental design, and Chapter 17 presents the experimental results and findings.

- Part V presents the final remarks on the dissertation study. Chapter 18 describes the contributions and the thesis conclusions. Chapter 19 describes the limitations, and Chapter 20 presents the future work.

Part II

Literature Review

Chapter 6: Review: Understanding Form Semantics

Forms are commonplace objects[14], and are organized based on longstanding conventions which are already encoded within users[15]. Therefore, forms make an excellent communication medium for non-technical users. A form template provides a simple abstraction over the relational database schema or an entity relationship model, which are otherwise difficult to comprehend by a user.

There are two kinds of forms: data-entry forms and search forms. Both reflect a portion of the underlying database. While search forms help in "determining" the underlying database[39], data-entry forms might help in "designing" a prospective database[16]. Form understanding is not a new problem, and has been studied in the past in the context of search forms. We conducted a survey[40] on various search form understanding techniques. Form understanding is a multi-staged process that involves modeling, parsing, segmentation, and segment processing. the following sections present the survey results in the light of the various stages of the form understanding process.

6.1 Modeling

In this stage, a form is modeled into a formal structure suitable for machine processing. This stage was studied under the two dimensions: information on implied queries, and information on constraints.

A form contains multiple segments, each corresponding to an implied query. The surveyed works use a variety of segment labels to refer to a segment. The segment label adopted by LITE[41] and LEX[42] is "logical attribute." These works model a form as a list of queries, each specific to an underlying database table attribute. The segment contents for LITE include a form element, and a text-label. LEX models a segment to have a text-label, multiple form elements, and an optional text-label associated with each form element. It assigns the semantic labels "attribute-label," "domain/constraint element," and "element label," respectively, to these components. The work on Hidden Syntax Parser (HSP)[43] adopts "conditional pattern" as the segment label. Each conditional

pattern represents a specific query capability of the underlying database. A conditional pattern consists of a text with a semantic label "attribute name," a form element with label "operator," and a form element with label "value." The model adopted by Khare and An[44] is also similar in that it represents a form as a sequence of segments, and uses "attribute-name," "operator," and "operand," as the semantic labels. In addition to modeling segments corresponding to implied queries, Benslimane et al.[39] also create groups of segments, known as the"structural units." Each structural unit corresponds to a logical entity in the database schema. Certain works do not explicitly assign any labels to segment or segment components, but do mention the segment contents. Kaljuvee et al.[45], LabelEx[46], and DEQUE[47] model a segment to consist of a text-label and one or more form elements. Dragut et al.[48] and ExQ[49] present a novel way of modeling a form as a tree structure having arbitrary number of levels. Both these works create groups and sub-groups of related form elements and text-labels, and hypothesize a hierarchical structure. Each internal node of the tree represents a text-label and has a group of related form elements as its descendants. Hereafter, the works by Kaljuvee et al.[45], Benslimane et al.[39], Khare and An[44], and Dragut et al.[48], are referred to as CombMatch, FormModel, HMM, and SchemaTree, respectively.

In terms of constraints, HSP and LabelEx model a form element to have a domain of values. DEQUE models a form element to have domain, and invisible and visible values. LEX models a segment to have a domain type, and a default value. It models a form element to have domain type (finite, infinite, Boolean), and a unit ($, grams, days, seconds). FormModel includes the relationship among structural units, constraints, and the underlying source information. HMM models miscellaneous texts which might include information on constraints.

6.2 Parsing

Parsing marks the beginning of automatic processing, and brings the form into a workable physical structure. While the modeling stage provides a logical image to a form, the parsing stage physically reads the form components. Parsing strategies were studied under the following dimensions: input mode, description, and purgation. The input to the parsing stage can be in two modes: HTML source code of a form, and its visual counterpart, i.e., a form as viewed on a Web browser. CombMatch,

LEX, FormModel, and HMM use HTML code as the primary input. Along with HTML code, LabelEx, LITE, HSP, DEQUE, ExQ, and SchemaTree use layout engines to extract the visual features, such as pixel distances between components.

Description refers to the tasks performed while parsing a form. LITE parses a form in the "Pruning" stage wherein the components that directly affect the layout and labels of form elements are isolated from the rest. CombMatch, in its "Chunk Partitioning" stage, segments an interface into chunks delimited by HTML and TABLE cell tags. LEX develops an "interface expression" that looks like 'teeeteeetteeeet.' HSP parses a page into a set of tokens using its module, "Tokenizer," and stores information such as name, layout position, etc. HMM creates a DOM tree of form components and traverses the tree in depth-first order. SchemaTree, in its "Token Extraction" module, creates lists of text tokens, field tokens, and image tokens, and also stores the information about their bounding boxes. Purgation denotes the components that are removed while parsing to avoid information overload on subsequent stages. LITE discards images and text styling information. FormModel and CombMatch remove stop words and text formatting tags. DEQUE ignores the components that correspond to font size, typefaces, and styling information. HMM ignores all the components except the form elements and the text-labels.

6.3 Segmentation

After a suitable logical representation and a physical structure are accomplished, the form is segmented, i.e., the information regarding the implied queries is extracted from the form. Figure 6.1 shows a segmented form having 2 queries.

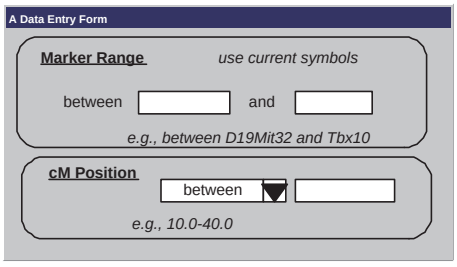


Figure 6.1: Segmented Search Form

Segmentation can be visualized as a 3-task process. The first task, text-label assignment, involves associating a form element with a surrounding text-label. The second task is grouping where the related form components are grouped together to form a segment. In the third task, semantic labeling, labels or query roles are assigned to individual components of a query. Automatic text-label assignment and grouping are difficult due to diversity in Web design. Automatic semantic labeling is difficult as Web designers usually do not assign explicit labels in the HTML source code. A majority of the works (LITE, CombMatch, DEQUE, LabelEx) only address the text-label assignment problem. LEX groups related text-labels and form elements together into "logical attributes." HSP finds groups of "conditional patterns." LEX, HSP, and HMM, perform grouping as well as semantic labeling. LEX also identifies the "exclusive attributes" on a form based on a domain-specific vocabulary. SchemaTree performs text-label assignment and creates segments and sub-segments resulting into a tree of form tokens. ExQ extracts the grouping information of a form into an unlabeled tree structure and then performs text-label assignment to generate a labeled tree.

Segmentation techniques, i.e., the mechanisms to segment a form, belong to 3 categories: heuristics, rules, and machine learning. Heuristic-properties are of 3 kinds: textual, styling and layout. Textual properties include text length, no. of words, string similarity, element's HTML name, etc. Styling properties include font size, font type, form element format, etc. Layout properties include position of a component, distance between two components, etc. To perform text-label assignment LITE exploits all 3 kinds of heuristics. CombMatch uses a combination of 8 different algorithms leveraging the 3 kinds of heuristics to assign text-label to a form element. DEQUE and LEX perform text-label assignment based on the textual and layout properties of components. In LEX, all the form elements associated with same text and the text itself are assigned to one segment. Based on heuristics, it also assigns the semantic labels, "attribute label," "constraint element," "domain element," and "element label" to the components.

A rule is a formalized heuristic. Rule-based techniques rely on regular expressions, grammar, or finite state methods; and create rules for associating a form element with a surrounding text. HSP assumes that a hidden syntax guides the presentation of form components on a form template. The

identification of segments and semantic labels is performed using a grammar. The grammar rules are based on layout properties and are derived using pre-studied examples. SchemaTree uses both rules and heuristics. A tree of fields is built based on the layout properties of form elements, and a tree of text tokens is built based on the layout and styling properties of the text-labels. Then, the two trees are integrated based on some common-sense rules, to generate a complete schema tree corresponding to the form. Recent years have seen an advent of machine learning techniques in the field of form understanding. LabelEx employs supervised machine learning to assign labels to form elements. It designs a "Classifier Ensemble" using Naive Bayes and Decision Trees classifiers, and employs both textual and layout properties to perform text-label assignment. HMM explores another machine learning technique, Hidden Markov Models. It creates a 2-layered artificial designer having the ability to understand a form based on the layout and textual properties of components. The first layer tags the components with semantic labels, and the second layer identifies the boundaries of segments. ExQ creates the form structure tree using hierarchical agglomerative spatial clustering. Each form element is considered to be a visual attribute block. To generate the tree, spatially closer and similarly styled blocks are clustered under the same internal node. ExQ performs node label assignment using annotation rules, and hence falls under a hybrid category.

6.4 Segment Processing

After the form is segmented, more semantics related to segments and segment components are extracted. This includes the information related to data and integrity constraints of the underlying database. While several approaches enlist this information in the modeling stage, very few actual extract it in the subsequent stages. These approaches were studied under the dimensions: techniques, and post-processing. LEX uses machine learning classifiers to identify more semantics from a segment, such as type, domain type, value type, unit of form elements, relationship and semantics of domain elements, and logic relationship of attributes. FormModel uses another machine learning technique, learning by example, to extract relationship between two "structural units," and constraints of a form instance. LITE and LEX post-process the text-labels by removing stop words such as "the," "any," etc. LITE also performs standard IR-style stemming on the text-labels. HSP's "Merger" module

reports conflicting tokens that occur in more than one query conditions, and missing tokens that they do not occur in any query condition. LabelEx devises heuristics for reconciliation of multiple labels assigned to an element and for handling form elements with unassigned labels.

6.5 Evaluation

Though evaluation is not a part of the core form understanding process, it acts as an after-stage in all surveyed approaches. Herein, the semantic information extracted by an approach is evaluated by comparing with either the manually extracted information, or a gold standard as in the cases of SchemaTree, LabelEx, and ExQ. The surveyed approaches are tested on several domains. The most popular choices of researchers are automobile, airfare, books, movies and real estate, followed by car rental, hotel, music, and jobs. Some of the least tested domains include biology, database technology, electronics, games, health, medical, references and education, scientific publication, semiconductors, shopping, toys, and watches. We have published a list of various datasets in an on line directory [50].

LITE, HMM, and LEX report the extraction accuracy, i.e., the number of correctly identified components (segments) over the total number of manually identified components (segments). DEQUE reports the label extraction accuracy and the domain value extraction accuracy. CombMatch reports the success percentage, i.e., the number of correctly identified text-labels over the total number of elements, and the failure percentage, i.e., the number of incorrectly identified text-labels over the total number of elements. HSP reports precision and recall, wherein precision is the number of correctly identified segments over the total number of identified segments, and recall is the number of correctly identified segments over the total number of manually identified segments. LabelEx reports recall, precision, and F-measure. SchemaTree measures text-label assignment accuracy, and the overall precision, recall and F-score. ExQ measures precision and recall for grouping, ordering, and node labeling.

Most of the surveyed works evaluate the performance by comparing their results with those of one or more of the contemporary works. HSP and LEX are the most widely used benchmarks for performance evaluation. HSP was chosen by LEX, LabelEx, and SchemaTree, to compare the performances of respective works; and LEX was chosen by LabelEx, SchemaTree, and HMM. Another

benchmark work is CombMatch, chosen by LITE.

6.6 Review Conclusion

To summarize the review of various form understanding strategies, we plot the works into a two dimensional graph as shown in the Figure 6.2. The two axes corresponds to the two dimensions: database description, and extraction technique.

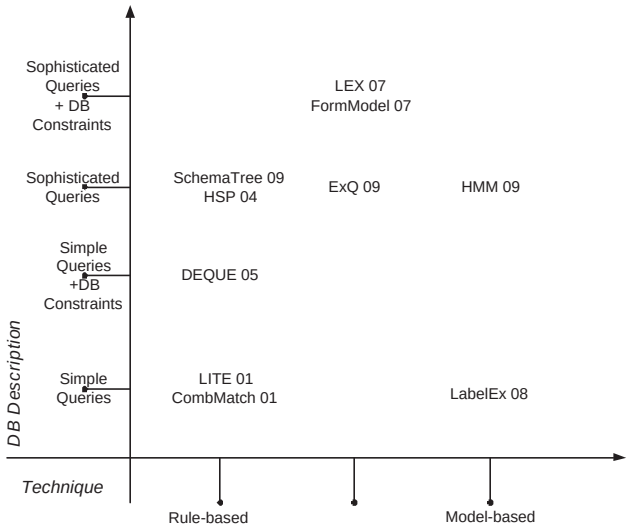


Figure 6.2: 2-D Representation of Form Understanding Approaches

Database Description: This dimension is described along the y-axis, and denotes the underlying database information extracted by a given approach. The surveyed approaches can be organized into 4 levels. The first level consists of LITE, CombMatch, and LabelEx. These works extract simple queries by performing text-label assignment. Figure 6.3a shows an example of a simple query extractable by associating "Gene ID:" with the adjoining textbox. This corresponds to the clause, "WHERE GeneID = 'PF11_0344'." However, text-label assignment at times results in extraction of partial query capabilities when it faces sophisticated designs like the one shown in Figure 6.3b. Such works might assign both textboxes to the text-label "Enter the length ...," but would fail to extract the complete implied query that corresponds to the clause, "WHERE length>=0 AND length <=12." At the next level lies the work DEQUE. This approach extracts simple query capabilities along with data and integrity constraints of the underlying database. The next level includes the

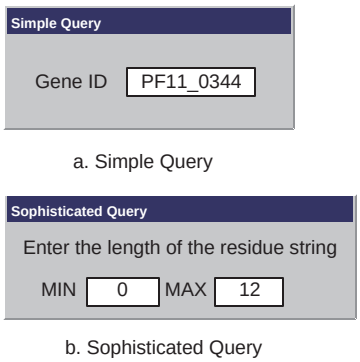


Figure 6.3: Simple and Sophisticated Queries

works that extract sophisticated queries, like the one in Figure 6.3b. HSP, LEX, and HMM identify such queries by grouping all related components into segments corresponding to logical attributes. FormModel forms a different type of segment that refers to an entity, "structural unit," instead of an attribute. SchemaTree and ExQ are different too in that they perform hierarchical grouping and the queries extracted might be associated with both attributes and entities. Both LEX and FormModel employ strategies for extracting data and integrity constraints too, and thus, occur at the highest level.

Extraction Technique: This dimension refers to the techniques employed during the stages, segmentation and segment processing. These techniques fall under two categories: rules and models. We blend rules and heuristics into the rule-based category, and supervised and unsupervised machine learning into the model-based category. HSP, LITE, CombMatch, DEQUE and SchemaTree represent the rule-based approaches. LabelEx and HMM are both model-based. LEX and FormModel lie in between the two categories because they extract implied queries using rules, and extract constraint information using models. ExQ too lies in between as it performs grouping using a clustering model and performs text-label assignment using rules.

The two dimensional holistic analysis reveals two striking points regarding the journey of form understanding in the past decade. First, a considerable progress has been made by the form understanding approaches in terms of the underlying database information extracted. This is depicted by the transition from simple to sophisticated query capabilities. Second, a considerable progress

has been made in terms of the improvement in the sophistication level of segmentation from rule-based to model-based techniques. In this thesis, we seek inspiration from various aforementioned approaches for search forms and tailor them to understand a data-entry form. In particular, our approach is inspired by the hierarchical modeling of forms that leads to a richer extraction of the semantic information associated with underlying databases. We model the form as a tree structure and use a mix of rules and machine learning techniques to automatically derive the semantic tree structure corresponding to a given form. While search forms provide a useful way in *determining* the underlying database[39], in this work we emphasize that data-entry forms provide key guidelines in *designing* a prospective database[16] as discussed in the next part.

Chapter 7: Review: Form-driven Database Design

This part of the literature review is related to the generation of a new database corresponding to a user-designed form. Form-driven database design is based on the premise that important information on databases could be retrieved by analyzing forms[14]. The earlier approaches[14;51] aimed toward assisting the database experts, i.e. toward automation, as user requirements began to surpass the perception power of the database designers[52]. The later approaches focused on allowing the users, with no technical knowledge, to design databases on their own. The idea is to enable users to design forms on their own by specifying the data collection requirements in a Do-It-Yourself(DIY) manner. The user-designed forms are then translated to databases using some forward engineering mechanism. The challenges associated with enabling non-technical users to develop databases have been referred to as the "birthing challenges"[11]. When users specify their needs as self-designed forms, the underlying database closely reflects the user's perception of data[11]. We now discuss some of the key form-driven database design approaches.

7.1 EDDS

The trend of using user-specified requirements to automatically generate a database began in 1988 when Choobineh et al.[14] proposed a rule-based *Expert Database Design System (EDDS)* that takes a collection of paper form templates as the primary input and generates an Entity Relationship Diagram specifying entities, relationships, attributes, and cardinalities. This work is based on the assumption that forms are superior to natural language in terms of formalism and structure. For each form, user specifies the title, the captions, the entries, and the source and sink of fields. Each template is then manually processed to derive the hierarchical field structure. The system first determines the order in which the input set of forms has to be processed, and then for each form considers the complex mappings between form and database. This is the first work to explicate the knowledge of form mappings. The system is semi-automated in that an expert designer's intervention

is needed while identifying entities and relationships.

7.2 FOBFUDD

An extension of EDDS is the *Form-based Functional Dependency Deducer FOBFUDD* system[53]. Along with the form schema, FOBFUDD also involves some examples of form instances provided by end-users. This helps in detecting functional dependencies and hence the relational schemas, as opposed to the ER model. FOBFUDD encodes a set of 90 rules to determine various single-attribute and multi-attribute determinant functional dependencies from a given set of optimally sequenced forms. To provide an account on the "correctness" of the deduced dependencies, each rule is associated with a certainty factor.

7.3 IIS*Case

The trend of fully automated approaches was marked by the design of IIS*Case[51], a CASE tool that accepts user-specified requirements as the input and generates a corresponding database schema at the back-end. User requirements are gathered as a *Form Type*, which is a modified version of a form template. Any standard form template can be expressed in terms of form type. Form type is a tree structure with component types as its nodes. Each component type has a name, and a set of attributes with associated domains and constraints (key, unique and tuple). Thus, a user indirectly specifies the relations, attributes, and constraints of the back-end database. The key difference between IIS*Case and[14] is that in IIS*Case, the design load is transfered from the designer to the non-expert user in order to generate a fully automated process. The concepts such as component type, form type, and constraints, might be difficult to understand for end-users.

7.4 Zohocreator

As the awareness of database usability increased, there were approaches aiming toward the do-it-yourselfer users. The new millennium brought with it several approaches that allow users to self-design data-driven applications. ZohoCreator[54] is a tool that allows users to design forms and hence databases, and collect data in the database. Using ZohoCreator, a user can build applications

by designing form templates. Each form template is a flat set of attributes. Each attribute can have one of the multiple available formats such as single line multi line, email, date, checkbox, etc. ZohoCreator presents these user-designed forms as tables. Users can easily gather data into custom-designed database by adding new rows to tables.

7.5 Deklarit

Deklarit[55] is a model-driven tool for application design provided by the Microsoft Visual Studio. This tool is intended for users with no background in databases and modeling. Users need to specify *business components* containing attributes and key constraints. User specifies business components with a name and a list of attributes and data types. A business component may contain another component giving rise to a hierarchical unnormalized structure. The relationships among components are inferred using universal relationship naming convention and a database schema with key and referential integrity constraints is created accordingly.

7.6 InfoPath

Among the DIY tools there is a WYSIWYG class of approaches which allow users to design applications and keep track of the design while they are creating artifacts such as forms and views. Microsoft released InfoPath[56] in 2007 which allows users to design simple and sophisticated form templates and collect data by filling out these forms. InfoPath is based on XML technology. The form templates get converted to XML Schemas and XSL transformation files, and the data gets collected into associated XML files.

7.7 REDCap

Research Electronic Data Capture (REDCap)[57] is yet another tool that provides informatics support for storing data associated with clinical translational research. The on line form editor allows users to create forms and fields in real time and stores the data in the Entity-attribute-value(EAV) format. This tool however is used in collaboration with the information specialists since it is not as intuitive to be directly used by the non-technical clinical researchers.

7.8 FormAssembly, etc.

Another popular tool is FormAssembly[17]. It provides the users with a simple yet powerful way of designing data-driven applications. FormAssembly enables users to design data-entry forms, and the form is then translated to a back-end database. While designing a form, the user is asked to enter the questions to appear on the form and respective formats (textbox, checkbox, etc.). The questions could also be arbitrarily grouped under sections and subsections. As the user designs a form, she is presented with a tree structure of the questions and sections for a better understanding of form structure. FormAssembly helps in designing complex forms by providing many features such as numerous formats and calculated fields. Finally, it allows users to collect data into custom-designed applications by filling out custom-designed forms. There is another class of WYSIWYG tools, PerfectForms[58], Wufoo[20], AppNowGo[59], and JotForm[19] that allows users to drag and drop various form components on a visual grid, and make it even easier for non-technical users to design forms. Users can collect data into the back-end database using these forms. Google Forms[60], is yet another application that allows users to design form by dragging and dropping components. The data collected on these forms gets stored into an MS Excel Spreadsheet as opposed to a database. Another work is the form designer tool of the OpenMRS system[61] that allows users to design clinical encounter forms in a WYSIWYG manner.

7.9 Review Conclusion

From the usability point of view, certain works[14;51;53;55] require users to learn technical jargon related to databases or data modeling, and are largely manual. In particular, these approaches were designed for IT professionals to develop databases using "form structures"; users have to specify the exact semantics of the underlying database schemas through forms. These are less likely to be usable for users with no background in databases, e.g., the clinicians. In contrast, the WYSIWYG tools[17;19;20;54;58;60] are a major leap toward database usability. However, in our work, we are interested in automatically deriving a database from an arbitrarily designed form, not necessarily from a DIY tool that on-the-fly captures the form semantics and the relationships among

various components. We intend to expand the range of the user input from a DIY tool designed form to an externally developed form. Also, we seek a framework that designs databases with certain desirable properties with respect to the form semantics. The mapping process for most of the existing works[13;14;52;53] is guided by data-modeling specific principles e.g., consistency, and expressiveness. These principles do not reflect the contribution of the requirements, and are thus inadequate for evaluating the mapping process. Ignoring quality has many unintended consequences such as logical inconsistency and update anomaly. Also, these works do not provide any empirical evaluation of the resultant databases questioning their applicability into complex domains. Although this dissertation seeks inspiration from the existing works on form-driven database design, we develop novel methodologies for understanding an arbitrarily designed form, and for developing a novel algorithm to design a high-quality relational database with respect to a given form.

Chapter 8: Review: Toward Database Integration

The DIY form-driven database design methodologies such as FormAssembly[17] are a major step toward database usability. Although these tools hide the underlying data storage details from the users, there is a major shortcoming. Each form is stored individually without semantic integration among the forms. When a new form is designed which is conceptually overlapping with a preexisting form, the existing tools do not merge the form with the existing database, i.e., the two forms do not get connected through the database structure. There are very few existing works that focus on the problem of automatically integrating a new form into an existing database while appropriately merging the overlapping database elements. We discuss the related work in this chapter.

8.1 Integration of New Forms

Lukovic et al.[52] extend their previous work[51] by providing a semi-automated approach to integrate the external form types, i.e., new user requirements, into the existing database. This approach functions at the implementation level, and is hence suitable for detecting collisions and performing integration. Each application containing one or more form types is converted to a subschema, and is consolidated into the existing application's schema. The consolidation takes place while sequentially ensuring consistency of attributes, key constraints, unique constraints, null value constraints, and referential integrity constraints. While collision detection is automated, collision resolution requires designer's intervention. This makes the approach semi-automated.

Appforge[13] is yet another work that provides an application building tool to non-technical users. Appforge describes the translation steps from user-specified actions into a sophisticated entity relationship model with multi-way relationships and aggregations. The user works with concepts like role, page, view, form, and container. The user creates forms wherein each form contains a flat list of fields. The system translates each form to a separate entity, with attributes as the fields of the form. The user can also design views (or nested views) and add new columns (of various kinds

including entity type) to existing views using the schema navigation menu, which is a hierarchical tree structured menu. This helps in creating relationships and relationship attributes among the entities of the back-end model. Using Appforge users may automatically extend an existing application by creating new forms and adding new columns to existing views. A significant aspect of this work is that they conduct usability study with 6 users, including 2 researchers who were database experts with advanced degree in computer science, 2 researchers who were non-database experts with advanced degree in computer science, 1 managerial position holder trained in computer science, and 1 recruiter familiar with using database applications. Appforge was easily understood and used by the users with advanced computer science degrees. The other 2 users were, however, very challenged while understanding and using Appforge. The interface was hence re-designed based on the difficulties and confusion faced by them. While Appforge is a promising application, it has not yet been tested on non-computer skilled users. Also, using the schema navigation menu for evolving a large scale database might impose additional visual and cognitive burden on users, as a menu corresponding to a large scale schema would not fit into a single screen.

8.1.1 Review Conclusion

In this dissertation, we are interested in developing highly automated strategies for integrating a form into an existing database. The existing works in this direction are largely manual and expose the users to the technical details of the underlying data model. This creates a friction between the non-technical users and their ability to evolve the existing database as per their changing needs. Another issue is that several integration issues occur due to a variety of terms used by different users to describe the same semantic concepts. In this dissertation, we also investigate whether standardization of terms can resolve certain integration challenges, and facilitate smooth merging of forms into databases. We review the related literature, in the context of the healthcare domain, in the next section.

8.2 Standardization of Terms

Standardization of clinical data has received a lot of attention in the past. The primary motivation of translating data into standard concepts is to resolve interpretation issues, facilitate clinical and outcomes research, and support future interoperability across systems and institutions. In plus, the research conducted in this area also highlights the usability of the carefully developed medial standards, and implies certain guidelines for refining the standards. In this section, we discuss some of the key works related to mapping freely written clinical data into standards such as SNOMED CT.

Henry et al.[62] investigate whether the narrative description spontaneously entered by nurses, in patient's progress notes and care plans, could be represented by the SNOMED-III terminology. For this investigation, 485 patient encounters are collected from 3 institutions, and the terms describing patient problems are manually extracted. The data to be mapped includes 1841 patient problems composed out of 761 unique terms. The problems are mapped to SNOMED concepts using exact string matching. Overall, 44% of the problems map to a single concept, and 69% map to one or more concepts and allowing the user to choose one. Although the results are not expert validated, it is concluded that it is possible to represent the nursing terms using standard terminology.

Another study[63] performs a mapping between the clinical terms used by the practitioners and an expert designed standard. This study is conducted at the Columbia Presbyterian Medical Center. The data to be mapped consists of the clinical diagnosis and medications information written by practitioners in a clinical profile system. The data terms are either provided by the practitioner, or are chosen from the SNOMED terminology . This data is required to be mapped to a home-grown standard vocabulary, the Medical Entities Dictionary (MED). MED is a semantic network with over 35000 entities, wherein each entity has a name and multiple synonyms. The proposed method creates word groups of the MED entity terms using the lexical variants from the UMLS. Each term from the clinical profile system is tokenized, and matched with the medical entities via the word groups. The possible matches are ranked based on longest common substring similarity with 75% cutoff, and presented to the user. Mapping the 1045 SNOMED-derived terms to entities leads to

a recall of 70% and a precision of 61%. Out of the 1225 practitioner supplied terms, 31% map to exactly one entity, and 51% map to at least 1 entity. The results from this dataset are, however, not evaluated.

The work *TokenMatcher*[64] also maps clinical notes containing medical complaints into SNOMED CT concepts. The algorithm pre-processes the notes using sentence boundary detection, term normalization, and POS tagging. A regular expression based entity recognizer is used to extract the relevant terms to be mapped. The algorithm also utilizes an augmented lexicon that consists of a general word to concept mappings derived from the SNOMED CT description table. The key is the token matching step of the algorithm that matches each clinical term to concepts using the augmented lexicon, and assigns a score to each concept description. The algorithm also employs abbreviation expansion using a list of 1254 medical abbreviations. It also performs negation identification by using some rules to identify the post coordinated concepts. The *TokenMatcher* has been developed as a web service but no formal evaluation has yet been conducted.

The above mentioned works address the problem of standardizing the clinical notes written for human processing and understanding. In contrast, the work *Model Standardization using Terminology Services,(MoST)*[65] presents a method to map a clinical data model into SNOMED CT concepts. The model considered by *MoST* is the European standard clinical model, known as Archetypes, which is the back bone of the clinical data entry forms. *MoST* is a method to find the candidate SNOMED CT concepts that correspond to the intended meaning of a term used in the data model. The method performs lexical processing of the terms using emergency medical text processing, word sense disambiguation, synonym identification, and term simplification. This is followed by the context processing including identification of the semantic category of the term using UMLS, and mapping the term to concepts using certain filtering rules based on the SNOMED CT categories and relationships. Finally, the modeler is presented with a list of candidate concepts to choose from. The method is tested on 19 models with 475 terms. The precision and recall calculation is relaxed in that any case, where the desired concept is part of the candidate concepts, is considered a success. Overall the method leads to a recall of 89% and a precision of 82%. After applying the context rules,

the precision increases to 90%.

8.2.1 Review Conclusion

To accomplish annotation, most of the existing works rely on the linguistic similarity techniques such as exploration of synonyms, morphemes and lexical variants. Such techniques can certainly lead to a large recall. However, the standard vocabularies are growing and getting richer; there are often multiple lexically matching concepts with different semantic intentions, leading to the context disambiguation challenge. It has become increasingly important to accomplish a high precision as well [66].

In this dissertation, we propose that the context-based techniques, when combined with the linguistic techniques, could lead to a higher precision. We propose a method to map a clinical data entry form to SNOMED CT concepts which is based on exploiting the semantic structure of forms. Conceptually, our work is similar to *MoST* in that we perform the mapping of clinical meta data as opposed to data. Technically, our work differs as the contextual information used by *MoST* is limited to the SNOMED CT semantic categories. Our work also relies on the context of the form term. Our work is closer to the clinical section classification method proposed in [67] that assigns standard labels to the sections of clinical notes by exploiting the organizational structure of the clinical documents. While most of the existing works are semi-automated and only present a candidate list of concepts, our work is completely automated and retrieves a unique concept corresponding to a given form term. In addition, we also conduct a real-world case study on the data-entry forms developed in 6 medical institutions, thoroughly evaluate the results, and draw several insights from the mapping results.

Part III

Solutions

Chapter 9: Overall Approach

We now present the overall approach to our solution to the form to database mapping problem. The approach can be summarized in the following manner. The input form is represented as an equivalent semantic form tree using a form understanding algorithm. We adopt a proactive approach to mapping in that we also standardize the form terms using an annotation technique focusing on the healthcare domain. Our solutions to the form understanding and the term annotation algorithms are described in Chapter 10. The generated semantic form tree is then studied with respect to the existing database; and the semantic correspondences between the form tree and the existing database elements are discovered and validated using user interventions and certain validation rules. This part is described in Chapter 11.

The form tree, with discovered correspondences to the existing database elements, is then mapped and merged with the existing database. In particular, the matching elements are merged to the target database elements and the new form elements are transformed into new database elements and the existing database is extended using the new database elements. The database design and evolution algorithms are described in Chapter 12. The approach is illustrated in the Figure 9.1. The technical significance is that the entire approach is designed while considering the goal of evolving a principle-compliant database in a highly automated manner.

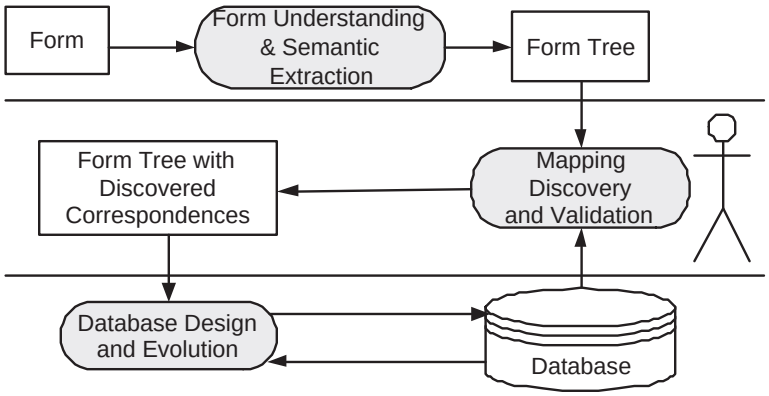


Figure 9.1: Overall Approach

Chapter 10: Form Understanding

This chapter presents our solution to the problem of semantic understanding of forms. First, we present the solution to automatically derive a form tree from an arbitrarily designed data-entry form, and then we present the solution to further refine the semantics of each form term using a standard terminology.

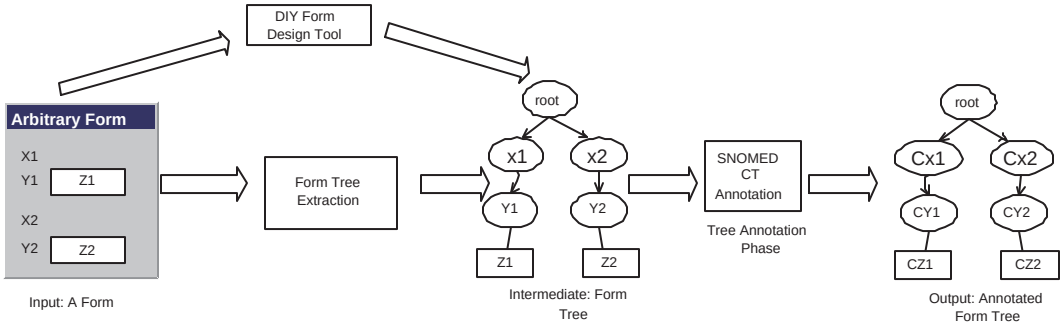


Figure 10.1: Form Understanding and Semantics Extraction Approach

10.1 Form Tree Generation

Data-entry Forms are designed primarily for data collection. Figure 10.2 shows an example form. As defined in Definition 3.1.1, a form is a logically organized collection of form elements where each element is either a `text-label`, e.g., *Name*, *Date*, or a `form-input` such as textbox, textarea, radiobutton, checkbox, etc. What is noteworthy is that a form is not just a thoughtful arrangement of elements to facilitate data-entry; it also reflects the designer's view of the semantic associations among the elements, e.g., the parent-child associations such as *FOR THE PATIENT-High Blood Pressure*, and sibling associations such as *BP-HR*.

A form could be represented using multiple schemes. The simplest is the source code itself, which is an ordered sequence of the form elements. Following this convention, the form in Figure 10.2 is represented as $< Name, \text{textbox}, Date, \text{textbox}, ... >$. However, such a flat representation fails to capture the designer's precise intentions, in particular, the semantic associations among the ele-

ments. Another way is to represent the form as a syntactic Document Object Model (DOM) tree[68]. Counter-intuitively, even this representation fails to capture the semantic parent-child associations among the elements. The DOM tree is necessarily a syntactic tree of the formatting elements in a specific language, e.g., HTML tags $< FONT >$, $< PARA >$, etc.; such representations capture no information on the semantic grouping of the form elements.

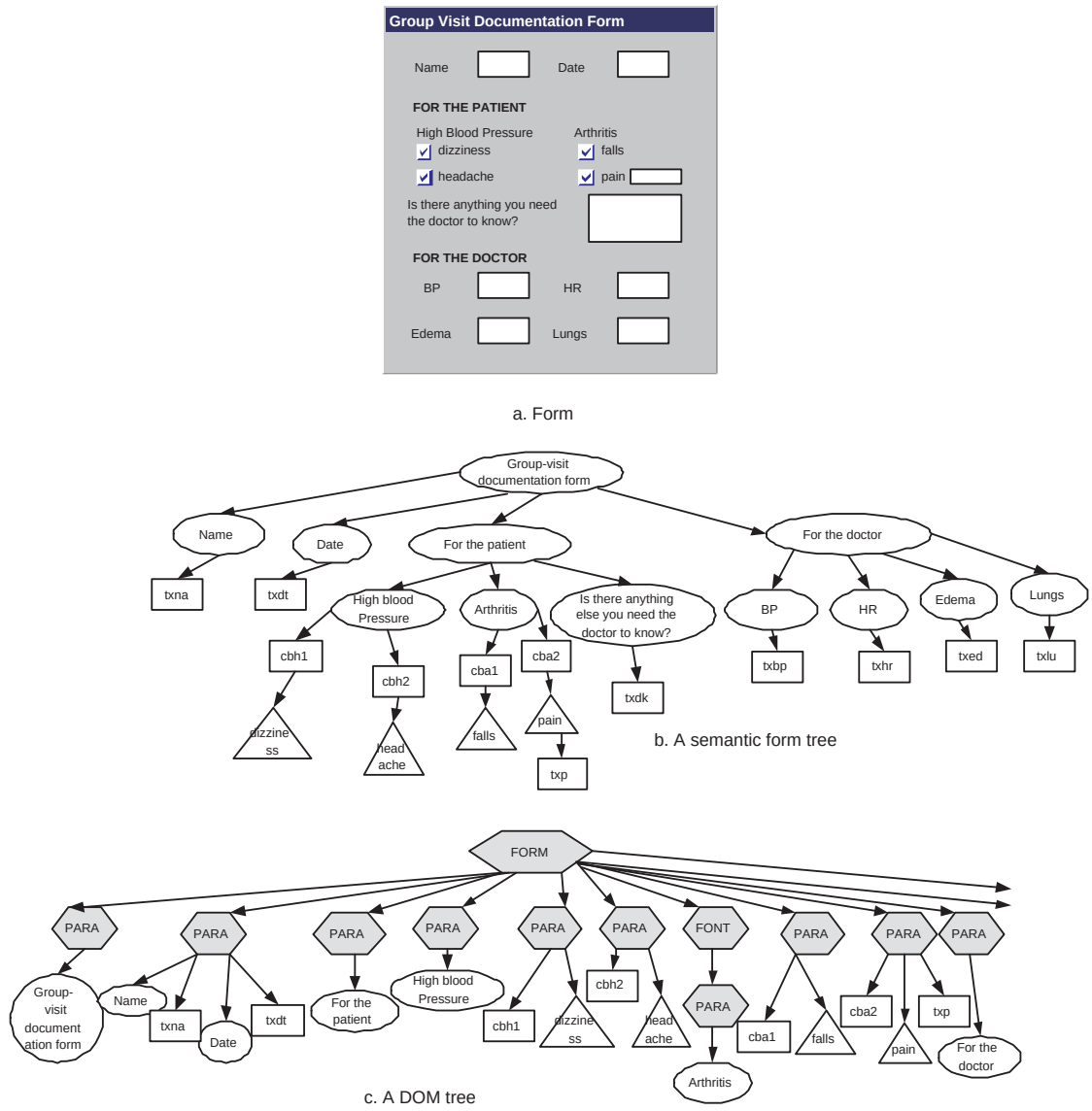


Figure 10.2: DOM Tree Vs Semantic Form Tree

In the context of the semantic mapping problem, we employ a new representation scheme known as the `form tree` that accurately captures the designer's intentions, and hence the semantic asso-

ciations among the form elements. Definition 3.1.2 formally describes the form tree. Figure 10.2 highlights the visual differences between a DOM tree and a semantic form tree. There are a couple of shortcomings of the DOM representation. First, the hierarchical information is lost. As a result, the grouping information, e.g., *High Blood Pressure* and *Arthritis* semantically belong to the same group *For the patient*, is not captured. Second, intuitively, the resultant database would not normalized as we have placed heterogeneous attributes under the same relation. Hence, accurately capturing the hierarchical semantics of a form is also important in generating a high quality database as discussed in the next chapters.

We observe that there are two ways of obtaining a form tree corresponding to a given form. The first way is to capture the user's intentions in real-time, i.e., while the form is being designed using a DIY tool. In this case, the form tree is indirectly specified by the user herself while laying out various elements on the form. Another way is to process an arbitrarily pre-designed form, and extract the form tree based on the information implied by the form elements. Extracting a form tree is challenging since the form source code provides no explicit information on the semantic associations among the elements. We provide our solutions[69;70] to the two approaches of tree extraction in the subsequent subsections. We focus on the healthcare domain while designing the solutions.

10.1.1 On-the-fly Capturing of the Form Tree

We develop a form design interface, i.e., a DIY tool, that enables users, especially, clinicians to design data collection forms on their own based on their data collection needs. In order to facilitate quick and easy specification of needs, this graphical user interface has been kept simple in terms of terminology as well as design. To attain a simple terminology, the interface concepts are represented through regular and intuitive terms. The interface allows clinicians (users) to specify a title for the form, add category, and its fields and also specify the format (textbox, radiobutton, dropdown list, etc.) for each field. Since data-entry forms have a hierarchical structure, we allow a category to contain subcategories with sub-fields as shown in Figure 10.4 where *BP* is a subcategory (contained in the category *Health Status*) with subfields *Systolic* and *Diastolic*. The interface allows a sequential flow of user steps. The form in the Figure 10.3, representing a simple need, can be designed using

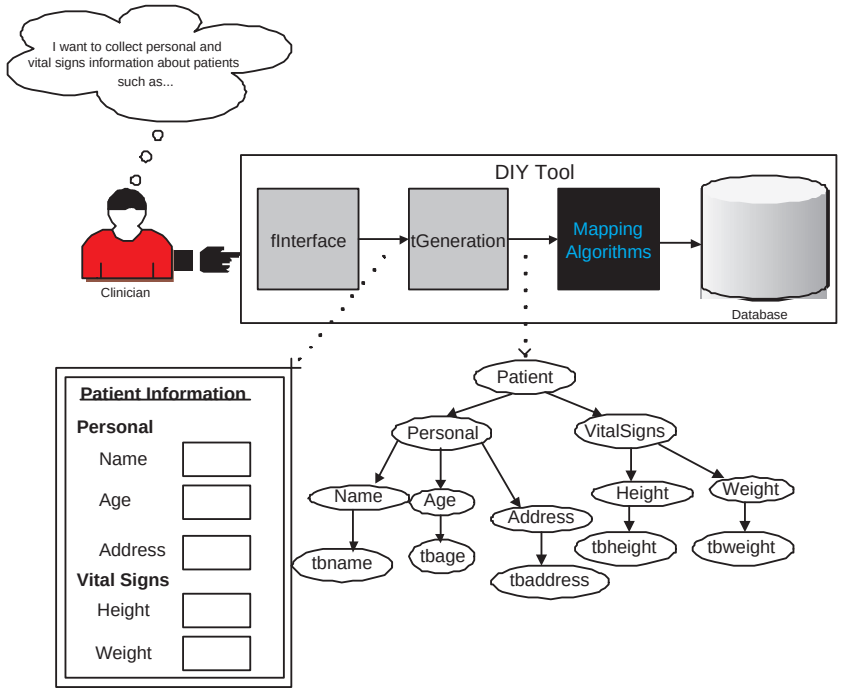


Figure 10.3: The do it yourself tool to design data-entry forms and capture form trees

the following steps (excluding button clicks):

1. Enter the title, *Patient Information*
2. Enter the category, *Personal*.
3. Enter the field, *Name*.
4. Enter the format, textbox for the previous field.
5. Enter the field, *Age* (See Figure 10.5).
6. and so on ...

At each step, the user is presented with a limited number of *concept gateways* associated with certain form concepts. This restricts users to a limited number of possible next steps and thus helps minimize design errors such as specifying a field without any format. Then, based on the gateway entered by the user, a component is added to the form being designed.

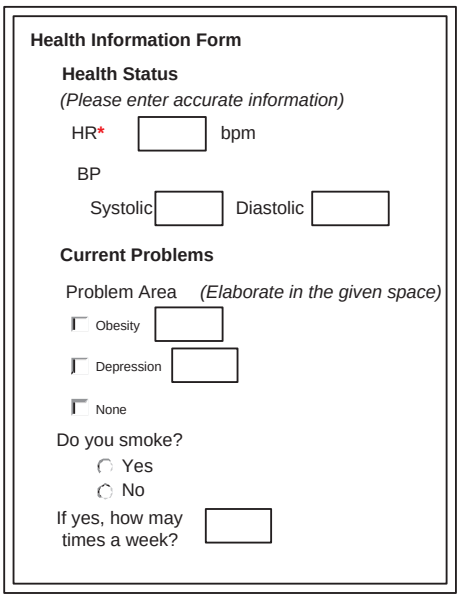


Figure 10.4: A form representing an advanced need. *Please enter accurate ..., Elaborate in a ...* are Supporting Texts, *Obesity* is an extended radiobutton option, *bpm* is a unit, *Do you smoke* and *How many times ...* make a condition

Our next concern was simplicity in design. Simplicity has been considered the first usability principle in context of designing interfaces for the HITs [71]. The idea is to have a minimalistic interface that keeps only those features that are relevant to clinicians, and thus prevent the clinicians from being overwhelmed with feature overload. We analyzed 51 data-entry forms currently being used in healthcare. Table 10.1 shows a list of features found in these forms and their average frequencies. The dataset was collected from the Web and all forms were in *pdf* or *doc* format, suggesting that they were paper-based forms. Given the success of these forms, we believed that clinicians would be already familiar with the features offered by these forms. The fInterface emulates the features of these paper-based forms and hence remains under the boundaries of clinician-friendliness. The interface of our first prototype, fEHRv1, supports all the frequently occurring features (no. 1-6) found in the dataset. It also allows the clinicians to specify up to one level of subcategories within a category as the dataset had at most one level of nesting between categories and subcategories.

The goal of the DIY interface is to induce the data collection needs, captured in a clinician-designed form, into the database. The intermediate step is to generate a form tree corresponding

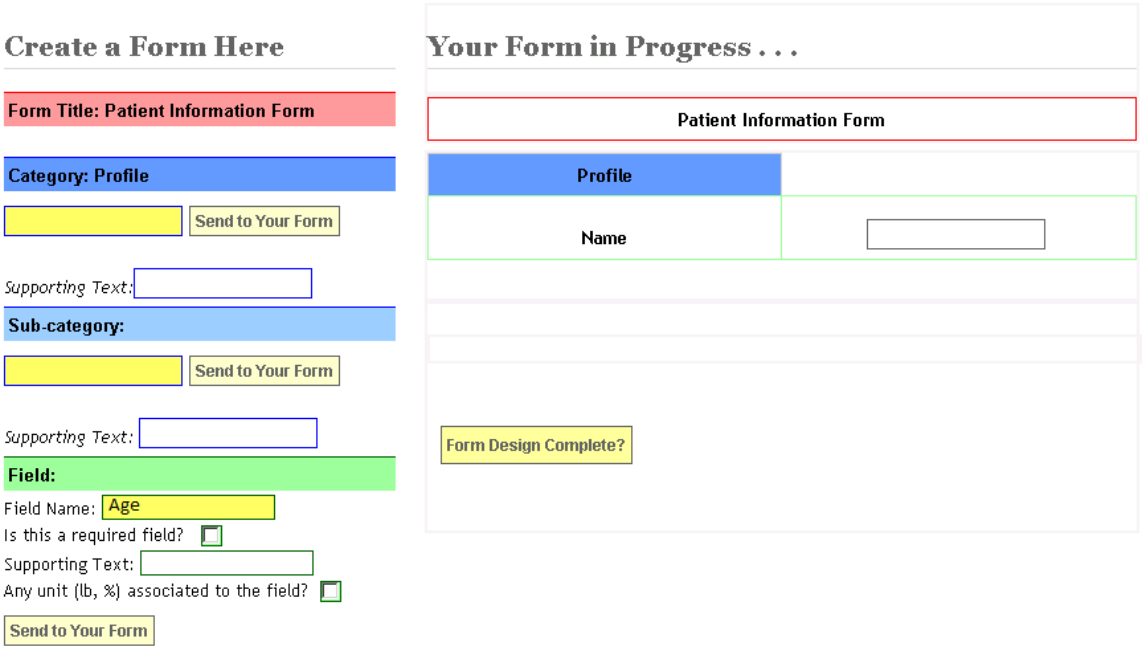


Figure 10.5: A screen-shot of the fInterface at step 5 . The left division is a placeholder for the clinician to enter various form components. There are 3 concept gateways in this case: Category, Sub-category, and Field; and the clinician decides to enter the Field gateway. The right division shows the form being designed

to the user designed form. Figure 10.3 shows a form and its corresponding form tree. The relationships among the form components are maintained through parent-child(category-field, category-subcategory, field-format) or sibling(category-category, field-field) associations in the tree. Some previous studies[48] have also proposed a tree representation for a form. The form tree, used in our work, differs in that it contains the format nodes corresponding to the form inputs such as *tbtime*, *tbsite*, etc. We argue and shortly show that these nodes are equally important in generation of a high-quality database. The tree generation module tGeneration dynamically derives a form tree based on the user actions captured in the tool. Since the nodes and the associations are generated on-the-fly based on the explicitly specified semantic associations, the generated form tree is always accurate with respect to the user-designed form.

Table 10.1: Feature Statistics of 51 Healthcare Data-entry Forms

No.	Feature	Frequency
1	Text Labels (Categories, Sub-categories, and Fields)	60.12
2	Text Inputs (textbox, textarea)	24.23
3	Radiobutton Groups	8.61
4	Checkbox Groups	8.59
5	Drop down lists	6.82
6	Supporting Texts, Units (Fig. 2)	2.70
7	Multi-formats (Fig. 2)	0.39
8	Extended Checkbox/Radiobutton (Fig. 2)	0.11

10.1.2 Automatic Tree Extraction Algorithm

The DIY method of tree generation, discussed in the previous subsection, is tool dependent. In other words, such a method can derive the form trees only when the form is designed using a specific DIY tool. However, in real-world, it is important to be able to process an externally designed form and map it to the existing database for integration purposes. We now present a method of generating a form tree given an arbitrarily designed form. Form understanding is not a new problem [40] and at least two approaches [48;49] have been proposed to automatically derive a tree structure for a given form. However, these approaches cannot be directly applied to the problem of automatically mapping forms to databases, with high effectiveness and efficiency, due to the following reasons:

1. These approaches are composed of rules and heuristics and are thus not likely to circumvent the ever-broadening varieties in form topologies [72]

2. In addition to the HTML code of the forms, these approaches rely on the visual information supplied by rendering engines(such as Gecko, Trident), which makes them browser dependent and inefficient.

3. These approaches are focused on search forms, which are much shorter and hierarchically simpler than the data-entry forms. We assess the length of a form by the number of form elements and the hierarchical complexity by the maximum height of the corresponding form tree. On comparing the characteristics of 50 search forms with 50 data-entry forms in the

healthcare domain, the latter were 10 times longer and twice as hierarchically complex than the former. Moreover, each search form represents a collection of the attributes of a single table in a database, whereas each data-entry form represents a collection of multiple database tables connected through appropriate integrity constraints.

In our solution, we address the above mentioned challenges in the following manner.

1. **Scalability**: We propose a deeper solution to tree generation that takes into account the implicit process of form design to handle a multitude of form topologies. The approach leverages the probabilistic nature of form design and develops a Hidden Markov Model (HMM) based artificial designer that has the ability to understand the semantics of any arbitrarily designed form.

2. **Efficiency**: The approach is solely based on the textual properties of the form elements obtained from the HTML code of the forms and the employed dynamic algorithms are optimized using memoization [73], thus, providing a time-efficient solution.

3. **Effectiveness**: The learning models are tailored for the data-entry forms, and are aligned with the hierarchical complexity of the input forms thereby providing a high extraction accuracy, as per the findings in our previous work on search interface understanding [44].

We accomplish tree generation in two phases. In the $tag - and - segment$ phase, each form element is assigned a semantic tag, and the tagged elements are recursively grouped into segments of arbitrary lengths. In the $tree - derive$ phase, the information from the earlier phase, along with the sequential order of the form elements, is used to derive the tree structure. Figure 10.6 gives an overview of the approach.

Challenges

A data-entry form is composed of two kinds of elements, text/label elements and input elements (radio, checkbox, textbox, etc). Each element has one of the 3 key semantic roles: category, field, format. A category represents the group label of a collection of fields, each field is associated with a format that accepts a user input. A simple example of category, field, and format is the

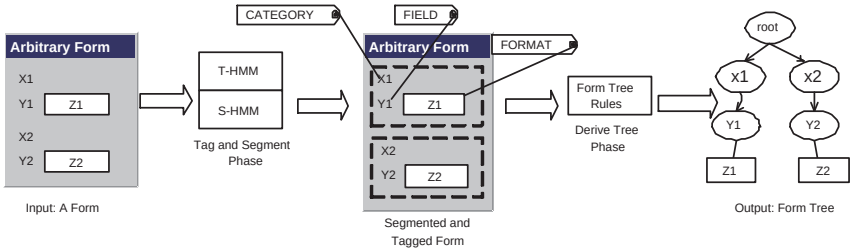


Figure 10.6: Tree Generation Approach

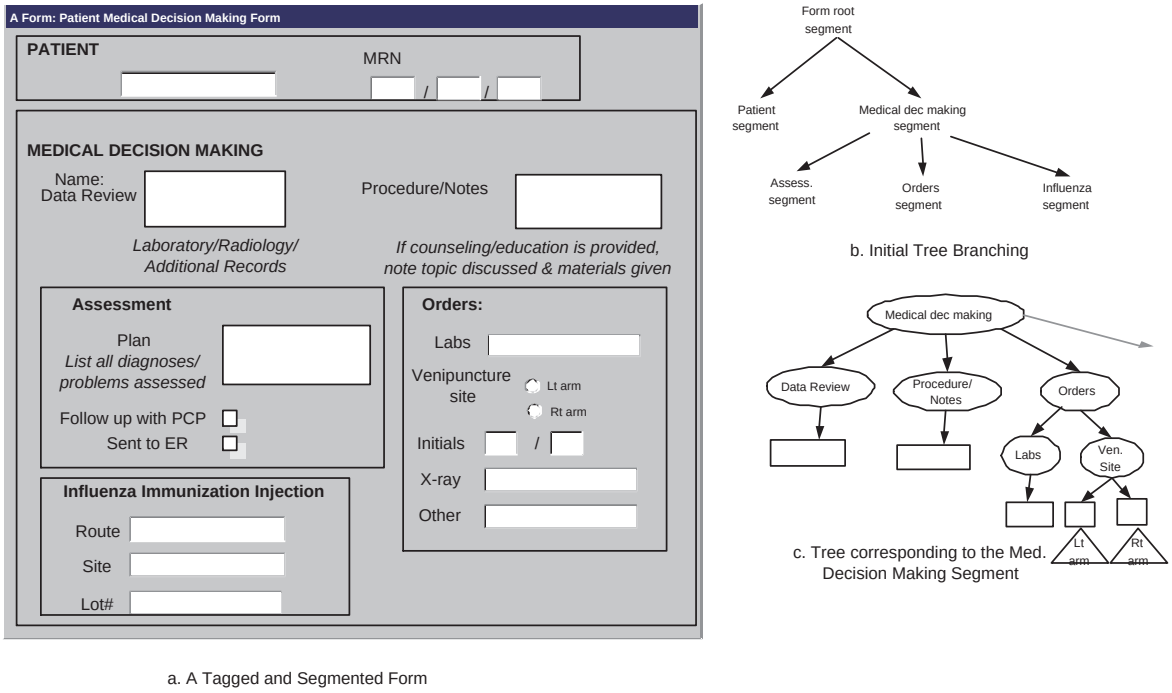


Figure 10.7: Tree Generation Steps

pattern *FOR THE DOCTOR*, *BP*, and *txbp*, respectively in Figure 10.3. Automatic tagging becomes challenging because of the presence of counter-intuitive patterns, the presence of miscellaneous texts (instructions for patients/clinicians, clinical codes, units of user input, etc.) intertwined with other key elements. Segmentation becomes challenging because of the absence of explicit boundaries specifying the semantic scopes of the groups. In addition, the forms often have recursive segments, i.e., segments within segments, that further worsens the problem. Some of these complexities are exemplified in the form in Figure 10.7.

Table 10.2: Observation Space T_HMM

Code	Description
σ_0^T	textbox/textarea
σ_1^T	select group
σ_2^T	radiobutton
σ_3^T	checkelement
σ_4^T	long text(more than 4 words)
σ_5^T	lower case non-colon-ending multi-character text
σ_6^T	colon ending text
σ_7^T	single character text
σ_8^T	uppercase text
σ_9^T	uppercase colon-ending text
σ_{10}^T	parenthesized text

Table 10.3: State Space T_HMM (Also Observation Space S_HMM)

Code	Description
q_0^T	Category Label
q_1^T	Field Label
q_2^T	Format
q_3^T	Subcategory Label
q_4^T	Subfield Label
q_5^T	Subformat
q_6^T	Misc. Text

Phase 1: Tag and Segment

Hidden Markov Models are used to model and understand the behavior of implicit processes. Data-entry form design is one such process. We simulate this design process into an artificial designer using suitable training algorithms. This trained model thus gains the ability to decode, i.e., tag and segment, a given unknown form. We organize the designer into 2 layers[44] in tandem. The first layer T_HMM tags the elements of the forms with their semantic roles, and the second layer S_HMM segments the forms into groups (and sub-groups) of elements. Tables 10.2, 10.3, and 10.4 describe the specification of the two layers.

Table 10.4: State Space S_HMM

Code	Description
q_0^S	Begins a segment
q_1^S	Inside a segment
q_2^S	Begins a subsegment
q_3^S	Inside a subsegment

Phase 2: Derive the Tree

After a form is tagged and segmented, the tree is derived using certain tree design rules. The primary branching of the tree is determined by the segmentation information, and the topology of nodes within a branched segment is determined based on the semantic tags.

The tree branching structure is determined in the following manner. Each segment is represented by a tree node. The root is a container segment that represents the entire form. A sub-segment within a segment becomes the child of the node represented by the segment. Figure 10.7b illustrates the tree branching process. After the initial branching, each segment node is elaborated into a segment tree based on the semantic tags associated with the segment elements using the following rules.

- The category becomes the root of the segment tree.
- A field node becomes the child of the segment root
- The format nodes associated with a given field node become the children of the field node.
- Some format nodes(radio, check, select) may need to be extended to contain the value nodes as their children.
- A subsegment becomes the child of the root of the container segment.

Figure 10.4c shows a portion of the form tree corresponding to a given segment node. Algorithm 10.1.2 summarizes the tree generation process.

ALGORITHM 10.1.2: generateTree($\mathcal{F}$)

Input: a form $\mathcal{F}$, trained models T_HMM, S_HMM

Output: a form tree $\mathcal{FT} = (N, E, <_{sib}, root)$

Steps:

1: **let** TE := executeHMM($\mathcal{F}, T\_HMM$); */* apply the first layer of HMM to generate a sequence of semantic tags corresponding to the sequence form elements */*

2: **let** $SE :=$ executeHMM(TE, S_HMM); */* apply the second layer of HMM to generate the grouping information of the tagged elements */*

3: $\mathcal{FT}$:=deriveTree$(\mathcal{F}, TE, SE)$; */* derive the tree corresponding to the tagged and segmented form using the tree design rules specified in Section 10.1.2 */*

4: **return** $(\mathcal{FT})$;

10.2 Term Annotation with SNOMED CT

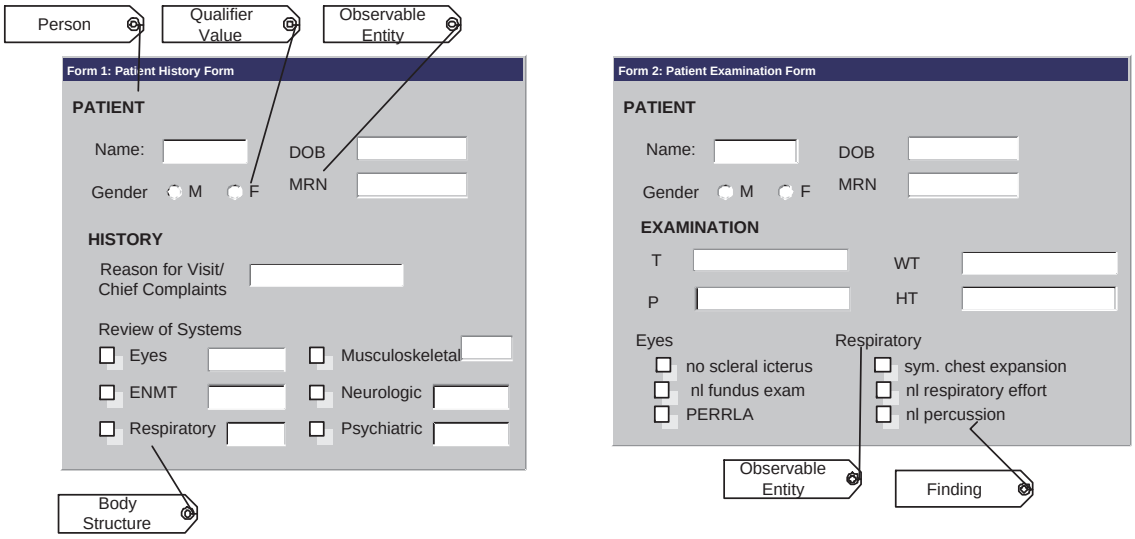


Figure 10.8: User-designed Forms. Tags represent the SNOMED CT semantic categories

Since this thesis focuses on healthcare applications, we customize our solution for the healthcare domain. Semantic heterogeneity across clinical data sources makes database integration and interoperability a huge challenge[25;62;66;74]. Heterogeneity is mainly caused by the diversity of the terms selected by users to design or populate different healthcare databases. To facilitate interoperability across disparate databases, it is important to incorporate controlled clinical terminologies into design artifacts including user interfaces and back-end databases[75;76].

Clinical encounter forms are an important tool in electronic health record (EHR) systems for collecting data into databases. The terms on an encounter form are often specified by the user, and are directly associated with the elements in the underlying database schema and instances. It would greatly reduce the database heterogeneity if the terms on the clinical forms are mapped

to, or annotated by, a standard terminology. Although a knowledge engineer can carefully design encounter forms and databases conforming to a standard terminology, this process is very costly and tedious. Also, there are other cases where either legacy systems need to be mapped to a standard terminology, or the non-technical users, e.g., clinicians, want to specify their own encounter forms. For these cases, it is desirable for an automatic tool to assist users in mapping form terms to standard terminologies. Form term annotation refers to the problem of mapping a form term to a standardized concept.

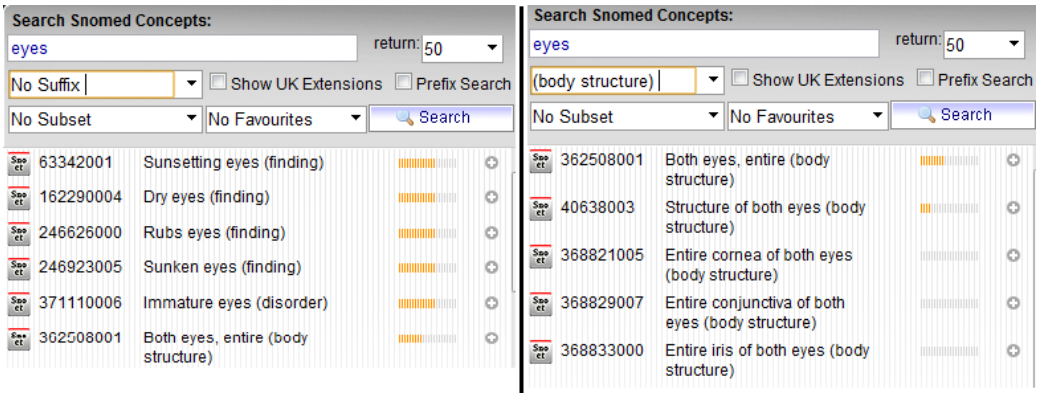


Figure 10.9: Mapping the term "eyes" to SNOMED CT. (a) general mapping (b) category-specific mapping

In this thesis, we study the **problem of mapping terms of clinical encounter forms to SNOMED CT concepts**, and develop a context-based method that leverages the semantic structure of forms to improve the mapping results. The *Systematized Nomenclature of Medicine–Clinical Terms (SNOMED CT)* is a widely used medical terminology. It is comprised of over 360,000 logically-defined clinical *concepts* belonging to various *semantic categories*[77;78]. Each concept is represented using a numeric *concept id* and multiple kinds of *descriptions*. One kind of description is the *fully specified name* that ends with the semantic category label, e.g., the description "Ocular hypermia (disorder)" implies that the concept belongs to the semantic category, disorder. In addition, the concepts are related to each other by *defining relationships*.

Compared to the traditional schema and ontology mapping problem[26;28;30], the problem of mapping forms to SNOMED CT raises several new challenges. First, a form is graphical user interface that lacks a well-defined semantical structure among the form elements. Form understanding is

challenging[40]. Second, the SNOMED CT is a large medical knowledge base that encodes concepts and relationships from many aspects of clinical information. Terminology navigation and efficient retrieval of relevant terms is difficult. Third, both forms and SNOMED CT usually do not have instances. Hence, the instance-based techniques for schema mappings are hard to apply. Finally, forms and the SNOMED CT are two entirely different structures. It is almost infeasible to convert them into a uniform formalism. Conceptually, the problem of form term refinement could be compared to the problems of social tag refinement and query refinement using ontologies and controlled vocabularies. While the latter ones belong to the IR perspective, form term refinement for the proposed framework belongs to the information modeling (and database design) perspective. Another difference is that the domain expertise of users of this framework is likely to be high as opposed to the IR users.

There are SNOMED CT terminology services that allow users to search concepts through the indexes of the concept descriptions in the SNOMED CT. The key problems with the current systems include too many irrelevant results and inability to distinguish semantic categories. In this section, we introduce and address the problem of mapping a given form term to a unique SNOMED CT concept. We focus on extracting the context of a term, and on using the context to improve the results of retrieving relevant SNOMED CT concepts.

Let us first consider solving the mapping problem using existing services. There are several browsers that provide public access to the SNOMED CT[79]. Underneath these browsers, the user-supplied keyword is compared with the descriptions of SNOMED CT concepts using certain linguistic techniques, and a ranked list of all matching concepts is created and returned for browsing purposes. These services can be understood to provide two kinds of mapping: (i) general mapping, wherein the user term is matched against all the SNOMED CT concepts, (ii) category-specific mapping, wherein the term is matched only against the concepts belonging to a specific semantic category.

As an example, we consider the *Snoflake* browser provided by the Dataline Software Ltd[80]. To perform the mapping, *Snoflake* looks for the SNOMED CT descriptions that contain the search term, and returns the associated concepts for further browsing. Each concept is assigned a *match-weight*,

which is calculated as the overlap ratio between the words contained in the search term and the words contained in the concept's fully-specified name. The concepts are sorted in a non-increasing order of their weights, and in an increasing order of their concept identifiers for equally weighing concepts. Figures 10.9a and 10.9b show the screen shots of the results of mapping the term "eyes" using general and category-specific methods, respectively. For each retrieved concept, the browser returns the concept id, the fully-specified name, and a visual bar representing the match-weight. Despite their public availability, these browsing services are inadequate to address the mapping problem due to the following reasons.

- Different clinicians specify different form terms to describe the same clinical concept, e.g., the use of abbreviations ("MRN," "Med. Rec.#") or synonymous and hyponymous terms ("vital signs," "constitutional," "physical status"). The services are not designed to handle the wide variation in the terms. We refer to this user-induced challenge as the *diversity challenge*.

- Another issue is due to the inherent richness of forms and SNOMED CT. The same form term, when used in different contexts, may map to different concepts. For instance, in Figure 10.8, the element labeled with the term "Respiratory" in Form 1 maps to a concept belonging to the body structure semantic category; another element labeled with the same term in Form 2 maps to a concept belonging to the observable entity category. This disambiguation task entails expert judgment. Moreover, a single term may linguistically match with multiple concepts, and locating the desired concept within this large result set also requires human intervention. We refer to this as the *context challenge*.

While several linguistic-based works[28;62;63;65] exist for addressing the diversity challenge, the context challenge for mapping form terms is not much explored. In this work, we focus on this challenge and propose a form structure-based approach to automatically retrieve an accurate concept corresponding to a given term.

10.2.1 Solution Premises and Representations

The proposed approach is based on the following premises. First, the key to the mapping problem is to identify the SNOMED CT semantic category appropriate for a given term. Once this identification is done, the first, i.e., the most string-similar, result retrieved by the category-specific mapping is usually the desired concept. For example, consider the element labeled with the term "Eyes" from Form 1 in Figure 10.8. If there is a mechanism to determine its semantic category, which in this case is body structure, then the desired concept could be recovered through a category-specific mapping, as shown in Figure 10.9b. The second premise is that the identification of a term's semantic category requires the knowledge of the *context* in which the term has been specified. We hypothesize that the term context can be derived from the semantic structure of the form, and that the implicit relationship between the term context and the desired semantic category can be formally captured into a statistical model. To materialize this, we employ the `form tree`, a representation construct to capture the semantic structure of a form; and we devise a machine-learning based model, the `sClassifier`, that classifies a given term into a semantic category based on the structure of the `form tree`.

In sum, the proposed approach functions in the following manner. (1) Determine the SNOMED CT semantic category of a given form term using the structure-based model. (2) Perform a category-specific mapping and map the term to the first returned concept.

The mapping problem is about finding semantically, and not just linguistically, matching SNOMED CT concepts for form terms. For this, we need the schemes to accurately capture the semantics of forms as well as SNOMED CT. Earlier we described the representation scheme adopt for a form, i.e., a form tree, and here we describe the representation schemes adopted for SNOMED CT.

The SNOMED CT services are owned, maintained, and distributed by the International Health Terminology Standards Development Organization [81]. The SNOMED CT is the most comprehensive clinical vocabulary that precisely represents clinical information across the scope of healthcare [82]. It consists of *concepts*, *terms*, and *relationships*. Each *concept* is identified by a unique identifier, *concept id*, and is represented by a unique human readable term known as the *fully specified name*.

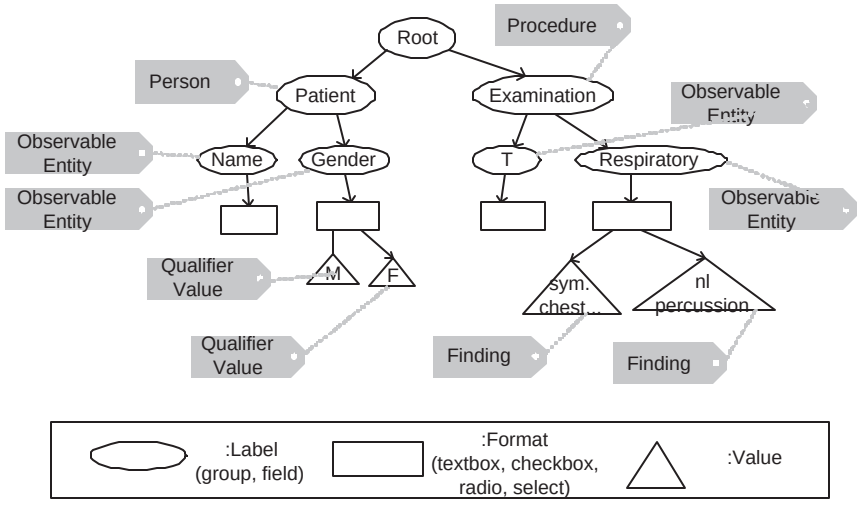


Figure 10.10: A Form Tree and the associated SNOMED CT semantic categories

Table 10.5: Descriptions of a SNOMED CT Concept

Description	Value
Fully Specified Name	Respiratory rate (observable entity)
Preferred Term	Respiratory rate
Synonym	Rate of Respiration
Synonym	Respiration Frequency

A concept is associated with multiple *descriptions*, which are the terms used to describe the concept. Each concept has 3 kinds of descriptions: (i) fully specified name: an unambiguous way to name a concept, (ii) preferred term: the most common term used by the clinicians to describe the concept, and (iii) synonym: additional terms used to describe the concept. As an example, the descriptions of a concept (concept id: C0231832), are enlisted in Table 10.5. The fully specified name ends with a parenthesized text that represents the *semantic category* to which the concept belongs, e.g., observable entity in this case. There are 19 semantic categories in SNOMED CT. The concepts are associated with each other using defining *relationships*. For example, the relationships of the concept *Fracture of bone (disorder)* with other concepts are enlisted in Table 10.6.

In this work, we are interested in concepts, fully specified names, and semantic categories. The semantic category of a given concept represents the top-level granularity concept associated with the concept through the IS-A relationship.

Table 10.6: SNOMED CT Relationships

Relationship Type	Related Concept
Is a	Bone Injury(disorder)
Associated morphology	Fracture(morph. abnormality)
Finding Site	Bone Structure(body structure)

10.2.2 Solution: Mapping Terms to Concepts

In this section, we present our solution[83] to map a form term to a SNOMED CT concept. The target application of the mapping approach is any methodology that employs forms to design and/or populate databases, wherein the user supplied form terms are used for naming the database elements. Through mapping, we intend to standardize the user terms to generate standard annotated databases, and thereby support future data integration and analysis.

We have shown that solutions solely based on the linguistic similarity between the term and the concept description do not achieve good accuracy. This is because a form term does not stand alone and is strongly associated with a certain context within the form. The same term when used in different contexts maps to different SNOMED CT concepts from different semantic categories. To address this, we propose a solution that (i) exploits the semantic structure of forms to determine the context, and the appropriate semantic category for a given term, and (ii) maps the term to a linguistically matching concept within the determined semantic category. Before we present our solution, we present the fEHR system, one of our initial motivations to devise an approach to map forms to SNOMED CT.

The form terms specified by the clinicians are eventually used to name the elements of the database schema, e.g., the term "Patient," used in the forms in Figure 10.8 is likely to be used to name a database table. At present, the terminology process in the fEHR system is uncontrolled in that the clinicians are free to supply any terms to the system. Due to differences in perceptions and domain expertise, it is highly likely that different clinicians would specify the same concept in different ways, thereby causing complications in future integration and analysis. To address this concern, we add a new middle-ware component to the fEHR system that maps the user defined form

terms into SNOMED CT concepts, thereby generating standard-annotated database schemas. While fEHR is a specific application, the proposed mapping approach can be employed by any application that utilizes forms to design and populate clinical information systems.

As discussed before, there are two main challenges associated with the mapping problem: the diversity challenge and the context challenge. An intuitive solution is to linguistically match the form term with the SNOMED CT concept descriptions and return the most matching concept. An example of such a linguistic technique is the general mapping provided by any SNOMED CT browser as shown in Figure 10.9a. Such methods, when combined with sophisticated term processing techniques, can certainly address the diversity challenge to a great extent. However, such methods treat every term as a context-independent entity. As such, they fail to disambiguate the context in which the term has been specified, and to determine the appropriate SNOMED CT semantic category for a given term. As a result, the context challenge remains unresolved.

To address this special challenge, an advanced simulation of the category-specific mapping (See Figure 10.9b) is needed, that automatically determines the semantic category for a given term based on its context, and maps the term to a linguistically matching concept belonging to that category. The key in performing this simulation is the automatic identification of the semantic category based on the context of the form term. To accomplish this, we design a statistical model that exploits the structure of the semantic `form tree` to derive the term context and predict the SNOMED CT semantic category. In the next subsections, we first describe this structure-based model, and then illustrate the overall approach.

Structure-based Model

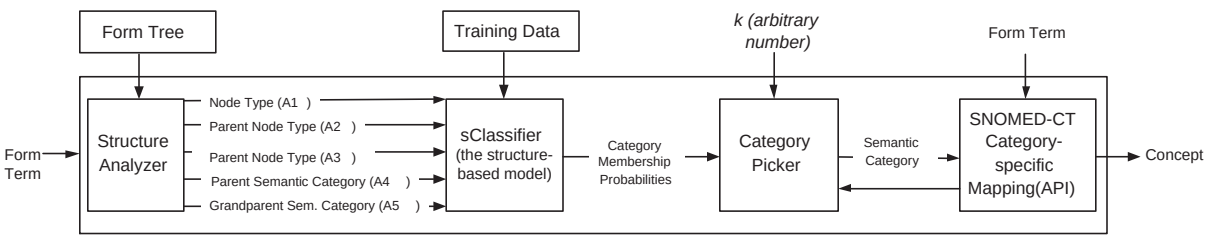


Figure 10.11: SNOMED CT Mapping

As mentioned before, the key to the mapping solution is to determine the semantic category appropriate for a given form term. What we intend to achieve is depicted in Figure 10.10 that shows a `form tree`, and the semantic categories associated with the terms contained in various nodes. A human expert can intuitively determine the category by perceiving the context of a term as implied in the form. To accomplish this automatically, we hypothesize that the context of a term contained in a given node can be extracted from the structure of the form tree. We encode the intuitive expert knowledge into a statistical model that earns the ability to determine the semantic category of any form term based on the tree structure. We refer to this model as the `sClassifier`. Following is a technical description of the model.

We use the Naive Bayes Classifier[84] to design the `sClassifier`. The Naive Bayes Classifier is one of the most effective classifiers based on the powerful Bayes theorem that determines the posterior probability, $P(\mathcal{H}\|X)$, that the hypothesis $\mathcal{H}$ holds for an observed data sample X. Each sample is represented as $X = (x_1, x_2, ...x_n)$ depicting measurements from n attributes $A_1, A_2, ..., A_n$. For a given set of m classes, $C_1, C_2..., C_m$, the classifier calculates the posterior probability $P(C_i\|X)$ for each class and assigns the data sample to the class with the maximum value. This classifier works with an assumption of class conditional independence, which states that the effect of an attribute on a given class is independent of the values of other attributes. To customize `sClassifier` for the problem of classifying a form term into a SNOMED CT semantic category, we use the following parameters.

***Class Labels*:** The classes comprise the predefined SNOMED CT semantic categories. Out of all the available semantic categories, we choose the ones that are frequently associated with clinical form terms. In particular, the model employs the following class labels: attribute, body structure, disorder, finding, observable entity, occupation, person, physical object, procedure, product, qualifier value, racial group, record artifact, and situation.

Data Attributes: Given any term, $\lambda(n)$, contained in a node n, the goal of the classification process is to predict the most appropriate class label based on the term context. This implies that the classification attributes should reflect the context of the node n that holds the term. We

hypothesize that the node context can be extracted from the local structure in the semantic form tree, and choose the following categorical attributes to accomplish classification:

1. Node type ($\tau(n)$): As per the Definition 3.1.2.
2. Parent node type ($\tau(n_j)$): The node type of the parent node n_j of the node n.
3. Child node Type ($\tau(n_i)$): The node type of the first child node n_i of the node n.
4. Parent semantic category: The semantic category of the parent node n_j, as determined by the `sClassifier`.
5. Grandparent semantic category: The semantic category of the grandparent node n_k, as determined by the `sClassifier`.

The domain of the values taken by the first three attributes includes label(group, field), field format (textbox/ checkbox/ radiobutton/ select), and value as defined in Definition 3.1.2. The domain of values taken by the last two attributes is same as that of the class labels. As an example, the values for the 5 attributes for the node labeled as "T" in Figure 10.10 are *field label*, *group label*, *textbox*, procedure, and *null* (since the tree root is not associated with any semantic category), respectively.

The main goal of this work is to study the impact, of exploiting the form structure on mapping performance. To create the structure-based model, we experiment with the Naive Bayes Classifier. In the future, we also intend to study the impact, of using other classifiers such as k Neural Networks and Classification Association Rules, on the model's performance.

Overall Approach

The overall approach to find a unique SNOMED CT concept suitable for a given form term is summarized in Figure 10.11. The approach is hybrid in nature, in that the first 3 modules are structure-based, and the last one is linguistic-based. The last module could be any application programming interface (API) that provides programmatic access to search and browse the SNOMED CT based on certain linguistic techniques.

The input to the mapping approach is the form term, contained by a particular node in the form tree. The first module, structure analyzer, exploits the structure of the form tree to extract

the context of the term. The context, represented as the 5 attributes, A_1, A_2, A_3, A_4, A_5, is fed as the input to the structure-based model, the `sClassifier`. This trained model determines the class membership probabilities for the given term. In other words, the model determines the probability that the term belongs to any of the 15 semantic categories.

The next module, category picker, sorts the probabilities in the non-increasing order of values. It then picks the top ranked category, and performs the "concept presence test" using the API module. The test determines whether any SNOMED CT concept, with a linguistic match between the term and the descriptions, exists in the given category. If the test is positive, then the control is passed over to the API module that performs a category-specific mapping and returns the "most" linguistically matching concept as the output. However, the test result may also be negative mainly because: (i) the training data may be inconsistent; the terms having the same attribute values may belong to different classes, e.g., the terms "Patient" and "Examination" in Figure 10.10 belong to different categories, person and procedure, respectively; (ii) the concept is not yet a part of the SNOMED CT, or there is no linguistic match between the term and the description of the desired concept. In such cases, the category picker module picks the next highest ranked category and repeats until a concept is retrieved, or the top k classes have been explored, where k is an arbitrarily chosen number between 1 and 15.

Chapter 11: Mapping Discovery and Validation

The next part of our solution is shown in the Figure 11.1. The goal of the mapping discovery and validation process is to detect semantically matching elements between the form tree and the existing database, i.e., determine the elements of the form tree that already exist in the database. This is decomposed into two steps.

- Derive the "initial correspondences" between the elements of a given form tree and those of a given database. These are the correspondences from the form tree to database which are determined based on certain linguistic or semantic matching techniques.
- Validate the set of discovered correspondences using user intervention or automatic heuristics. The validated correspondences are used for making important merging decisions in the subsequent stages of the mapping algorithms.

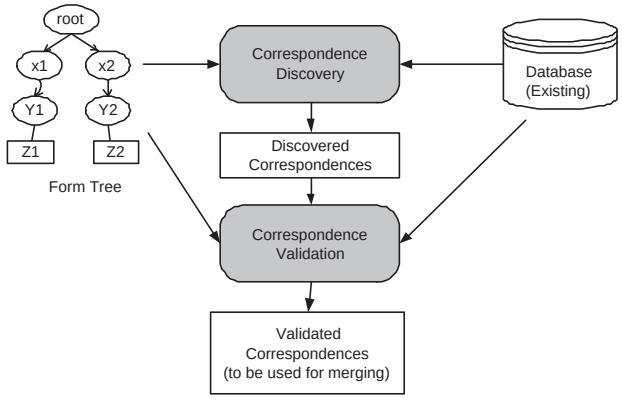


Figure 11.1: Correspondence Discovery and Validation

11.1 Discovering Correspondences

Mapping discovery is the process of matching all the form terms with all the existing database elements, i.e., column, table, and value names. This could be performed in two ways: using raw form terms, or using concept annotated form terms, given the term annotation module is used.

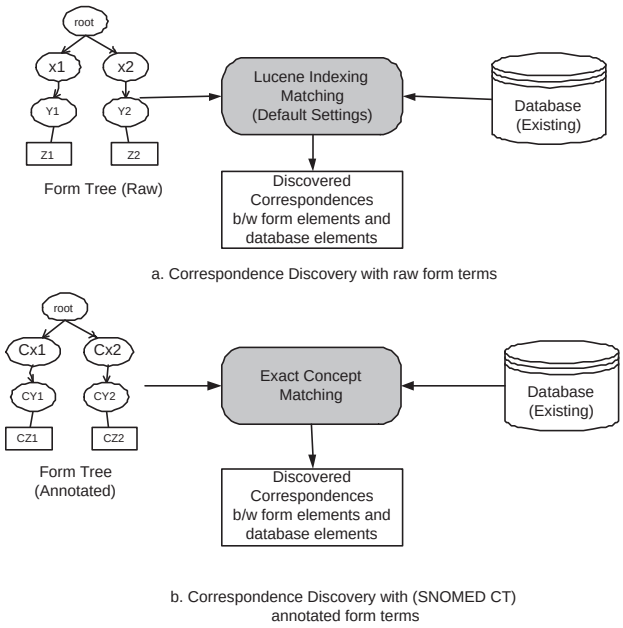


Figure 11.2: Correspondence Discovery between Form Tree Elements and Database Elements

11.1.1 With Raw Form Terms

The process of discovering correspondences between form terms and database elements is depicted in the Figure 11.2a. In particular, the system indexes all the element names in the database. For each form element, the system automatically discovers a list of candidate database element names that likely match the form element. The back-end system consists of an index of the all elements in the database. We build inverted indexes on table names, column names, and each table cell whose type is String. We use the Lucene[85] open source search engine for indexing and searching. This can be understood using an example. Consider an existing database shown in the Figure B.1 (see Appendix), and consider the set of forms shown in the Figures A.4, A.5, A.6, and A.7 (See Appendix). Using the mapping discovery module, the following 34 correspondences would be generated between the forms and the existing database. We represent a discovered correspondence as $f \rightsquigarrow \mathsf{D.d}$ where f is a form element and d is a database element belonging to the database component D.

1. *Past Medical History* $\rightsquigarrow$History

2. *Past Medical History* $\rightsquigarrow$History.PastMedicalHistory

3. *Resident* ⇝Patient
4. *Resident* ⇝PrimaryCarePhysician
5. *Name* ⇝Patient.Name
6. *Name* ⇝PrimaryCarePhysician.Name
7. *Diagnosis* ⇝ReviewOfSystems
8. *Diagnosis* ⇝Examination
9. *Notes* ⇝History.ReasonForVisit
10. *Notes* ⇝MedicalDecisionMaking.Notes
11. *Vital Signs/Physical Status* ⇝Constitutional
12. *Vital Signs/Physical Status* ⇝ReviewOfSystems.Constitutional
13. *R* ⇝Constitutional.RR
14. *R* ⇝ReviewOfSystems.Respiratory
15. *Vision* ⇝Vision
16. *Vision* ⇝Eyes
17. *Med Rec #* ⇝Patient.MRN
18. *Allergies* ⇝History.Allergies
19. *T* ⇝Constitutional.T
20. *BP* ⇝Constitutional.BP
21. *P* ⇝Constitutional.P
22. *Ht* ⇝Constitutional.Ht
23. *Wt* ⇝Constitutional.Wt

24. *CognitiveStatus* ⇝Psych

25. *SkinColor* ⇝Skin

26. *Appearance* ⇝Constitutional.Appearance

27. *VitalSigns* ⇝Constitutional

28. *PhysicalStatus* ⇝Constitutional

29. *Memory* ⇝Psych.IntactMemory

30. *Intact* ⇝Psych.IntactMemory

31. *VitalSigns* ⇝ReviewOfSystems.Constitutional

32. *PhysicalStatus* ⇝ReviewOfSystems.Constitutional

33. *R/Adequate* ⇝Vision.Options.Adequate

34. *L/Adequate* ⇝Vision.Options.Adequate

These correspondences have been manually identified based on the terms and their semantics taken independently of their context. The point is to depict the large number of correspondences that get discovered as a result of adopting an automatic technique. The correspondences (1) through (16) fall under the category of $1 : m$, i.e., a form element is discovered to be associated with multiple database elements. Correspondences (17) through (26) denote the category of $1 : 1$ correspondences, i.e., a unique database element has been discovered for each form element in this category. Correspondences (27) through (34) denote that category of $m : 1$ correspondences wherein several form elements are discovered to correspond to a single database element.

11.1.2 With Concept Annotated Terms

Another technique for correspondence discovery comes into picture when the term annotation module, described in Section 10.2, is used to further refine the semantics of the form tree, and when each element of the existing database is also assumed to be annotated using the same concept,

i.e., SNOMED CT. In this scenario, the above example of discovering correspondences between the forms in Figures A.4 through A.7 and the database in Figure B.1 (see Appendix) would lead to the following 15 correspondences.

1. *Past Medical History* $\rightsquigarrow$History.PastMedicalHistory
2. *Name* $\rightsquigarrow$Patient.Name
3. *Notes* $\rightsquigarrow$MedicalDecisionMaking.Notes
4. *R* $\rightsquigarrow$Constitutional.RR
5. *Vision* $\rightsquigarrow$Vision
6. *Med Rec #* $\rightsquigarrow$Patient.MRN
7. *Allergies* $\rightsquigarrow$History.Allergies
8. *T* $\rightsquigarrow$Constitutional.T
9. *BP* $\rightsquigarrow$Constitutional.BP
10. *P* $\rightsquigarrow$Constitutional.P
11. *Ht* $\rightsquigarrow$Constitutional.Ht
12. *Wt* $\rightsquigarrow$Constitutional.Wt
13. *Appearance* $\rightsquigarrow$Constitutional.Appearance
14. *Intact* $\rightsquigarrow$Psych.IntactMemory
15. *R/Adequate* $\rightsquigarrow$Vision.Options.Adequate
16. *L/Adequate* $\rightsquigarrow$Vision.Options.Adequate

The use of annotation helped in disambiguating the semantics of terms and eliminated several (more than 50% in this small example) incorrect correspondences wherein the terms resembled linguistically but differed semantically from one another.

11.2 Validating Correspondences

After the correspondences are discovered using completely automated linguistic or semantic techniques, it becomes a must to further validate them to ensure the *correctness* principle. First, let us consider the correspondences discovered using the linguistic techniques as described in the Section 11.1.1.

1. The correspondences (1) through (16) from the category $1:m$ need validation given the principle of *compactness* $\mathcal{P}3$. Hence, out of all the correspondences for a given form element, either none or one is valid. For instance, for the form element *Resident*, the valid correspondence is the one with Patient (i.e., (3)) and not the one with PrimaryCarePhysician (i.e., (4)).

2. For the case of $1:1$, either a correspondence is valid or invalid, e.g., the correspondence (25) *SkinColor* $\rightsquigarrow$Skin is not valid.

3. For the case of $m:1$ for a given form element, it is possible that all the correspondences are valid as in the case of (33) *R/Adequate* $\rightsquigarrow$Vision.Options.Adequate and (34) *L/Adequate* $\rightsquigarrow$Vision.Options.Adequate.

Due to the above observations, it becomes important to add a layer of validation to the discovered correspondences. Even when semantic methods of discovery are used, validation is required as two semantically similar elements may not be structurally compatible, and hence may not be fit to be merged into the resultant database.

A naive approach to validate the entire set of discovered correspondences is to use a user intervention for each correspondence as shown in the Figure 11.3. Similar techniques have been used in schema mapping solutions[86–88], where users need to specify a set of simple correspondences between schemas as part of the input for a schema mapping solution. If this approach is adopted, the number of validation screens presented to users will be very large especially when the forms and the databases scale up. This would violate the goal of minimal user intervention. Therefore, we use certain heuristics to automatically eliminate or validate certain correspondences, and then pass on the remaining for user intervention. The user selects the best matches from a list of candidate

names based on her domain knowledge. The higher level process flow is illustrated in Figure 11.4.

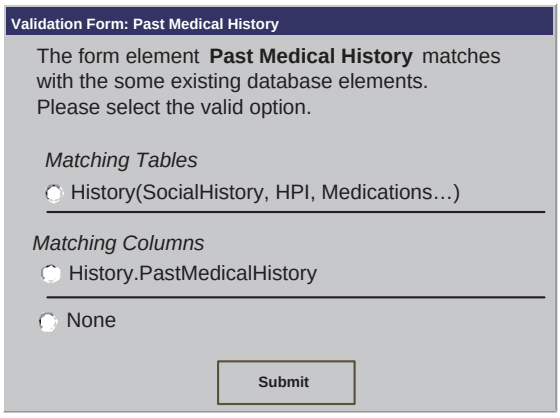


Figure 11.3: A Validation Form for User Intervention

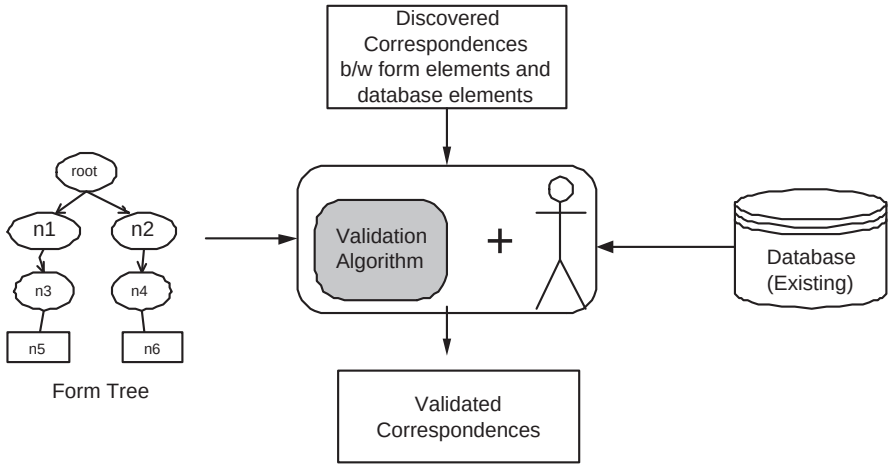


Figure 11.4: Correspondence Validation

The automatic validation process occurs in multiple phases, wherein every correspondence transitions from one "state" to another. We define 5 states for correspondences in the following manner.

1. Initial State (S_i): When a correspondence is just discovered.

2. User Validation State (S_u): When a correspondence is fit for direct validation by the user, i.e., it needs to be presented to the user as shown in Figure 11.3.

3. Revisit State (S_r): When a correspondence needs to be revisited after all the user validations have been performed.

4. Eliminated State (S_e): When a correspondence is eliminated or devalidated.

5. Validated State (S_v): When a correspondence is selected to be the final valid correspondence.

During the validation process, a correspondence transitions from one state to the other as shown in Figure 11.5a. The validation phases are described as follows.

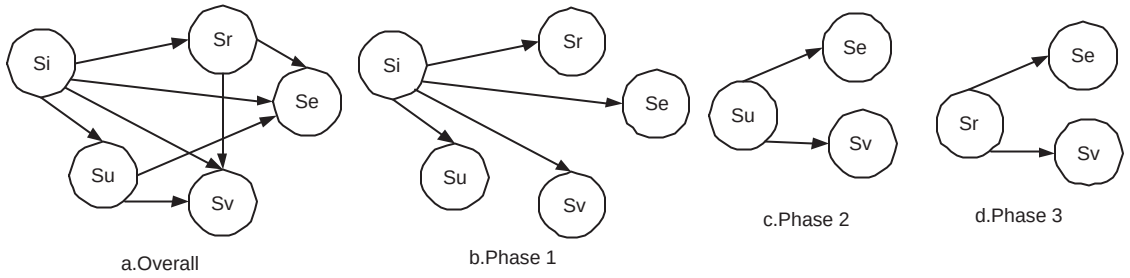


Figure 11.5: Correspondence State Transitions

1. First, certain heuristics are applied to the discovered correspondences and the transitions are made from the initial state to the 4 other states as shown in the Figure 11.5b. These heuristics are described as follows.

   - The first heuristic stipulates that when multiple value nodes belonging to a radiobutton node correspond to distinct values from a lookup table, referred to as the "winner table," then these correspondences are valid. Also, an implied correspondence between the field node containing the radio node and the lookup table is created and validated. This scenario is shown in Figure 11.6a. When multiple winner tables are found for a given set of sibling value nodes, then the value correspondences are transferred to the user validation state, S_u, and the implied table correspondences are transferred to the revisit state S_r. This is depicted in Figure 11.6b. All other correspondences from the sibling value nodes are eliminated.

   - The second heuristic stipulates that when multiple sibling element nodes correspond to the distinct column of the same table, then an implied correspondence is created between the parent element node and the table, and this is moved further for user validation, i.e., to state S_u. The column correspondences are moved to be revisited, i.e., to state S_r.

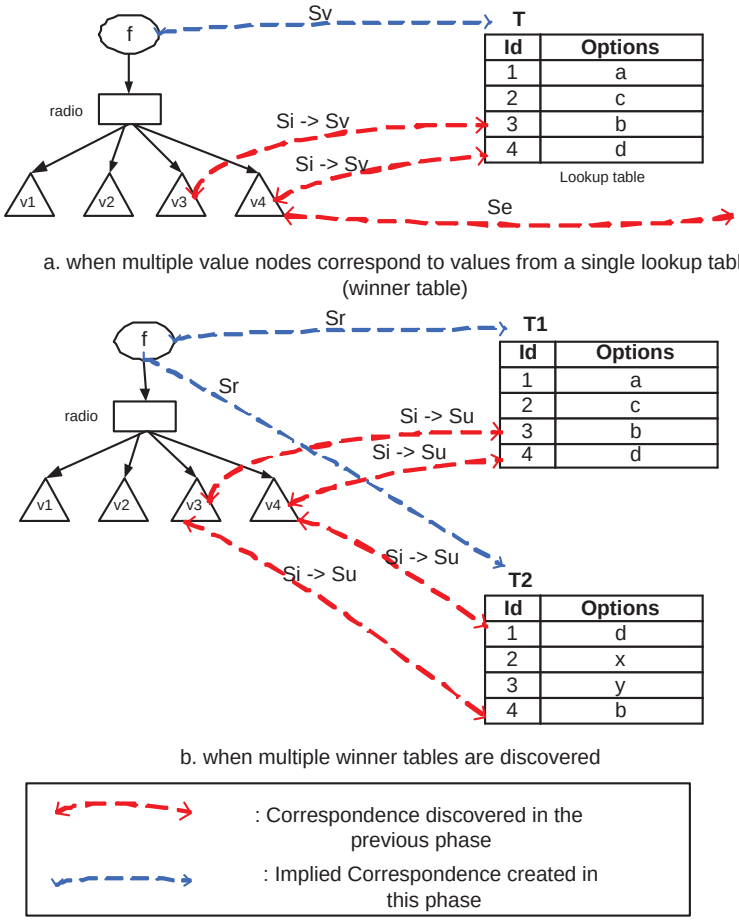


Figure 11.6: Validation Heuristic 1

The idea is to have the user take the higher level, i.e., table level decisions, and defer the column-level decisions to the design and evolution module of the framework. This scenario is depicted in Figure 11.7a. All other correspondences from the sibling element nodes are eliminated.

- The third heuristic stipulates that when sibling element nodes correspond to the column of table T_1, or to any table referenced by table T_1, then an implied correspondence is created between the parent element node and the table T_1 and it is moved for user validation. This scenario is depicted in Figure 11.7b.

2. In the next phase, all the correspondences in the user validation state, S_u, are presented to the user who then accepts or rejects them individually. The transitions that occur in this phase

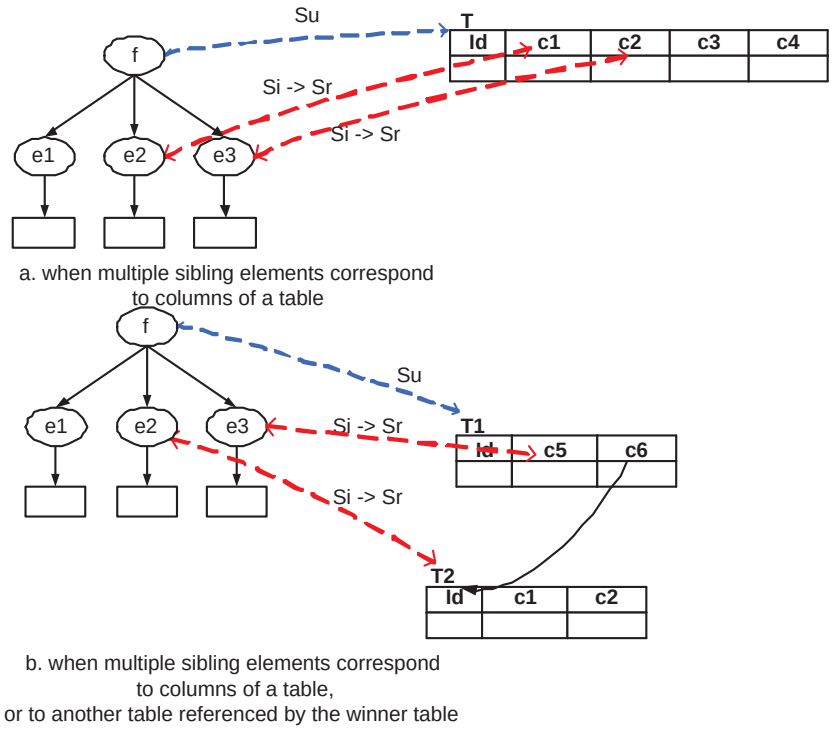


Figure 11.7: Validation Heuristics 2 and 3

are depicted in Figure 11.5c.

3. In the last phase, the correspondences in the revisit state, S_r, are revisited. The decision for their validation or elimination is made as per the results of the previous phase. For instance, in the scenario shown in Figure 11.6b, if the correspondence $v3 \rightsquigarrow \mathsf{T1.Options.b}$ is validated by the user, then the correspondence $f \rightsquigarrow \mathsf{T1}$ is moved to the valid state S_v, and the correspondence $f \rightsquigarrow \mathsf{T2}$ is moved to the eliminated state S_e. Likewise, for scenario in Figure 11.7a, if the correspondence $f \rightsquigarrow \mathsf{T}$ is validated by the user then the related correspondences, i.e., from child nodes of f, are also validated. The state diagram for this phase is shown in Figure 11.5d.

Chapter 12: Database Design and Evolution

Given a form tree and the discovered correspondences between the tree and the existing database, the next step is to integrate the form tree into the database. The approach for evolving the database is illustrated in the Figure 12.1, and could be summarized in the following manner. The form tree is translated into an equivalent new database, using the *Birthing* algorithm. All the validated correspondences are transferred to this newly created database. This point onward, each correspondence is from an element in the new database to an element in the existing database. The correspondences are analyzed for their appropriateness in terms of merging into the existing database. The *Merging* algorithm studies each correspondence and decides whether a given database element in the new database is eligible to be merged with the corresponding element in the existing database. Finally, the merging decisions are used to extend the existing database with respect to the new database using the *Extension* algorithm. The entire process of database evolution is closely guided by the principles of high-quality ($\mathcal{P}1$ through $\mathcal{P}4$) and optimization ($\mathcal{P}5$ through $\mathcal{P}7$). We describe the birthing and the merging algorithms in the next sections.

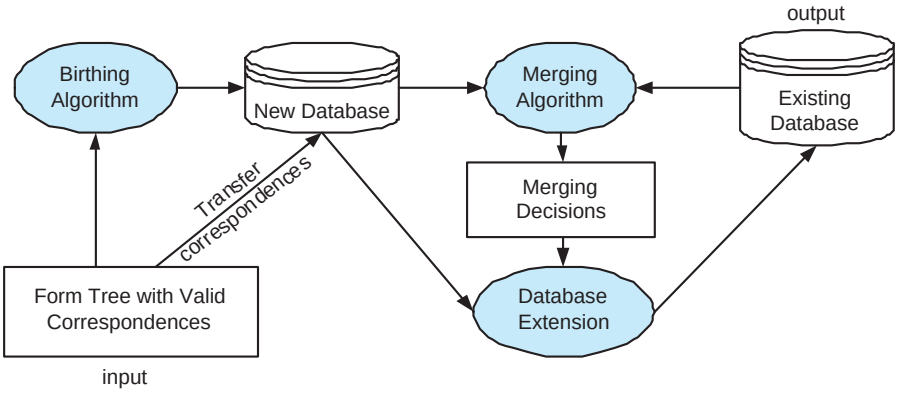


Figure 12.1: Database Design and Evolution: Overall Approach

12.1 Birthing Algorithm

Birthing is the process of automatically generating a new database from a given set of user requirements. The term *birthing* was proposed by Jagadish et al.[11]. In this thesis, the birthing algorithm takes as input a form tree and creates new database tables. There are well-defined rules for translating an ER diagram into a relational database[35]. Consider a form tree $\mathcal{FT} = (N, E, <_{sib}, root)$ as a conceptual model. An internal logical element $e \in \mathcal{E}$ corresponds to an entity and an edge between two logical elements $(n_i \longrightarrow n_j) \in E$, $n_i, n_j \in \mathcal{E}$ corresponds to a relationship in an ER model. The cardinality constraints of such a relationship is always many-to-many because cardinality constraints cannot be obtained directly from a form (though we implemented a user-friendly component that asks users to answer cardinality questions; see the section about experiments.) However, a form tree is different from a traditional ER model in many ways. It contains *inputs* and *values* which can be organized in complex and messy ways. Moreover, the hierarchical relationships between form elements capture important semantics regarding the information collected on the form. Given a form or a form tree, there could be many ways of designing a corresponding database as shown in Figure 12.2.

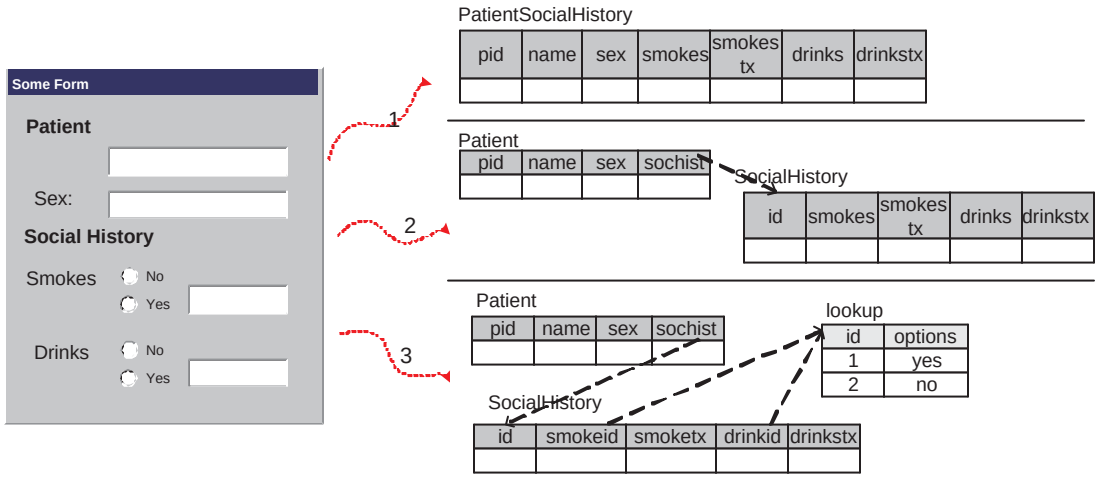


Figure 12.2: Birthing Databases using Forms: Multiple Techniques

Through the birthing algorithm, we have devised a disciplined way of performing the transformation from form tree to database while ensuring the compliance with the quality and optimization

principles. Let us understand the higher-level sketch of the birthing algorithm and how it derives a database in a principled manner, using the two examples illustrated in Figure 12.3. The preliminary inputs to the birthing algorithm are the root of the form tree and an empty database. The algorithm works in the following manner. Starting with the tree root, a relation T is created for each internal non-root node n. To every created relation T corresponding to n, attributes are added such that the parent-child associations between n are its child nodes are recorded correctly in the database. The procedure is repeated if a child node n_i is an internal node, otherwise an attribute corresponding to n_i is added to the relation T, e.g., Time is an attribute of AllergyInjection. Finally, the relationship among all the child nodes of the root node, i.e., among all the categories in the given form, is recorded by creating a relationship table between any pair of child nodes, e.g., the relationship between the two relations Reaction and AlleryInjection is captured by the join relation AllergyInjectionReaction in Figure 12.3a. In terms of handling the advanced format nodes as the as ones in Figure 12.3b, the algorithm works in the following manner. Consider the internal format nodes, *rbvis* and *cbsym*. The radiobutton node *rbvis* indicates the presence of multiple exclusive options in the form which should get translated to values in the database as shown in the relation Visit. The checkbox node *cbsym* indicates the presence of multiple non-exclusive options in the form which should get translated to distinct (boolean) attributes as shown in the relation Symptoms. Furthermore, this form also has a subcategory BP under the category *Current State*. This relationship is captured by creating a join table CurrentStateBP between the respective relations to cover all possibilities of cardinalities between the two entities.

We now elaborate on the normalization property, i.e., the traditional criteria for avoiding certain undesirable characteristics. A normalized database (i.e., in the Third Normal Form[38]) is defined with respect to a set of functional dependencies. Since in our case a user only provides data entry forms, we need to automatically deduce functional dependencies from data entry forms and translate the functional dependencies to the associated database. We represent the translation relationship between a form tree and a database as a set of correspondences. Specifically, given a form tree $\mathcal{FT} = (N, E, root)$ and a database $\mathcal{D} = (I, R, \Sigma)$, a correspondence $\mathcal{FT}$:P/e $\rightsquigarrow$ $\mathcal{D}$:D.d relates a node

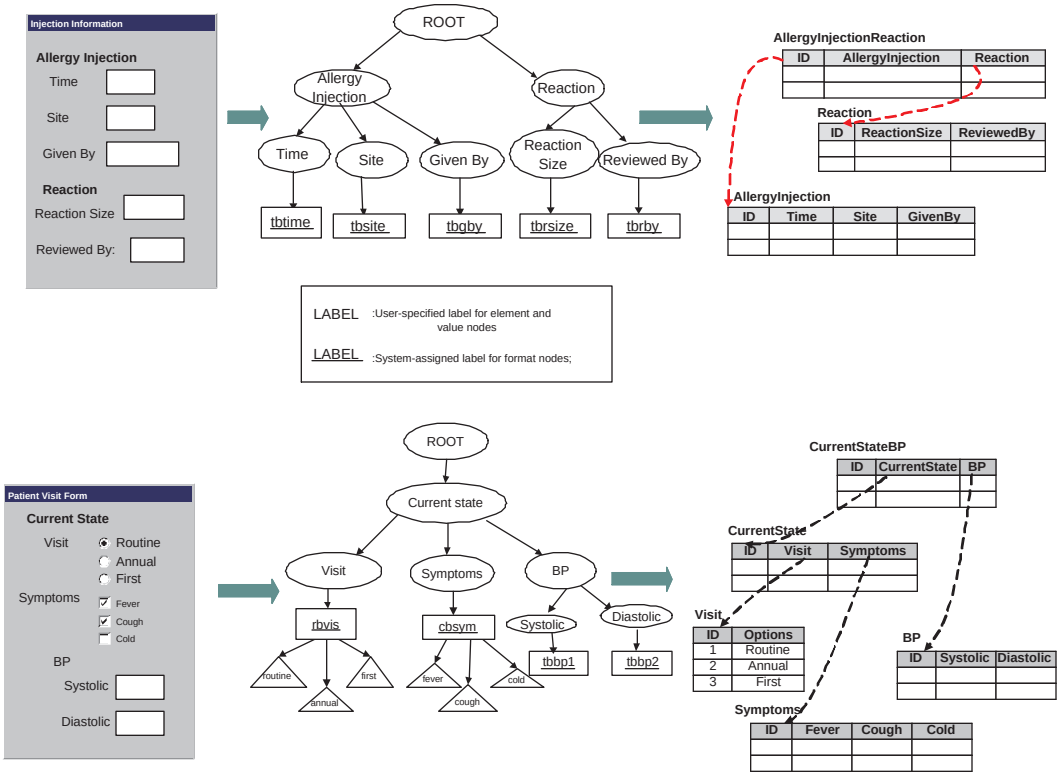


Figure 12.3: Examples: Database Birthing

$e \in N$ of the form tree reached by a simple path P to an element d in the database component D. A simple path P is always relative to the root of the form tree, in which "/" is used to represent a parent/child relationship. A database component could be either a tuple in a table T or the schema of the table. We say a functional dependency $\mathcal{D}$:D.d$_i \rightarrow$ $\mathcal{D}$:D.d$_j$ in the database is *associated with* the $\mathcal{FT}$ if there are two correspondences $\mathcal{FT}$:P/e$_i \rightsquigarrow$ $\mathcal{D}$:D.d$_i$ and $\mathcal{FT}$:P/e$_j \rightsquigarrow$ $\mathcal{D}$:D.d$_j$, where P/e$_i \rightarrow$ P/e$_j$ is a functional dependency in the $\mathcal{FT}$. To formally specify the normalization property, we consider the integrity constraints derivable from a form tree. Element nodes may represent either entities or attributes of entities in the application domain described by the form. Format nodes and value nodes in a form tree are directly related to attributes or values of entities. A parent-child edge between two element nodes gives rise to a functional dependency relationship between the nodes. Figure 12.5 illustrates various cases of deriving integrity constraints from a form tree. In particular, the procedure starts with the root of the form tree and implements the patterns illustrated in Figures 12.5 and 12.6. The semantics of the patterns are illustrated in Figure 12.4.

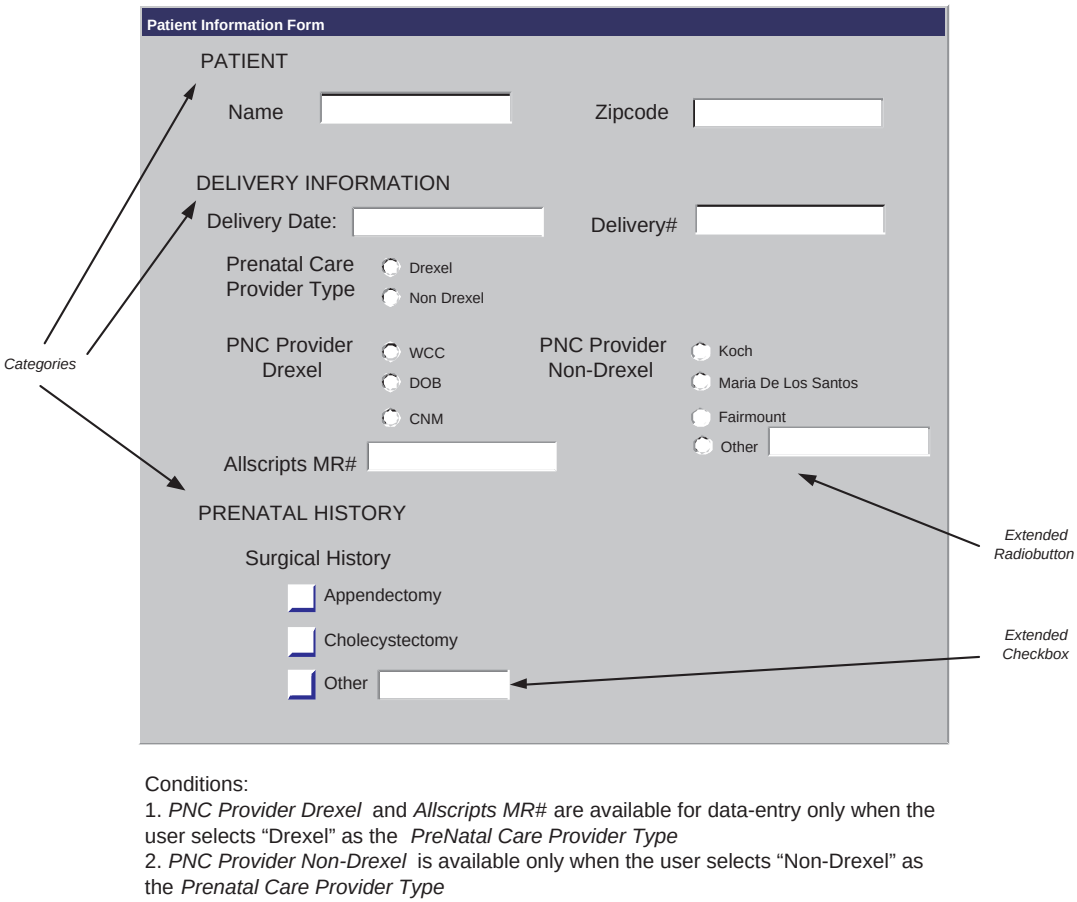


Figure 12.4: A Typical Form in the Healthcare Domain

Pattern (1): Textbox Figure 12.5a shows the textbox pattern. When this pattern is encountered, the algorithm induces an artificial functional dependency into the database. If n has a single child which is a text box and n has a parent, then n is mapped to a column named after n in the parent table T_j. The textbox n_i and the column c are associated using a "data binding mapping," which indicates that the value collected through the textbox is stored in the column.

Pattern (2): Radiobuttongroup This pattern is shown in Figure 12.5b). The presence of this pattern indicates that the node n containing the radiobutton and the parent node n_j represent two separate entities having a 1:M cardinality between them. To remove any transitive dependency (principle $\mathcal{P}4$), the node n is translated into a new table T. The values are stored in the database as a lookup values.

Pattern (3): Checkbox group This pattern is shown in Figure 12.5c. This pattern is treated

similar to the previous pattern. The only difference is that each value is stored as a yes/no table column since it is possible to select multiple checkboxes at the same time.

Pattern (4): Category-subcategory Figure 12.5d illustrates this pattern. An edge between two logical elements suggests that the respective entities are independent of each other. To honor the normalization principle $\mathcal{P}4$, the nodes are translated into separate entities. To cover different variations of cardinalities, they are connected via a many-to-many relationship table.

Pattern (5): Siblings The root of the form tree is mapped to a table representing an n-ary relationship. The presence of sibling element nodes indicates that the two entities represented by them are independent of each other, and are hence translated into two separate tables. This removes any transitive dependency and thus honors the normalization principle $\mathcal{P}4$. (See Figure 12.5e)

Pattern (6):Conditions The forms had several instances where two fields were linked through a "condition" (See Figure 12.4). We realize that conditions provide important information about databases (e.g. generalization and multi-level generalization) and hence should not be overlooked. Accordingly, we revised the form design module to create a conditional edge between a value node, e.g., *Drexel*, and the conditional field node, e.g., *AllscriptMR#*. In response, the revised birthing module creates a new table T_k*_cond* for each set of conditions associated with a radiobutton value the following principles $\mathcal{P}1$ and $\mathcal{P}6$, and adds a foreign key reference from T_k*cond* to T_j following the principle $\mathcal{P}5$. (See Figure 12.6a)

Pattern (7):Extended Radiobutton We also found several instances of extended radiobutton options (See Fig. 12.4). The enhanced birthing algorithm deals with extended radiobuttons by creating a separate table T_add for all the additional text options associated with the given field. Following the optimization principles $\mathcal{P}5$ and $\mathcal{P}6$, T_add has a column c_l for storing texts associated with all possible extended radiobutton options, and references the parent table T_j as well as the radiobutton look up table T. (See Figure 12.6b)

Pattern (8): Extended Checkbox We also found several instances of extended checkbox options (See Fig. 12.4). The module deals with the extended checkboxes by simply adding a new column c_l, mapped to the text node n_l, in the table T. (See Figure 12.6c)

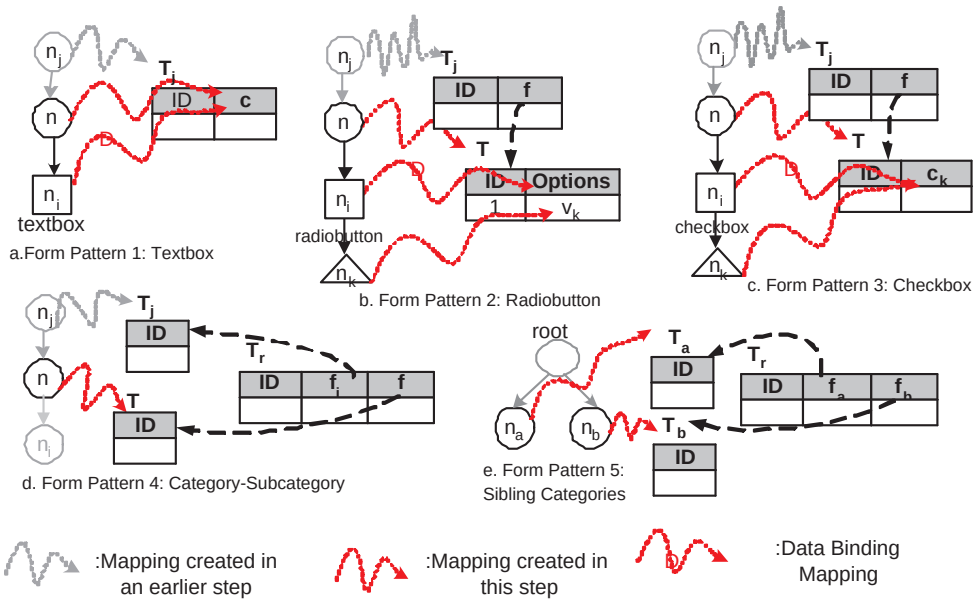


Figure 12.5: Birthing Cases

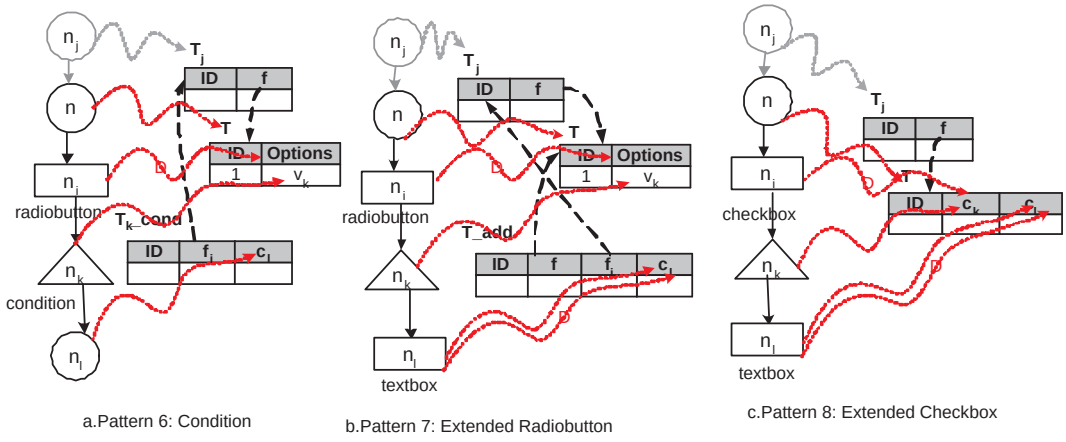


Figure 12.6: Birthing Cases More

Overall, the birthing algorithm creates tables for logical elements and edges between logical elements. A form tree is preprocessed for extracting the data type $\tau(e)$ and constraint $\kappa(e)$ of an element e. In addition, we only consider one-level extension of an input filed, e.g. the check box with the value Obesity extended by a text box. The algorithm 12.1 formally describe the birthing algorithm. It should be noted that the "mapping" in discussion in this section refers to the *transformation mapping*, i.e., which element of the form transforms into which element of the target database. This is different from the *discovered mapping* that refers to which element in the form is semantically

equivalent to which element in an existing database. The algorithm 12.1 generateDB($\mathcal{FT}$) creates a relational database. It calls a procedure createTables(P/n, $\mathcal{D}$, $\mathcal{M}$). The procedure recursively creates tables from a subtree rooted at the node P/n in a top-down fashion, where P is the path from the root to the node n, and $\mathcal{M}$ is the current mapping between the form tree $\mathcal{FT}$ and the database $\mathcal{D}$.

ALGORITHM 12.1: generateDB($\mathcal{FT}$)

Input: a form tree $\mathcal{FT} = (N, E, <_{sib}, root)$

Output: a database $\mathcal{D} = (I, R, \Sigma)$ and a set of mappings $\mathcal{M} = \{M_i$: $\{\mathcal{FT}$:P/e $\rightsquigarrow$ $\mathcal{D}$:D.d$\}$ with $\{\mathcal{D}$:T$_i$.A$_i$ = $\mathcal{D}$:T$_j$.B$_j\}\}$ between $\mathcal{FT}$ and $\mathcal{D}$

Stepts:

1: **let** $\mathcal{D}$ be an empty database and $\mathcal{M}$ be an empty set of mappings;

2: createTables($root$, $\mathcal{D}$, $\mathcal{M}$); */* recursively create tables for $\mathcal{D}$ and mappings for $\mathcal{M}$ in a top-down fashion starting from the root. */*

3: **return** $(\mathcal{D}, \mathcal{M})$;

An extending field is added as an extra column of the table referencing the extended field. Join predicates are added to the mapping as foreign keys are created. The procedure works as follows: (1) It stores values in individual lookup tables; (2) It merges root's children to an n-ary relationship table; and (3) It inlines an element with a single text box child to the element's parent. We turn a relational database into a *database graph*, where the nodes are tables, columns, and values, and edges are foreign key referencing, table-column, and column-value relationships. For a foreign key column, we replace the table-column relationship with a referencing relationship between two tables. Then the form tree stripped off all value and input nodes (i.e., it becomes a subtree) is isomorphic to the database graph generated by createTables from the subtree.

PROCEDURE createTables(P/n, $\mathcal{D}$, $\mathcal{M}$)

Input: a node $P/n \in N$ in a form tree $\mathcal{FT} = (N, E, <_{sib}, root)$, a database $\mathcal{D}$, and a set of mappings $\mathcal{M}$

Output: the updated database $\mathcal{D} = (I, R, \Sigma)$ and the set of updated mappings $\mathcal{M} = \{M_i$: $\{\mathcal{FT}$:P/e $\rightsquigarrow$ $\mathcal{D}$:D.d$\}$ with $\{\mathcal{D}$:T$_i$.A$_i$ = $\mathcal{D}$:T$_j$.B$_j\}\}$ between $\mathcal{FT}$ and $\mathcal{D}$

Steps:

create a table T_n with name $\mu(n)$ and a key column id; /* $\mu(n)$ *returns a system acceptable term as a schema element. A mapping between the system-generated term and the original label is inserted in the catalog table* T_{meta}. */

let $T_{n_j} \in \mathcal{D}$ be the table corresponding to the parent n_j of n;

if n is the root which does not have a parent **then**

  add the correspondence $\mathcal{FT}$:$P/n \rightsquigarrow \mathcal{D}$:$T_n$ to the mapping;

  let $T_{n_j} = T_n$;

end if

for each child n_i of n, i.e., $n \rightarrow n_i \in E$ **do**

  if n_i is a text box and n_i is the only child of n **then**

    add a column $\mu(n)$ with data type $\tau(n)$ and constraint $\kappa(n)$ to T_{n_j}, remove T_n if $T_{n_j} \neq T_n$;

    add $\mathcal{FT}$:$P/n \rightsquigarrow \mathcal{D}$:$T_{n_j}.\mu(n)$ to the mapping;

    add $\mathcal{FT}$:$P/n/n_i \rightsquigarrow \mathcal{D}$:$t_{T_{n_j}}.\mu(n)$ to the mapping;

  else if n_i is a radio button with a value node n_k as child, let $\mu(n_k)$ be the value of the value node n_k **then**

    if $\mu(n)$ is not a column in T_{n_j} **then**

      add a f.k.column $\mu(n)$ to T_{n_j} referencing the id of T_n;

      add the predicate $\mathcal{D}$:$T_{n_j}.\mu(n) = \mathcal{D}$:$T_n$.id to the mapping;

    end if

    if T_n is just created **then**

      add a column option to T_n;

    end if

    insert $< autoid, \mu(n_k) >$ as a tuple to T_n;

    add $\mathcal{FT}$:$P/n/n_i/n_k \rightsquigarrow \mathcal{D}$:$t_{T_n}.\mu(n_k)_{option}$ to the mapping;

    if n_i has an extended child n_l **then**

      add a new column $\mu(n_l)$ to T_{n_j};

      add a new correspondence from n_l to the new column;

    end if

  else if n_i is a check box or a select list with a value node n_k as child, let $\mu(n_k)$ be the value of the

value node n_k **then**

if T_n is just created **then**

add a column option to T_n;

end if

insert $< autoid, \mu(n_k) >$ as a tuple to T_n;

add $\mathcal{FT}$:$P/n/n_i/n_k \rightsquigarrow \mathcal{D}$:$t_{T_n}.\mu(n_k)_{option}$ to the mapping;

create a new many-to-many table $T_{n_j n}$;

add two new columns to $T_{n_j n}$ referencing the id of T_{n_j} and T_n, respectively;

if n_i has an extended child n_l **then**

add a new column $\mu(n_l)$ to $T_{n_j n}$;

add a new correspondence from n_l to the new column;

end if

else if n_i is a text box **then**

add a column $\mu(n_i)$ with data type $\tau(n_i)$ and constraint $\kappa(n_i)$ to T_n;

add the correspondence $\mathcal{FT}$:$P/n/n_i \rightsquigarrow \mathcal{D}$:$t_{T_n}.\mu(n_i)$ to the mapping;

if n_j is the root */* inline the node n to the root so that the root is mapped to an n-ary relationship table */* **then**

add a f.k.column $\mu(n)$ to T_{n_j} referencing the id of T_n;

add the predicate $\mathcal{D}$:$T_{n_j}.\mu(n) = \mathcal{D}$:$T_n$.id to the mapping;

else {*/* map the edge between n_j and n to a many-to-many relationship */*}

create a new many-to-many table $T_{n_j n}$;

add two new columns to $T_{n_j n}$ referencing the id of T_{n_j} and T_n, respectively;

end if

else {*/* recursively create tables for descendants */*}

createTables($P/n/n_i, \mathcal{D}, \mathcal{M}$);

end if

end for

12.2 Merging Algorithm

Given a form tree $\mathcal{FT}$, a database $\mathcal{D}_e$, and a set of discovered correspondences $\mathcal{M}_d$, the merging algorithm aims to integrate the entire form into the database $\mathcal{D}_e$, and generates a complex semantic transformation mapping $\mathcal{M}_t$ between the form tree and the final database. The problem is at least as difficult as the problem of *schema mapping*[86;89] which takes as input a set of simple correspondences between two database schemas and infers complex semantic mappings between the two schemas. Because the same information can be structured differently in different database schemas, almost all the current solutions to schema mapping are semi-automatic requiring human intervention and examination.

In the context of mapping forms to databases, we first devise a forward engineering method for creating databases from form trees, i.e., the birthing algorithm. Then the merging problem is equivalent to discovering "equivalent" structures between the newly created database $\mathcal{D}_n$ and the existing database $\mathcal{D}_e$. In this process, the initial discovered and validated correspondences $\mathcal{M}_d$ provide information for linking atomic elements. Consequently, the entire mapping process (also illustrated in Figure 12.1) could be visualized in terms of the following steps:

1. Derive a new database $\mathcal{D}_n$ from the given form tree.
2. Shift all discovered correspondences $\mathcal{M}_d$ from the form tree to $\mathcal{D}_n$.
3. Analyze the discovered correspondences for their fitness for merging.
4. Extend the existing database $\mathcal{D}_e$ for unmergeable and unmapped elements.

In this section, we focus on the third step of the process, also known as the merging algorithm.

Premises

When developing the merging algorithms, we consider the compliance of the extended database with the quality and optimization principles. The goal is not just to automatically evolve a database with respect to a new form, but also to generate a database comparable to expert-designed systems, which accurately reflect the domain requirements and are optimized for storage and query. Intuitively, we

expect that the form tree is *correctly* $\mathcal{P}1$ and *completely* $\mathcal{P}2$ mapped to the final database. The database should not contain redundant elements ($\mathcal{P}3$) and should be normalized with regard to the standard database design principles. In addition, the final database should be optimized in terms of data storage and query processing. With these guidelines, during the merging process, we analyze each correspondence for its fitness for merging, primarily focusing on redundancy($\mathcal{P}3$) and normalization($\mathcal{P}4$), and secondarily focusing on the optimization principle of minimizing the NULL values($\mathcal{P}6$).

When merging a new database with an existing database, with the discovered correspondences as anchors, 4 key scenarios are encountered.

1. When a table T_n in the new database $\mathcal{D}_n$ is discovered to correspond to a table T_e in the existing database $\mathcal{D}_e$.

2. When a column c_n in the new database $\mathcal{D}_n$ is discovered to be correspond to a column c_e in the existing database $\mathcal{D}_e$, and the respective tables do not correspond to each other.

3. When a table T_n in the new database $\mathcal{D}_n$ is discovered to correspond to column c_e in the existing database $\mathcal{D}_e$.

4. When a column c_n in the new database $\mathcal{D}_n$ is discovered to correspond to a table T_e in the existing database $\mathcal{D}_e$.

In the next subsections, we describe each scenario in depth. Before that, we now describe certain heuristics which are common to all.

- If a table (new or existing) contains a not null descriptive column which is not a part of the discovered correspondences, then the table is kept intact and not merged with any other table. This, however, induces some redundancy in the evolved database.

- Each merger involves a trade-off between redundancy and potential for having null values, i.e., between the principles $\mathcal{P}3$ and $\mathcal{P}6$. We define the following two terms to establish the trade-off.

  - *Quality Tuning Factor:* A user-configurable value that indicates the weightage given to the quality, in particular to minimize redundancy.
  - *Null Value Ratio:* A calculated value that indicates the potential of having NULL values in a given table in the final database.

- A discovered correspondence may link two elements with different data types and constraints. Each column of a table has a data type and constraints indicating whether the values are unique or NULL-allowed. We use the following functions to get the data type and constraint information of a column c, respectively: $\mathsf{dataType}(c)$, $\mathsf{isUnique}(c)$, and $\mathsf{isNull}(c)$. The elements with conflicting data types and constraints should not be merged.

- The transformation correspondences for the current form (and any other affected form) are modified accordingly.

Scenario 1: Table-Table Merger

This merging scenario is illustrated in the Figure 12.7. We first describe the case wherein a regular table(i.e., a non-lookup table) is identified to be merged to another regular table as shown in the Figure 12.7a. If all the columns and the foreign keys in the two tables match, the tables are merged without further investigation. If there are any unmatched columns in either table, the algorithm decides whether (i) to generate a new table corresponding to T_n in the existing database, or (ii) to merge T_n into T_e by possibly adding new columns to it T_e. This decision reflects a trade-off between the quality and optimization principles. While the former option is likely to violate the compactness principle, $\mathcal{P}3$, the latter is likely to violate the optimization principle, $\mathcal{P}6$. The algorithm employs an user-defined *quality tuning factor(qtf)* ($0 \leq qf \leq 1$) to offer flexibility to maintain this trade-off; $qtf = 0$ favors $\mathcal{P}6$, $qtf = 1$ favors $\mathcal{P}3$. When merging two tables, the algorithm compares the value for qtf with the metric, *null value ratio(nvr)*, denoting the possibility of null value columns on merging the two tables. The *nvr* represents the ratio of the maximum number of columns(in either table) likely to have null values over the total of columns in the integrated table. If *nvr* falls below qtf, the algorithm merges the two tables, otherwise, it keep them separate. The upper half of Figure 12.7

shows an example of a table merging situation with $nvr = 0.4$. Case (a) is an example of a high value for qtf, that results in a compact database but may lead to null values while data collection using the column maritalstatus. Case (b) exemplifies the contrary situation. The Algorithm 12.2 formally describes this scenario.

ALGORITHM 12.2: mergeTables($\mathcal{FT}$,$\mathcal{D}_{new}$, $\mathcal{D}_{exist}$, $\mathcal{M}_{disc}$)

Input: a form tree $\mathcal{FT}$, a new database $\mathcal{D}_{new}$ derived from a form tree $\mathcal{FT}$, an existing database $\mathcal{D}_{exist}$, and an initial discovered mapping $\mathcal{M}_{disc}$ between the two databases

Output: an extended database $\mathcal{D}_{exist}$, a transformation mapping $\mathcal{M}_{trans}$ between the form tree $\mathcal{FT}$ and the extended database $\mathcal{D}_{exist}$

Steps:

1: let T_{meta} containing mappings from labels of form tree nodes, λn, to system-created element names of the database schema, $\mu(n)$;
2: let $\mathcal{M}_{trans}$ be an empty set of transformation correspondences;
3: **for** each table $\mathsf{T}_{new} \in \mathcal{D}_{new}$ **do**
4: **if** T_{new} is not covered by the discovered mapping $\mathcal{M}_{disc}$ **then**
5: extract the element labels $\lambda(n)$ in the form tree which correspond to the table and column names of T_{new};
6: create a new table T'_{new} in $\mathcal{D}_{exist}$ corresponding to T_{new}; the table and column names of T'_{new} are generated by the $\mu(n)$ function from the element labels corresponding to the table and column names of T_{new};
7: add a table name mapping between $\mu(n)$ and $\lambda(n)$ to the table T_{meta};
8: add the correspondences from $\lambda(n)$ in $\mathcal{FT}$ to the new table $\mathsf{T}'_{new} \in \mathcal{D}_{exist}$ to $\mathcal{M}_{trans}$;
9: **else if** $\mathsf{T}_{new}(a_1, a_2, ..., a_n)$ is mapped to a table $\mathsf{T}_{exist}(b_1, b_2, ..., b_m) \in \mathcal{D}_{exist}$ **then**
10: let $\mathsf{T}_{new} \rightsquigarrow \mathsf{T}_{exist}, a_1 \rightsquigarrow b_1, ..., a_h \rightsquigarrow b_h$ be the discovered correspondences;
11: **if** $\forall i \in \{h+1, .., m\}$, $\mathsf{isNull}(b_i)$ is true and $\frac{(m-h)+(n-h)}{h} < qf$ $(m - h = 0$ for $m < h)$ **then**
12: **for** a dangling column $\mathsf{a}_i \in \mathsf{T}_{new}, i \in \{h+1, ..., n\}$ **do**
13: extract the element label $\lambda(n)$ corresponding to a_i from the form tree $\mathcal{FT}$;
14: add a new column $\mu(n)$ in the table $\mathsf{T}_{exist} \in \mathcal{D}_{exist}$;
15: add a mapping between $\mu(n)$ and $\lambda(n)$ to the table T_{meta};

16: add the transformation correspondence between the form element, corresponding to a_i, to the new column $\mu(n) \in \mathsf{T}_{exist}$ to $\mathcal{M}_{trans}$;

17: **end for**

18: **else**

19: add a new table T_{new}'(a_1, a_2,...,a_n) in $\mathcal{D}_{exist}$ corresponding to $\mathsf{T}_new(a_1, a_2, ..., a_n)$;

20: **end if**

21: **end if**

22: **end for**

23: return $< \mathcal{D}_{exist}, \mathcal{M}_{trans} >$;

There is yet another case wherein both the tables to be compared are lookup tables as shown in the Figure 12.7b. In this case the algorithm compares the values of both the tables and if more than 2 values are found to be matching then the lookup tables are merged to ensure compactness.

Scenario 2: Column-Column Merger between Different Tables

This merging scenario is illustrated in the Figure 12.8 wherein certain columns belonging to different tables are discovered to be equivalent to each other. In this scenario, the algorithm decides whether (i) to keep the two columns separately in different tables, or (ii) to merge the corresponding columns into the existing table and link the two tables through a foreign key reference. Like the previous scenario, this decision reflects a trade-off between the quality and optimization principles. While the former option is likely to violate the compactness principle, $\mathcal{P}3$, the latter is likely to violate the optimization principle, $\mathcal{P}6$.

The algorithm breaks the tie using the values of the *quality tuning factor(qtf)* and the *null value ratio(nvr)*. In this case, the value of *nvr* denotes the possibility of having null value columns on merging the columns. The *nvr* represents the ratio of the number of non-matching columns over the total number columns in the existing table. If *nvr* falls below qtf, the algorithm merges the columns, otherwise, it keeps them separate. The upper half of Figure 12.8 shows an example of a table merging situation with $nvr = 0.5$. Case (a) is an example of a higher value for qtf, that results in a compact database but may lead to null values in the chiefcomplaints and the HPI columns from

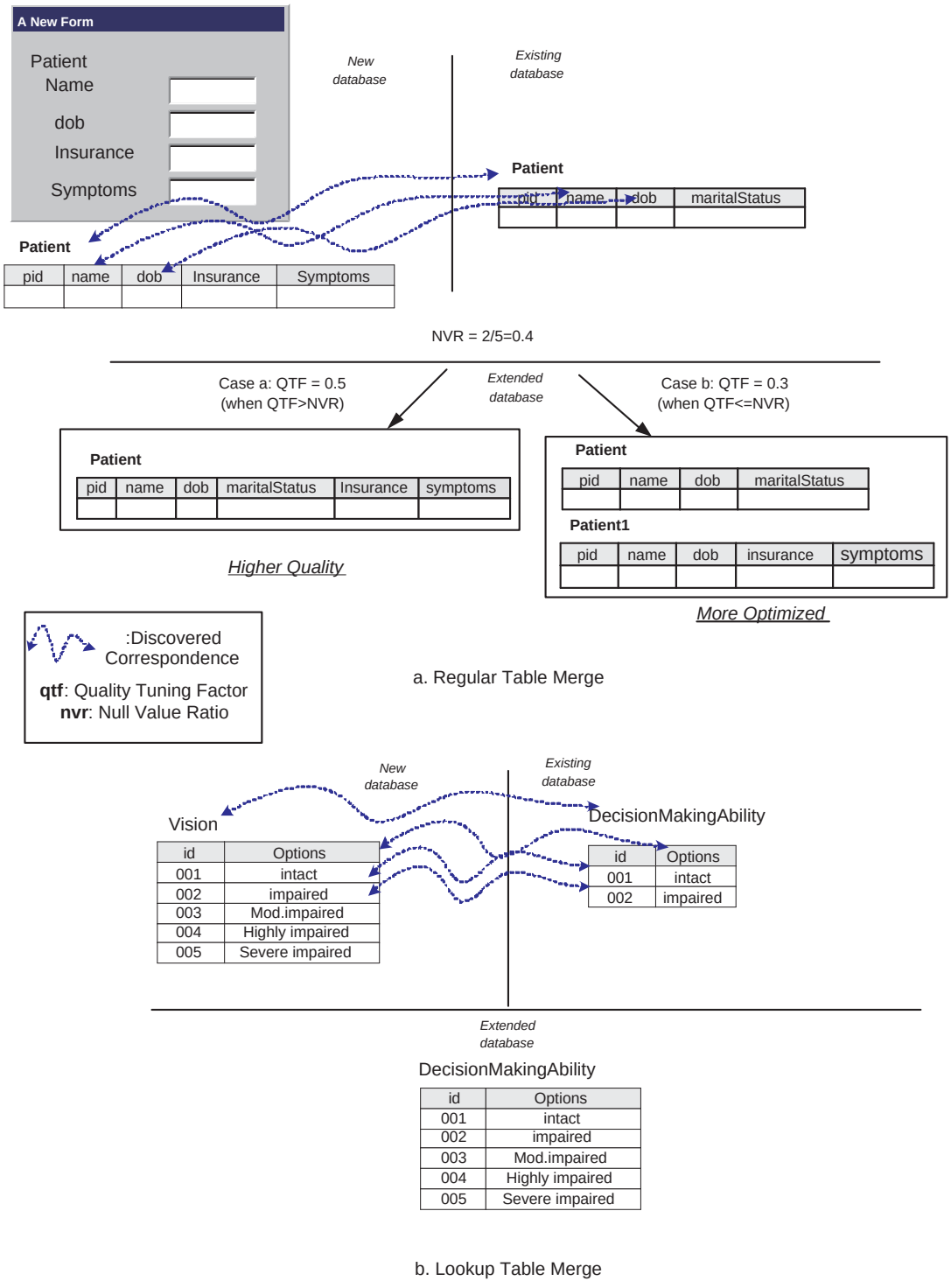


Figure 12.7: Merging Scenario 1: Table to Table

the History table, while collecting data using the form mapped to the Diagnosis table. Case (b) exemplifies the contrary situation.

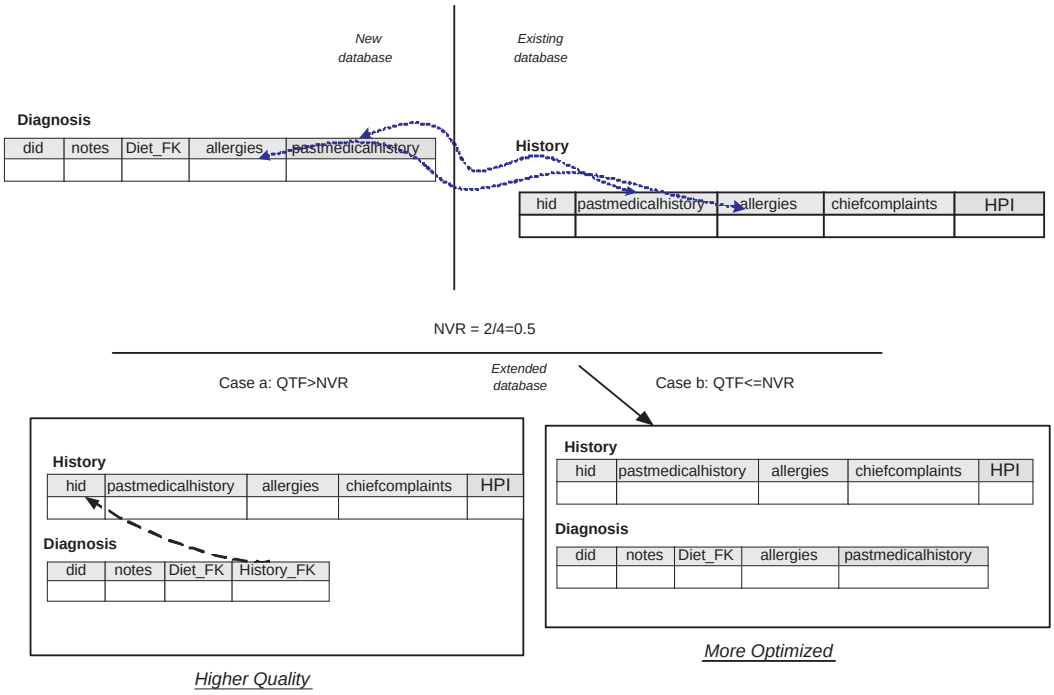


Figure 12.8: Merging Case 2: Column to Column (Different Tables)

Scenario 3: Table-Column Merger

This merging scenario is illustrated in the Figure 12.9 wherein a table in the new database is discovered to correspond to a column in the existing database. This scenario could be further classified in terms of whether the new table is a regular table or a look-up table associated with an extended radiobutton.

Let us consider the first case shown in the Figure 12.9a. In this scenario, the algorithm decides whether (i) to keep the matching column in its original container table, or (ii) to transfer the column into the new table and link the tables via foreign key reference. Like the previous scenario, this decision reflects a trade-off between the quality and optimization principles. While the former option is likely to violate the compactness($\mathcal{P}3$) and the normalization ($\mathcal{P}4$) principles, and the latter is likely to violate the optimization principle, $\mathcal{P}6$. The algorithm breaks the tie using the values of the qtf and the nvr. In this case, the value of nvr denotes the possibility of having null value columns in the new table on transferring the column in this table. The nvr represents the ratio of

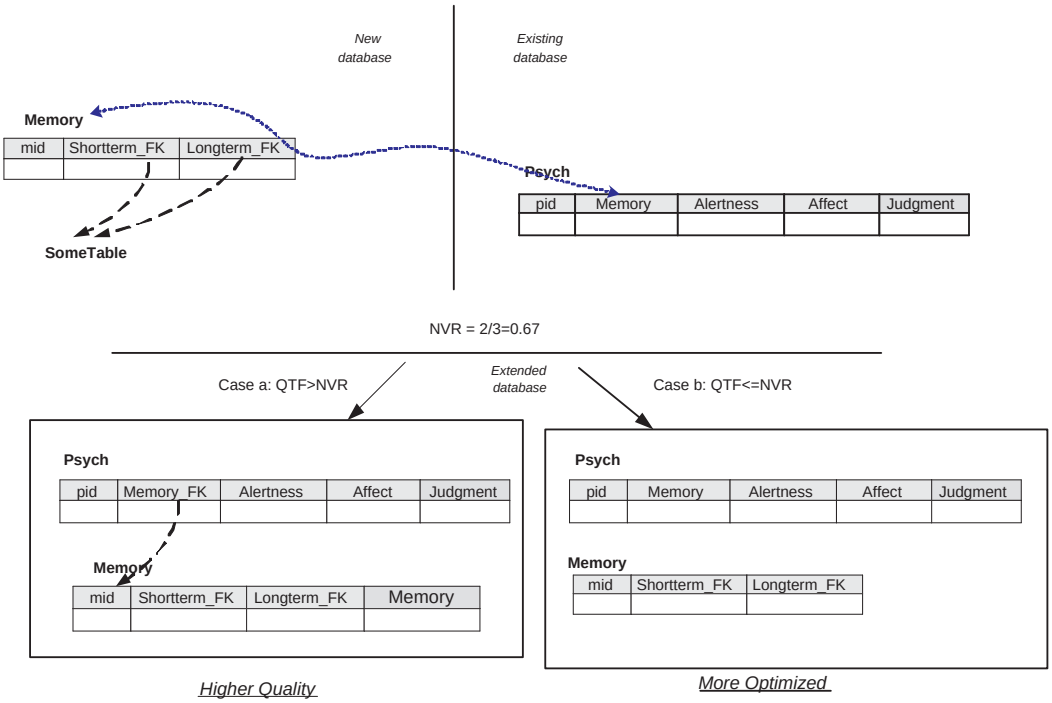
the number of non-matching columns in the new table over the number of columns in the new table after the merger. If nvr falls below qtf, the algorithm transfers the column to the new table, else, no change is made. The upper half of Figure 12.9a shows an example of this merging situation with $nvr = 0.67$. Case (a) is an example of a higher value for qtf, that results in a compact and more normalized database but may lead to null values in the shortterm_FK and the longterm_FK columns from the Memory table, while collecting data using the form mapped to the Psych table. Case (b) exemplifies the contrary situation.

Similarly for the second case shown in the Figure 12.9b, the algorithm decides whether (i) to keep the elements separately, or (ii) to merge the supporting text column with the existing column and link via foreign key reference. This decision reflects a trade-off between the quality and optimization principles. While the former option is likely to violate the compactness($\mathcal{P}3$) and the normalization ($\mathcal{P}4$) principles, and the latter is likely to violate the optimization principle, $\mathcal{P}6$. The algorithm breaks the tie using the values of the qtf and the nvr. In this case, the value of nvr denotes the possibility of having null value columns in the existing table if the merger is executed. The nvr represents the ratio of the number of non-matching columns in the new table over the number of columns in the new table after the merger. If nvr falls below qtf, the algorithm transfers the supporting text column to the existing table, else, no change is made. The upper half of Figure 12.9b shows an example of this merging situation with $nvr = 0.67$. Case (a) is an example of a higher value for qtf, that results in a compact and more normalized database but may lead to null values in the rest of the columns in the table T2, while collecting data using the form mapped to the Smokes table. Case (b) exemplifies the contrary situation.

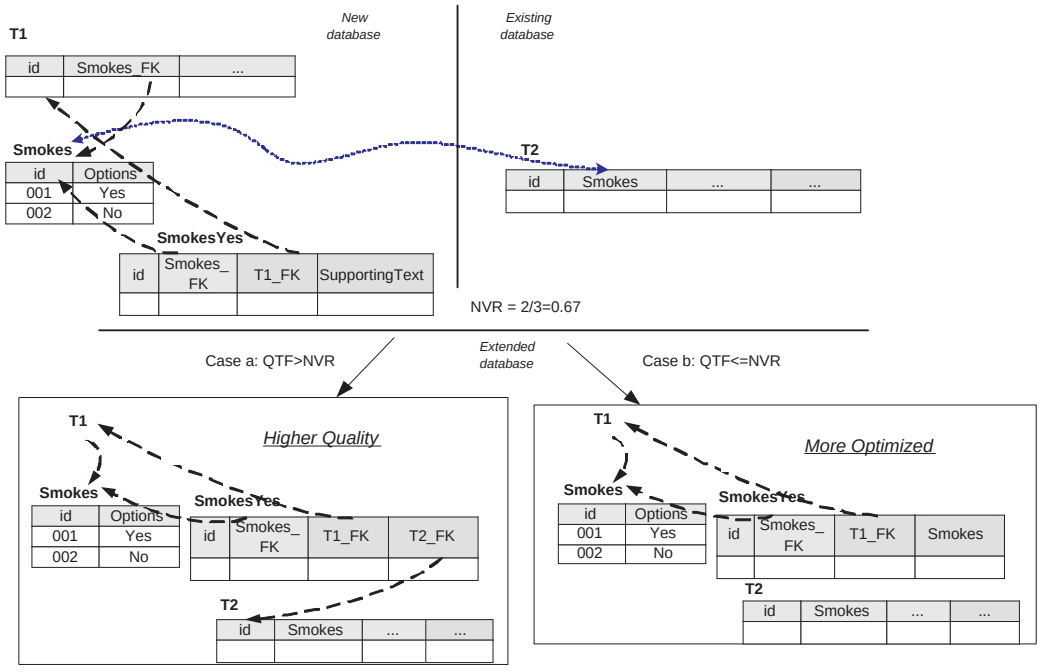
Scenario 4: Column-Table Merger

This merging scenario is illustrated in the Figure 12.10 wherein a column from the new database is discovered to correspond to a table in the existing database. This scenario could be further classified in terms of whether the existing table is a regular table or a look-up table associated with an extended radiobutton.

Let us consider the first case shown in the Figure 12.10a. In this scenario, the algorithm decides



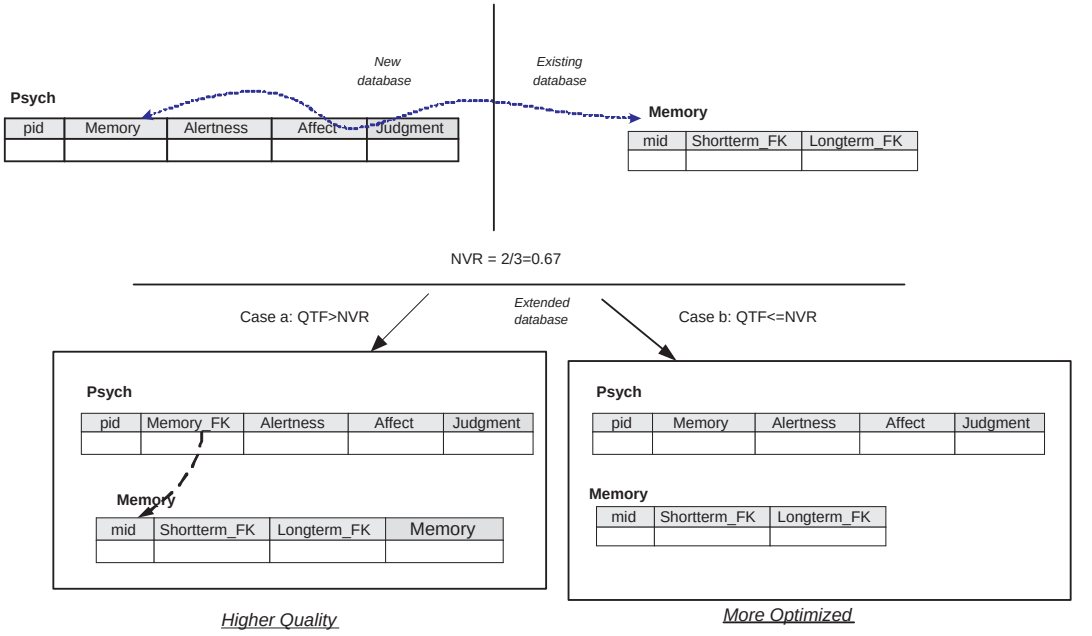
a. Regular Table To Column Merge



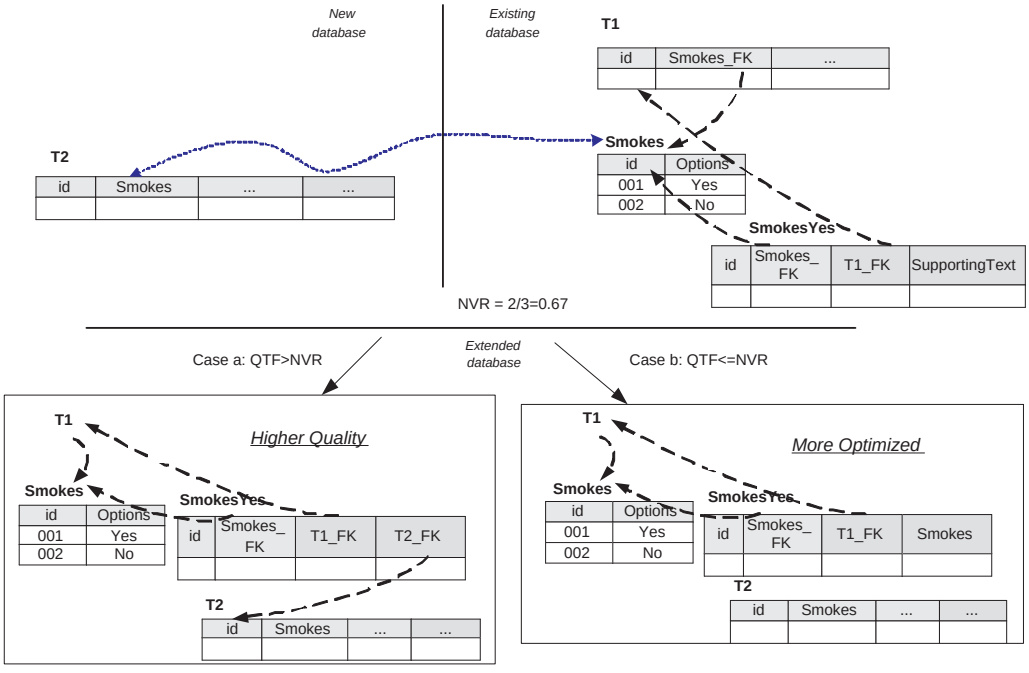
b. Lookup Table(associated with extended Radiobutton) To Column Merge

Figure 12.9: Merging Case 3: Table to Column

whether (i) to keep the matching column in its original container table, or (ii) to transfer the column into the existing table and link the tables via foreign key reference. This decision reflects a trade-off between the quality and optimization principles. While the former option is likely to violate the compactness($\mathcal{P}3$) and the normalization ($\mathcal{P}4$) principles, and the latter is likely to violate the optimization principle, $\mathcal{P}6$. The algorithm breaks the tie using the values of the qtf and the nvr. In this case, the value of nvr denotes the possibility of having null value columns in the existing table on transferring the column in this table. The nvr represents the ratio of the number of non-matching columns in the existing table over the number of columns in the existing table after the merger. If nvr falls below qtf, the algorithm transfers the column to the existing table, else, no change is made. The upper half of Figure 12.10a shows an example of this merging situation with $nvr = 0.67$. Case (a) is an example of a higher value for qtf, that results in a compact and more normalized database but may lead to null values in the shortterm_FK and the longterm_FK columns from the Memory table, while collecting data using the form mapped to the Psych table. Case (b) exemplifies the contrary situation. Similarly for the second case shown in the Figure 12.10b, the algorithm decides whether (i) to keep the elements separately, or (ii) to merge the new column with the existing supporting text column and link via foreign key reference. This decision reflects a trade-off between the quality and optimization principles. While the former option is likely to violate the compactness($\mathcal{P}3$) and the normalization ($\mathcal{P}4$) principles, and the latter is likely to violate the optimization principle, $\mathcal{P}6$. The algorithm breaks the tie using the values of the qtf and the nvr. In this case, the value of nvr denotes the possibility of having null value columns in the new table if the merger is executed. The nvr represents the ratio of the number of non-matching columns in the new table over the number of columns in the new table after the merger. If nvr falls below qtf, the algorithm transfers the existing supporting text column to the new table, else, no change is made. The upper half of Figure 12.10b shows an example of this merging situation with $nvr = 0.67$. Case (a) is an example of a higher value for qtf, that results in a compact and more normalized database but may lead to null values in the rest of the columns in the table T2, while collecting data using the form mapped to the Smokes table. Case (b) exemplifies the contrary situation.



a. Column To Regular Table Merge



b. Column to Lookup Table(associated with extended Radiobutton) Merge

Figure 12.10: Merging Case 4: Column to Table

Part IV

Evaluation

Chapter 13: Objectives

The goal of the proposed framework is to ensure that the evolved databases are principle compliant, and to minimize the number of user interventions required to carry out the mapping process. The first goal of the evaluation study is to determine how well the systems meets the compliance and intervention goals. Another goal is to assess the impact of each module of the framework in accomplishing the two goals. We seek to evaluate several aspects of the framework as illustrated in Figure 13.1 and as enlisted below.

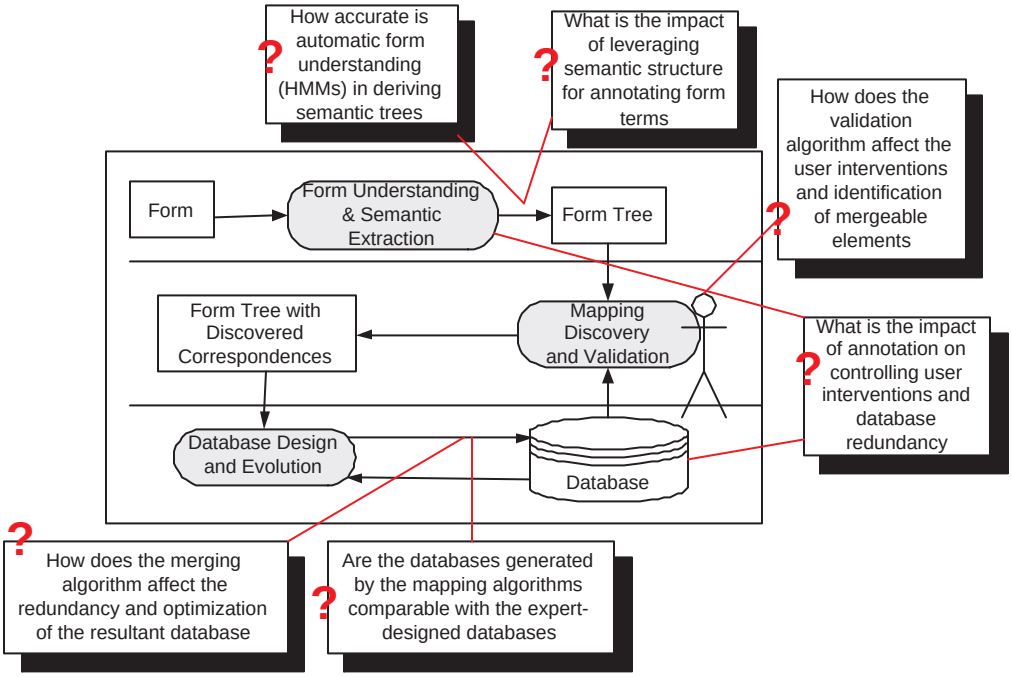


Figure 13.1: Evaluation Goals

1. Principle Compliance Goal:

   - How accurate is the automatic form understanding approach in deriving semantic trees?
   - How well does the validation algorithm facilitate identification of mergeable elements?
   - How do the merging and birthing algorithms affect the redundancy and optimization of

the resultant database?

- Does annotation help control database redundancy?
- Are the databases generated by the birthing and the merging algorithms comparable with the expert-designed databases?

2. User Intervention Goal:

- How does the validation algorithm affect the user interventions?
- What is the impact of annotation in controlling user interventions?

Chapter 14: Data and Gold Standard Benchmarks

We conduct the experiments in the healthcare domain wherein the usage of forms is very prevalent, and the information systems are quite unusable from an integration point of view. The data used for the experiments are described in Tables 14.1 and 14.2. This includes 52 highly complex and lengthy forms actively used for patient data collection in 6 medical institutions. The forms from each institution(i.e. belonging to one dataset) were inter-related and had overlapping elements. The forms, not available in the HTML format, were manually converted. These datasets are the primary input to all the experiments conducted in this dissertation work. Some of these forms are illustrated in the Appendix Section A, Figures A.1 through A.10. To facilitate the evaluation of experiment results, we rely on the following benchmark datasets.

.

1. *Gold Standard Trees:* In order to evaluate the accuracy of the form tree extracted using the automated form understanding algorithms, we needed a dataset that contains the "equivalent" trees for the input forms. We prepared this dataset using the form design interface described earlier in Chapter 3, Section 10.1.1, that allows users to build a form while specifying various containment relationships. The system captures the semantic intentions of the users on the fly, and produces an accurate semantic tree for any form being designed. This gold dataset has 52 form trees in all, i.e., a gold tree corresponding to each form in the input dataset.

Table 14.1: Experiment Datasets

No.	**Source**	**Tot. Forms**
1	Walk in Clinic Encounter Forms[90]	3
2	Nursing Patient Admission Forms[91]	6
3	Labor and Delivery Data-entry Forms[92]	7
4	Adult Visit Encounter Forms[93]	18
5	Family Practice Forms[94]	13
6	Child Visit Encounter Forms[93]	5
All		52

Table 14.2: Experiment Datasets Descriptions

No.	Avg. Label	Avg. Input	Total Terms	Mappability	Avg. Interventions
1	32.33	49.33	161	75.77	3.67
2	17.17	33	261	63.98	1.5
3	16.14	37.29	294	56.80	0.14
4	47.83	65.22	1603	56.20	13.5
5	82.61	100.46	1519	59.38	17.23
6	53	67.4	397	62.21	10.4
All	48.32	65.85	4234	59.17	10.4

2. *Gold Standard SNOMED CT Annotations:* The input forms contain 4235 crude terms(or phrases) supplied by the clinicians when the forms were originally designed. We manually identified a SNOMED CT concept, corresponding to each form term and stored the concepts as the SNOMED CT gold standards corresponding to each form. We found that not all the terms were mappable, i.e., relevant with respect to SNOMED CT. This mostly included the terms such as "no scleral icterus" and "chronic back pain," that correspond to the post-coordinated concepts, and the terms such as "follow up with PCP" and "sent to ER," that partially correspond to certain concepts. The *mappability* of the forms, i.e., the percentage of the relevant terms found in the forms, is shown in the fourth column of Table 14.2. Overall, 2506 (i.e., 59.17%) of the terms are mappable.

3. *Gold Standard Databases:* In order to evaluate the accuracy of the databases generated using the birthing and merging algorithms, we needed a dataset that contains the "ideal" databases that would be potentially used to store the information collected using the given set of forms. Coming up with this gold dataset was challenging. In the quest for the compiling ideal databases for the given forms, we separately consulted with two database experts each with at least 10 years of design experience. For the convenience of the experts, we provided only 3 datasets containing 16 forms . After hours of careful analysis and multiple iterations, the experts produced a database corresponding to each of the 3 sets of forms. Since the databases were very large, the experts did not specify the mappings and only specified the database schemas. We compiled these databases into our gold dataset, which contains two gold stan-

dards, each having 3 databases corresponding to the 3 sets of forms. A portion of one of the gold databases is illustrated in Appendix Section B Figure B.1.

Chapter 15: Experiment Prototype & Settings

The entire framework is implemented as a Web-based application running on an IBM x3400 server with 8 GB memory. The implementation technologies and platform include Java, AJAX, Tomcat, and MySQL server. For the experiments, we tuned each module of the framework to align with the theoretical and practical goals. The following sections describe the details of the settings adopted for each module.

15.1 DIY Form Design Module

The forms had several categories and subcategories, indicating the prevalence of form patterns 4 and 5 (Figure 12.5). Some preliminary testing suggested clear violation of the principle $\mathcal{P}7$ as several unwanted join tables get created in the absence of precise information on cardinalities. To address this, we added a user interaction to the mapping module. An example interaction is shown in Figure 15.1 where the user is given choices on various cardinality combinations.

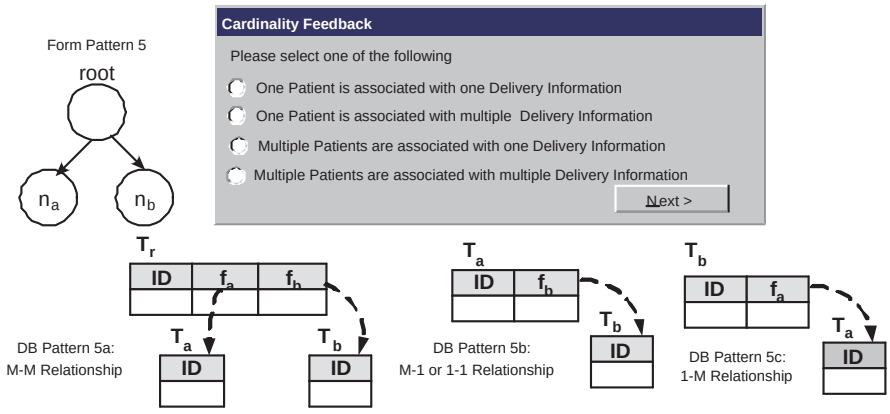


Figure 15.1: User Interaction and More Patterns

15.2 The Automatic Tree Generation Module

To carry out the experiments, we trained and decoded the employed HMMs using the Expectation Maximization and the Viterbi algorithms[95], respectively. The employed dynamic algorithms are optimized using memoization[73], thus, providing a time-efficient solution. The training data comprises

Table 15.1: Extraction Accuracy (%) for the T_HMM States

Model	q_0^T	q_1^T	q_2^T	q_3^T	q_4^T	q_5^T
Baseline HMMs	83.71	85.29	89.84	77.53	83.17	75.28
Aligned HMMs	90.26	91.74	93.71	78.65	88.91	82.54

the gold standard trees. We fine tune the HMMs based on our findings from the previous work[44] that states that: *a given set of forms could be most accurately understood by a model trained on the data having appropriate variety and frequency of design patterns.* We therefore design a model that is aligned with the hierarchical pattern of the form to be tested. We train the model using the training data whose hierarchical complexity matches with that of the form to be tested. We use the leave-one-out cross validation method for training. We develop two versions of the models HMM_{less} (trained on 27 forms) suitable for extracting form trees with the maximum height ranging from 1 through 5 and HMM_{more} (trained on 25 forms) suitable for extracting the trees with a maximum height of 6 or more. We train the model using the training data whose hierarchical complexity matches with that of the form to be tested. For each form in the input dataset, we choose the appropriate HMM training model, and run the tree generation algorithm to derive the tree structure.

Table 15.1 shows the accuracy of extraction of key states for the regular and the aligned models. This clearly demonstrates the advantages of aligning the model with respect to the hierarchical characteristics of the input form and confirms our earlier conclusion[44].

15.3 The Annotation Module

To train the semantic structure-based `sClassifier`, we used leave-one-out cross validation across the terms belonging to a particular dataset. The classification training data comprises the gold standard SNOMED CT annotations. We heuristically chose the value of k, i.e., the number of top classes to be considered for prediction, as 4. As the API module, we used *SnAPI*, a product provided by the Dataline Software Ltd[80]. In terms of the underlying linguistic techniques, *SnAPI* is the programmatic equivalent of the *Snoflake* browser introduced earlier in Section 10.2. We

heuristically chose the threshold for the match-weight function adopted by the *SnAPI* as 0.2.

15.4 Mapping Discovery

Given the large-scale of the forms, and hence, that of the potential databases, we adopted Lucene indexing[85] in the merging module. In particular we prepared three indexes: table index, column index, and lookup value index. To accomplish the exact concept matching when using the annotated form trees, we maintained 3 tables that contain information about all the table names, column names, and lookup value names, along with their respective SNOMED CT concept ids.

15.5 Birthing and Merging Algorithms

Based on the new cardinality disambiguation component, we modified the birthing algorithm to minimize the creation of an extra table; the new solutions, i.e, patterns 5a, 5b, and 5c, are shown in Figure 15.1. In the merging algorithm, we arbitrarily set the quality tuning factor to 0.7.

Chapter 16: Experimental Design

To evaluate the entire framework, we designed 3 main experiments. The first two experiments were designed to study the performances of the form tree extraction module, and the term annotation module, respectively. The third experiment was to designed to study the process of mapping forms to existing databases.

16.1 Experiment 1: Automatic Form Understanding

The first experiment is a simple experiment designed to assess the performance of the automatic technique for extracting semantic form tree from an arbitrarily designed form. This is illustrated in Figure 16.1. Through this experiment, we measure the accuracy of the extracted form tree with respect to the respective gold form tree.

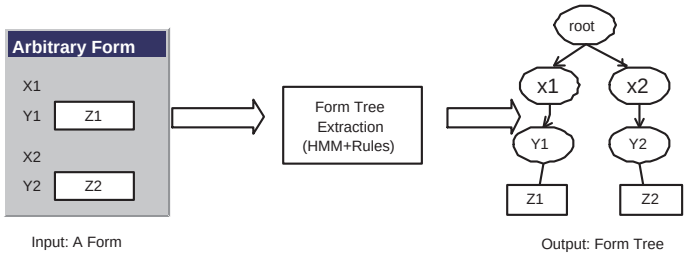


Figure 16.1: Experiment 1: Automatic Extraction of Semantic Form Trees

16.2 Experiment 2: SNOMED CT Annotation

The main goal of the second set of experiments was to study the impact of using semantic structure on the annotation performance. We conducted experiments using 3 versions of this experiment; with each version we increased the extent of the structural information utilized.

We first devised a **baseline** approach based on pure linguistics. Given a term, this approach uses the *SnAPI* general mapping functionality and maps the term to the most linguistically matching, i.e., the maximum match-weight, SNOMED CT concept. Next, we conducted experiments using the proposed **hybrid** approach. We further enhanced the structure based component of the **hybrid**

approach by expanding the candidate set of the semantic categories considered. In particular, we modified the category picker module such that it first retrieves the most linguistically matching concepts for all the top k classes; then, among the candidates, picks the maximum match-weight concept. We call this the **hybrid++** approach. These approaches are illustrated in Figures 16.2a, 16.2b, and 16.2c, respectively.

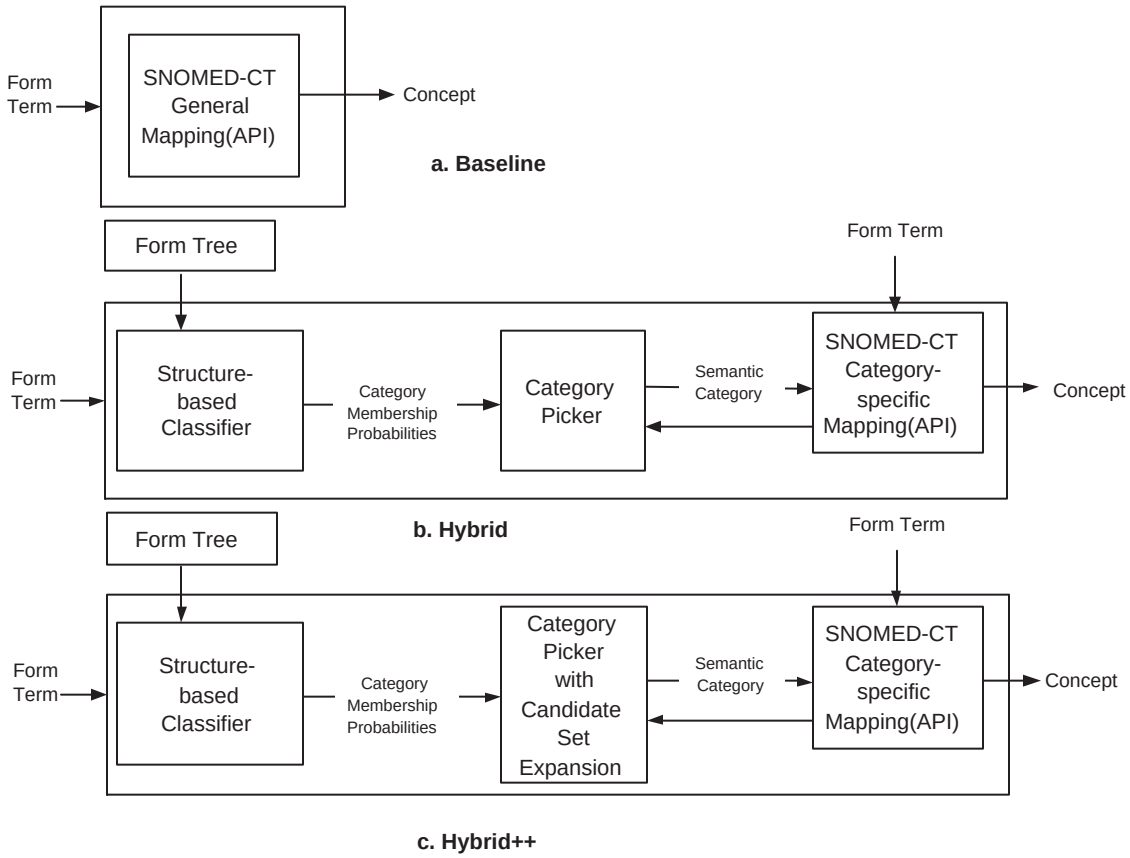


Figure 16.2: Experiment 2A: SNOMED CT Term Annotation

It was found that since *SnAPI* uses exact string matching as its underlying linguistic technique, the unsuccessful cases occurred because of string mismatch between the term and the concept descriptions. Hence, we added a term processing component that removes special characters (-,#, /, etc.) and performs acronym expansion using a dictionary of 103 frequently used clinical acronyms such as "T"(temperature), "BTL"(Bilateral Tubal Litigation), "VTE" (venous thromboembolism), etc. The dictionary is listed in the Appendix Section C, Tables C.1 through C.3. The revised design is illustrated in Figure 16.3.

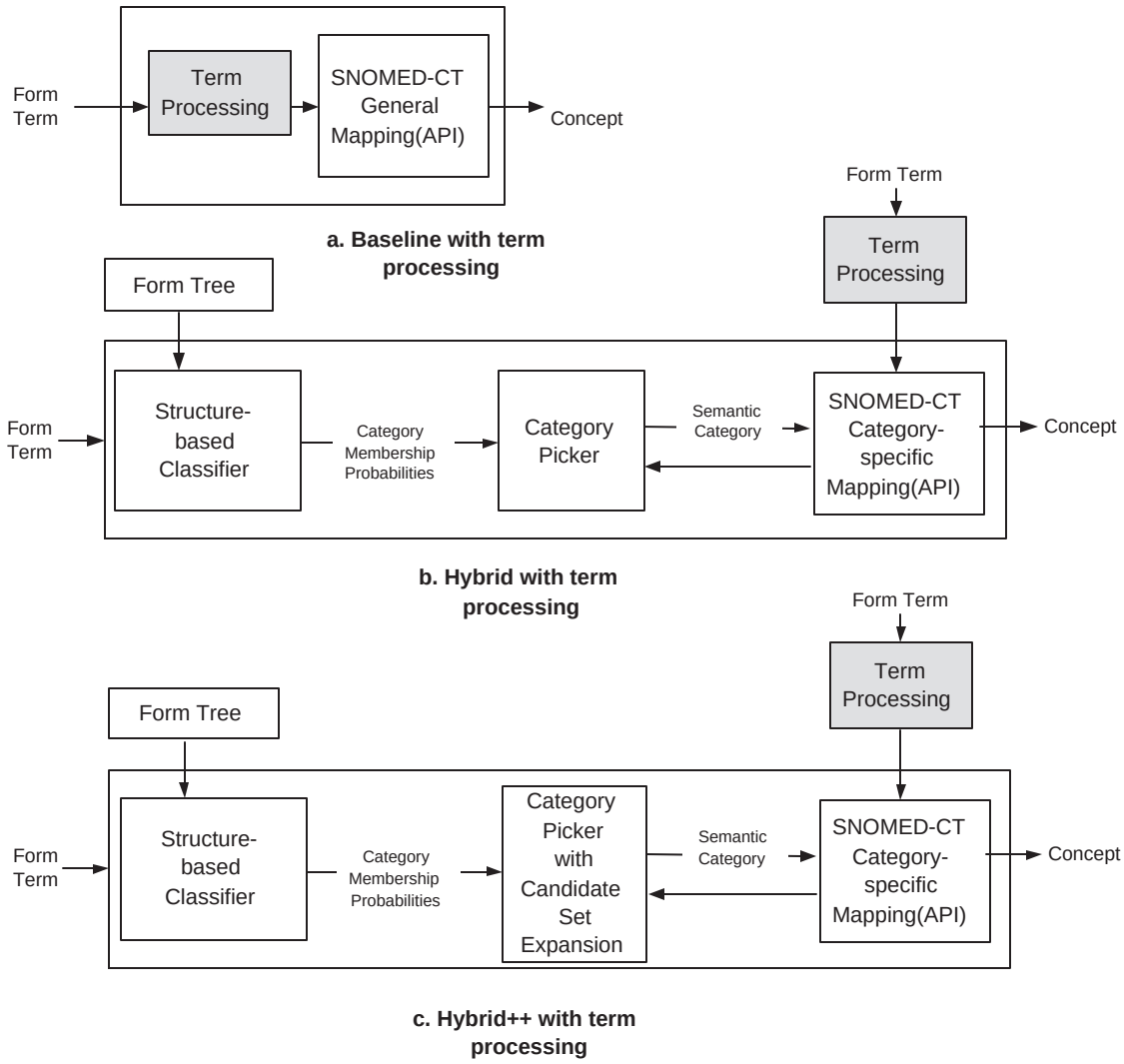


Figure 16.3: Experiment 2B: SNOMED CT Term Annotation with Term Processing

To assess the performance of the approach, we report the annotation precision and the annotation recall, wherein precision is the number of correct annotations over the total terms annotated by the system, and recall is defined as the number of correct annotations over the total number of gold annotations.

16.3 Experiment 3: Mapping Forms to Database

Next we designed the holistic experiments to evaluate the entire framework for mapping forms to databases. To test each dataset, we start with an empty existing database and incrementally map forms in a particular order to the existing database. The final output is the evolved database.

The first version of the mapping experiments is shown in Figure 16.4. This experiment begins with a form tree, and goes through the stages of correspondence discovery and validation. Then a new database is generated corresponding to the form tree, and all the validated correspondences are transferred to it. Finally, the new database is merged with the existing database using the merging algorithm. The correspondence discovery is performed based on linguistic matching between the form terms and the database element names. This is called as the **linguistic-based discovery** version.

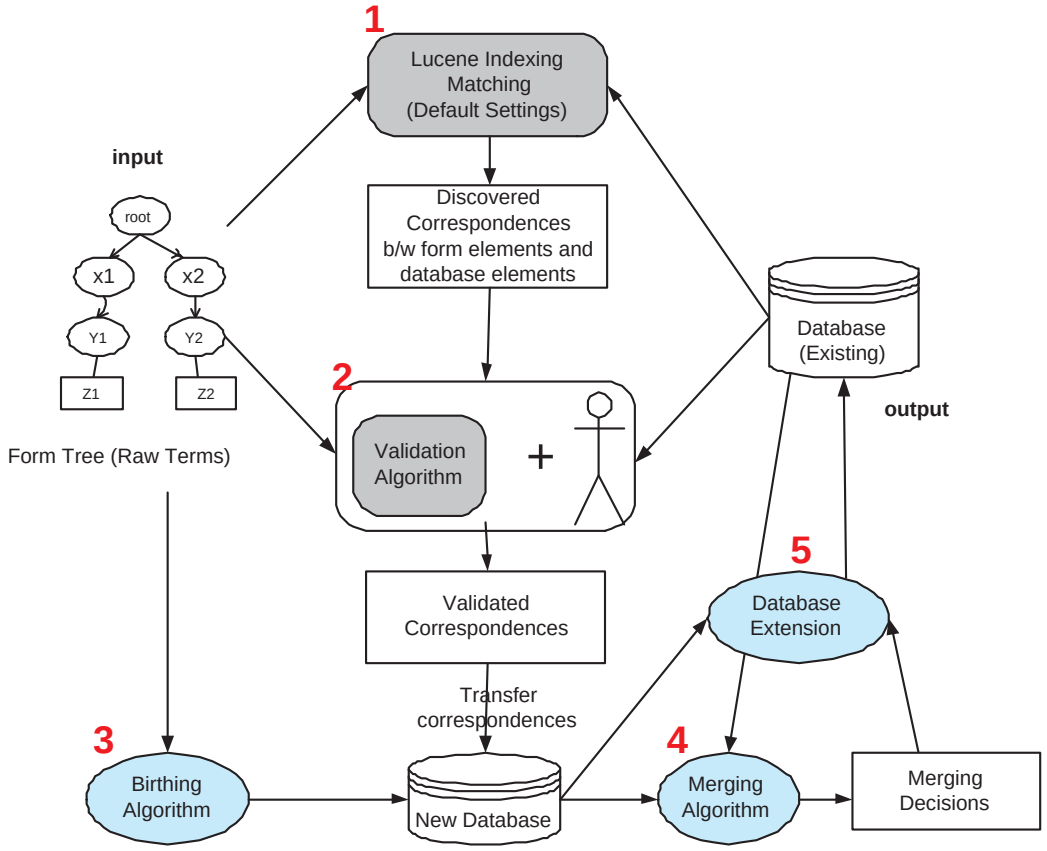


Figure 16.4: Experiment 3a: Linguistic-based Discovery

We conducted another round of experiments as shown in Figure 16.5. This experiment begins with an annotated form tree, and goes through the stages of correspondence discovery and validation. The discovery is performed using exact concept matching, and this version is hence called the **concept-based discovery** version. The rest of the experiment design remains the same. Finally, we conducted another set of experiments combining the methods employed by both the linguistic-based

and the concept-based versions. The experiment design is hybrid in nature in that it considers both the exact concept match and the linguistic match techniques for discovering the correspondences. The **hybrid discovery** version is summarized in Figure 16.6.

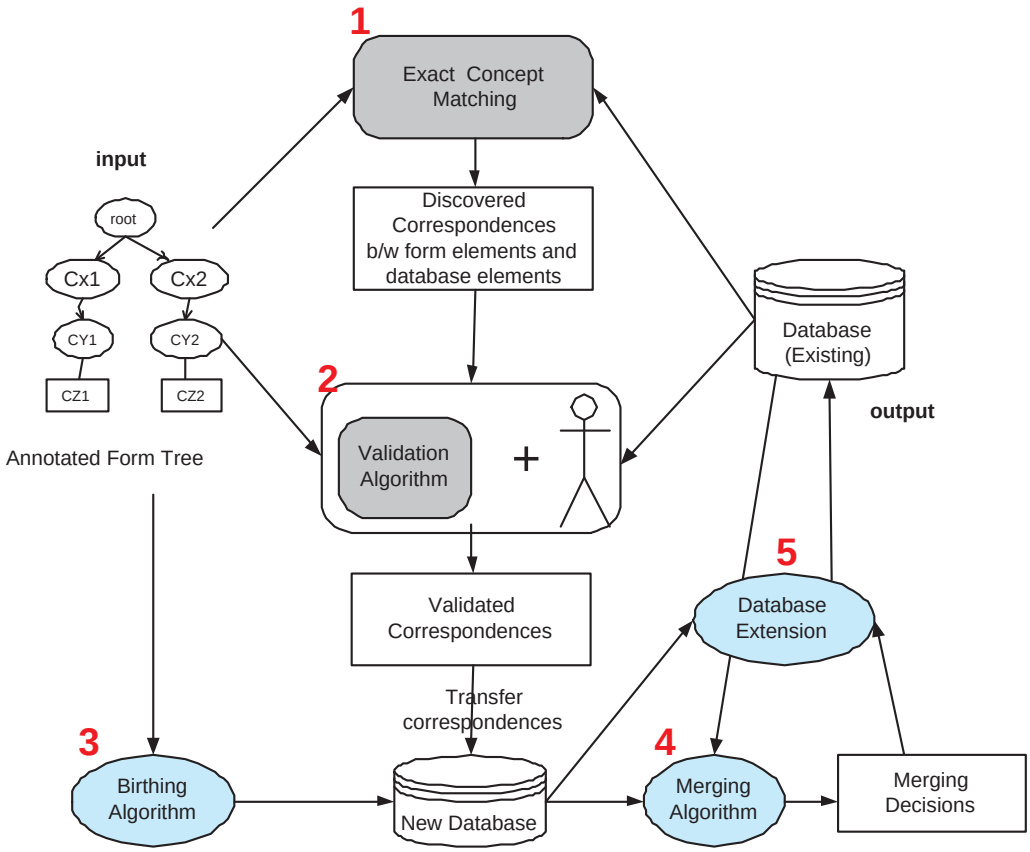


Figure 16.5: Experiment 3b: Concept-based Discovery

For these experiments, we report several general measures including the scale of the generated databases, and the duration of the mapping process. In terms of the goal of principle compliance, we report an approximation of the redundancy present in the evolved database, a comparison between the principle compliance of the framework evolved databases with that the expert designed databases, and the extent of annotation of the resultant databases. In terms of the intervention goal, we report the impact of using the validation algorithm on user interventions, the average number of user interventions required to validate correspondences for mapping a given form, the number of options presented on the validation screens, and the relevance of the validation screens presented to users.

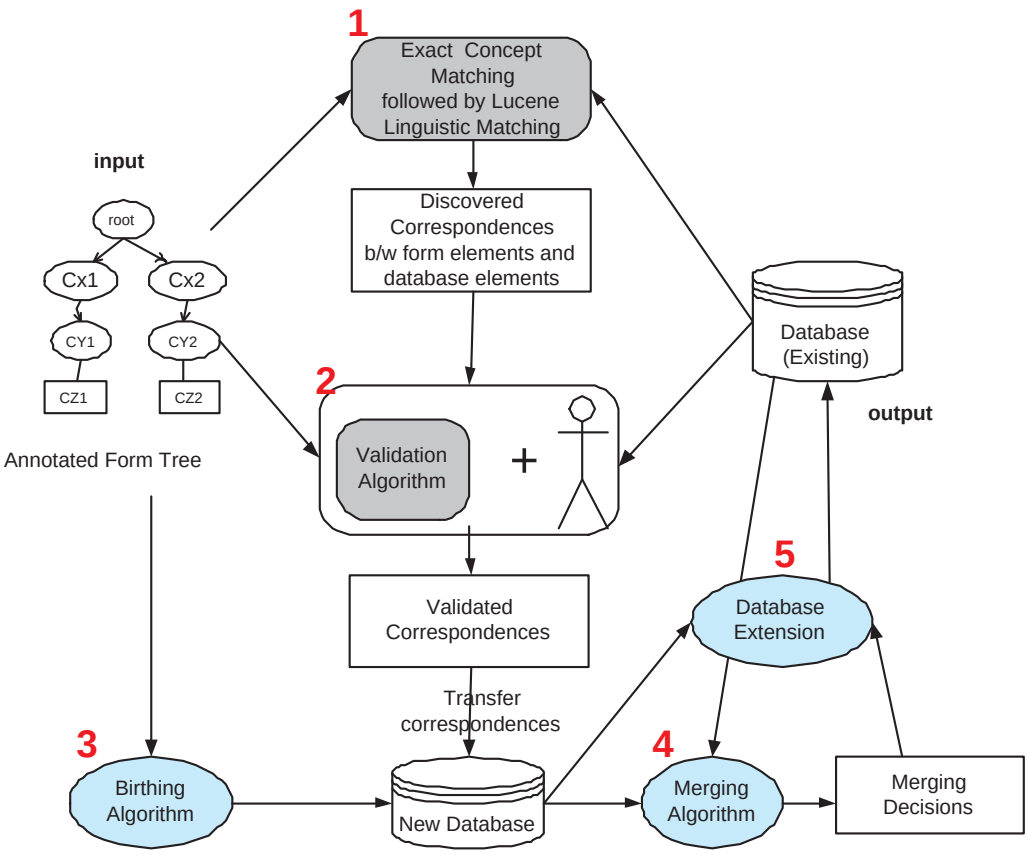


Figure 16.6: Experiment 3c: Hybrid Discovery

Chapter 17: Results and Findings

This chapter describes the results and the key findings of the all the experiments. Sections 17.1 through 17.3 describe the results of the three experiments. Section 17.4 summarizes the experiments, and draws implications.

17.1 Automatic Form Understanding

To conduct the first set of experiments, we used the tree extraction module to generate the form trees corresponding to the input set of 52 forms. To determine the accuracy of a given system generated tree, we compare it with its gold counterpart. Since, a form tree essentially represents the parent child associations(i.e. containment relationships), we compare the parent child edges between the two trees. On automatically comparing the edges of the two trees, we find that 97.85% of the parent-child associations are accurately captured by the extraction algorithm. The set-wise results are described in Table 17.1. Unsuccessful cases are due to the elements and segments misidentified by the HMM, resulting into an inaccurate association in the tree. Examples include associating a node with a wrong parent, or representing siblings as parent-child nodes or vice versa. The algorithm finished generating an average form tree, with 135 edges, in 0.08 seconds.

17.2 SNOMED CT Term Annotation

The **baseline** approach resulted into an average precision and an average recall of 0.60 and 0.46, respectively. Next, we conducted experiments using the proposed **hybrid** approach. Using this approach, the precision ranged from 0.69 through 0.89, and the recall ranged from 0.42 through

Table 17.1: Tree Extraction Results

.	*Dataset1*	*Dataset2*	*Dataset3*	*Dataset4*	*Dataset5*	*Dataset6*
Tot. Edges	272	362	461	2606	2674	644
Accuracy (%)	95.22	97.51	100	97.58	98.46	96.11

0.69, for all the datasets. For the **hybrid++** approach, the mapping precision ranged from 0.81 through 0.92, and the recall ranged from 0.51 through 0.74.

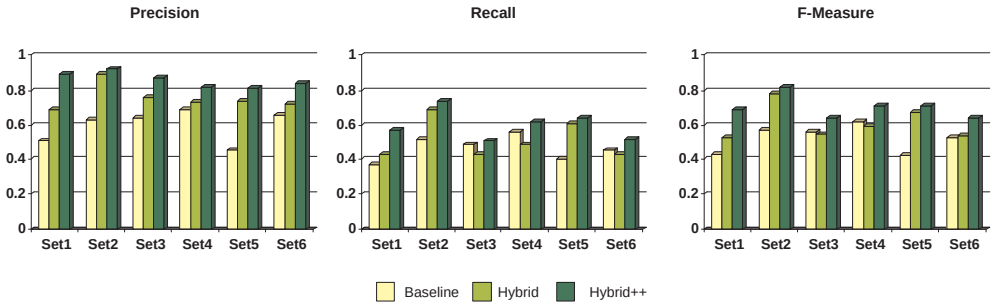


Figure 17.1: Annotation Performance

Figure 17.1 summarizes the dataset-wise results of the experiments conducted using the 3 methods. The first two graphs describe the annotation precision and recall, and the last graph denotes the F-measure of annotation. The **hybrid** approach improves the precision performance over the **baseline** approach for all the 6 datasets. On an average, the precision improved by 26%. The recall improved by at least 15% for at least three datasets and decreased by 4-12% for the other three. The second hybrid approach, **hybrid++**, further improved the performance over the first hybrid approach, on an average, by 13% in terms of the precision, and by 17% in terms of the recall.

The **hybrid++** method achieved an average precision of 0.86 and an average recall of 0.60. We investigated the reasons for a low recall. It was found that since *SnAPI* uses exact string matching as its underlying linguistic technique, the unsuccessful cases occurred because of string mismatch between the term and the concept descriptions. Hence, we added a term processing component that removes special characters (-,#, /, etc.) and performs acronym expansion using a dictionary of 103 frequently used clinical acronyms such as "T"(temperature), "BTL"(Bilateral Tubal Litigation), "VTE" (venous thromboembolism), etc. We re-conducted the experiments for the 3 methods, wherein the form terms were processed before being fed into the API module for concept retrieval. As illustrated in Figure 17.2, the average performances of the three methods improved consistently. The **hybrid++** method, with the term processing component, achieved an average precision of 0.89 and an average recall of 0.76. The average durations taken to annotate a form from

all the datasets are 1.28s, 1.77s, 2.31s, 10.29s, 8.12s, and 3.44s, respectively.

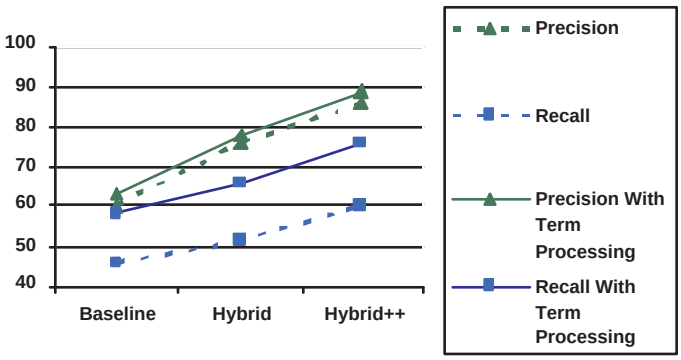


Figure 17.2: Impact of the term processing component

Finally, we could draw the following implications.

- *Impact of Structure:* The **hybrid** approach involving both structure and linguistics, led to 26% improvement in the precision over the **baseline** approach. On extracting further knowledge from the semantic structure, i.e., using the **hybrid++** method, the average precision improved by 43% and the average recall improved by 29%, over the **baseline** approach. This success is because of the increase in the number of correct concept predictions achieved as a result of incorporating the structural knowledge. This clearly indicates that the structural knowledge has the ability to address the context challenge, and improve the overall mapping performance.

- *Impact of Linguistics:* The impact of linguistics could be quantified by the change in performance upon the addition of the new term mapping component. The new linguistic component improved the precision of the three methods only slightly by 3-5% each. This is depicted by the two close lines for precision in Figure 17.2. This component, however, had a lot of impact on the recall and improved the performance by at least 25% for all the three methods. This is because the new component helped in retrieving a much larger number of terms. This indicates that linguistic-based approaches can certainly improve the recall and address the diversity challenge to a large extent.

- *Annotation Performance:* Even with the limited training data, the **hybrid++** method with term processing achieved a promising performance with an average precision of 0.89 and an av-

erage recall of 0.76. Earlier works have maintained that a high precision can only be achieved using expert analysis[63], yet our automated approach performed well. The relatively lower value of recall could be attributed to the simplicity of the linguistic functions used in this work. The recall improved consistently and significantly upon the addition of the new term processing component. This suggests that it can be further improved by including some external resources such as clinical and general thesaurus, medical acronym dictionary such as RNotes[96], and incorporating some sophisticated term processing techniques such as stemming and auto-correction[97].

17.3 Mapping Form to Database

Finally, we conducted the last set of experiments to study the performance of the mapping process simulated by the prototype. As mentioned before, we conducted three versions of these experiments by altering the correspondence discovery technique. In the following subsection, we first describe the general results, and then describe the goal accomplishments for the three versions.

17.3.1 General Results

To give a general idea of the result of the mapping experiments, we describe the scale of the generated databases, and the mapping duration.

Description of the Evolved Database

The scales of the evolved databases are shown in Figure 17.3. The x-axis shows the kind of database element, and the y-axis shows the total number of elements. The figure clearly illustrates the wide variety in the database scales; the generated databases range from having 35 tables to as many as 450 tables. Each area indicates the contribution of a form in evolving the database. The peaks denote the general pattern of forms in a given dataset. Most of the datasets peak at columns, implying the prevalence of textbox fields in the forms. The database 2 peaks at values implying the prevalence of select and radiobutton fields in the forms. The database 5 peaks at foreign keys indicating the prevalence of categories and subcategories in the forms. The broad areas represent the presence of longer forms, and the narrower regions represent the presence of shorter, or mergeable forms.

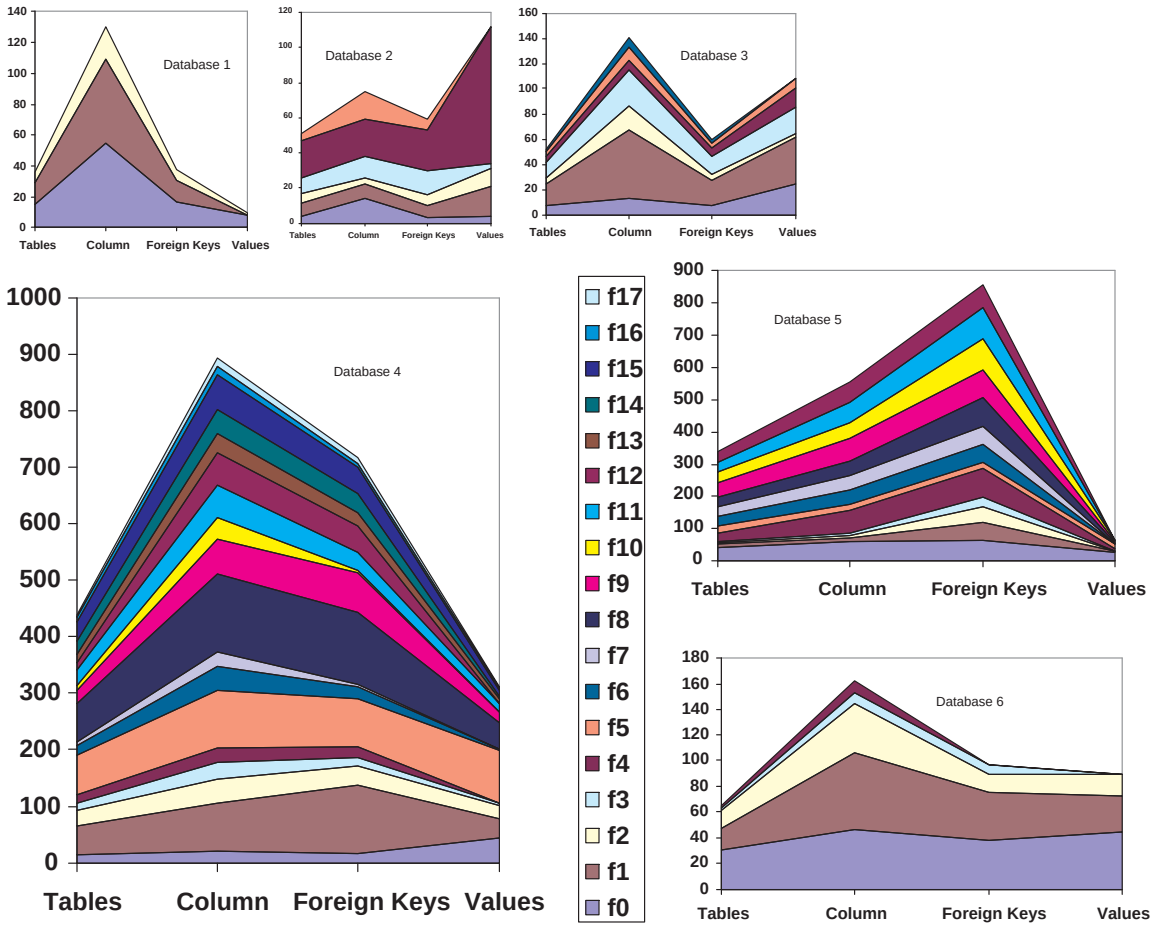


Figure 17.3: Database Scale

Mapping Duration

The duration of mapping a form to an existing database is displayed in Figure 17.4. The x-axis denotes the forms, and the y-axis represents the mapping duration in seconds. The duration does not include the form tree generation time, user intervention time, or the execution of database SQL DDL statements. The duration followed no fixed pattern. It depended on multiple factors including the size of the form, and the size of the existing database. Lucene indexing helped in controlling the duration and it ranges from a few milliseconds to 200 seconds, even for the large-scale databases such as the ones generated from the datasets 4 and 5.

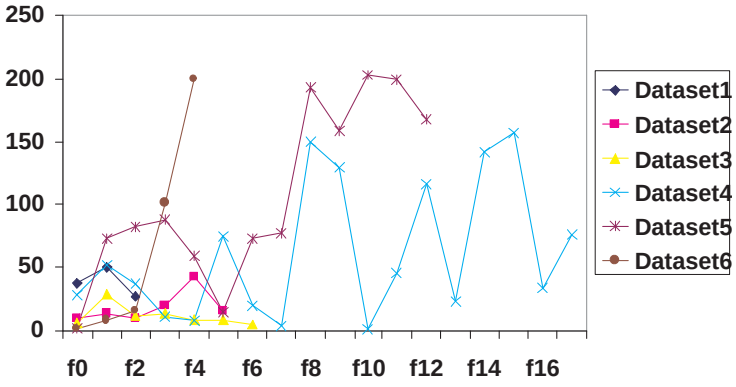


Figure 17.4: Time Taken for Mapping Forms(in seconds)

17.3.2 Measuring Principle Compliance

In this section, we report the compliance of the evolved database with the principles of high quality and optimization. Given the large scale of databases, it was nearly impossible to manually analyze each database with respect to the principles. Given the entire database is generated using the birthing algorithm, we could intuitively state the evolved databases are *correct* ($\mathcal{P}1$), *complete* ($\mathcal{P}2$), and *normalized* ($\mathcal{P}4$). In this study we focus on the *compactness* principle ($\mathcal{P}3$), and provide an approximate quantitative account on the compactness of the databases. Also, we compare the small-scaled system generated databases with the gold databases in the light of the mapping principles.

Database Compactness

To give an account on the *compactness* property, it was essential to determine the number of mergers, and the number of duplication of semantically similar elements in the database. Given the large scale of both the forms and the databases, a manual analysis of the databases was not possible. We thus created an approximate universal set of various merging instances encountered during the mapping process. This set consists of the "union" of the situations detected by the mapping discovery phases of the three versions of the experiments, i.e., linguistic-based, concept-based , and hybrid discovery. For all the datasets, about 1,875 distinct merging situations were encountered.

With this universal set, for each method, we categorize each situation into three categories, (i)

when the situation was turned into actual mergers; (ii) when the situation was turned into duplication of elements; (iii) when the situation remained undetected. The result of this categorization for the 3 versions is shown in the Figure 17.5.

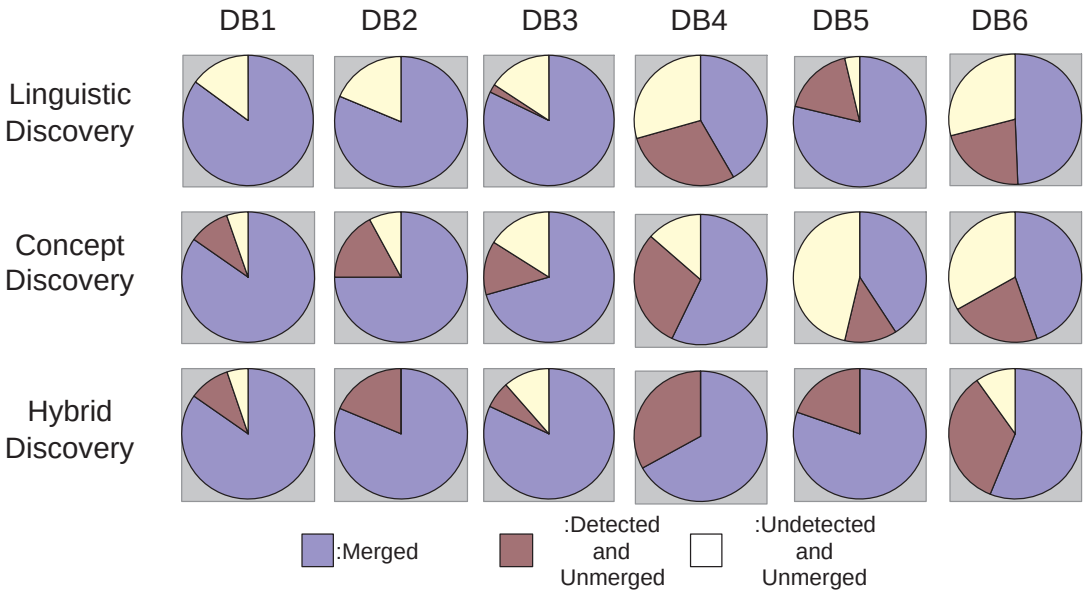


Figure 17.5: Compactness of Databases

For the linguistic-based discovery method, 4 databases had at least 75% compactness. In the remaining two databases, i.e., 4 and 6, at least 20% of the situations were not turned into actual mergers because of some peculiar form characteristics such as:

- *Format Diversity:* The formats of the columns to be merged were different, e.g., the column Date was specified as string in one database, and as a date type in another one. The column Gender appears in a textbox format in one form, and as a radiobutton group with options *Male* and *Female* in another form. Another example is the form element *DOB* that could be associated with a single textbox, or multiple textboxes corresponding to date, month, and year. Such kinds of mergers were rejected by the algorithm.

- *Section Scattering:* Different aspects of the same concept were spread out in different forms, or the same concept was listed under different sections from different forms, leading to a higher null value ratio. An example of this situation is shown in Figure 17.6. The potential null value ratio for the merger was higher than the quality tuning factor. Hence, the merger was rejected

in favor of the optimization principle $\mathcal{P}6$.

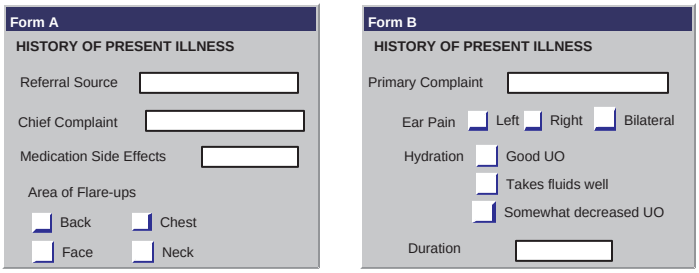


Figure 17.6: Dataset 4: Variety of Forms

For the linguistic-based discovery, the undetected situations (avg. 18%), represent the ones involving the terms that required sophisticated processing, or SNOMED's rich descriptions for identification, e.g., the term "O" and "Objective;" "HPI" and "History of Present Illness;" "BP" and "Blood Pressure."

For the concept-based discovery, 3 databases (1, 2, and 3) had at least 70% compactness. In the remaining databases, i.e., 4, 5, and 6, at least 38% of the situations were not turned into actual mergers. The low performances for the datasets 4 and 6 are because of the peculiar form characteristics as described before. The low performance of dataset 5 is primarily because of the undetected situations. The undetected situations represent the ones involving the terms that match linguistically and semantically, but do not have a corresponding concept in the SNOMED CT services. The dataset 5 encountered 46% of such situations, and hence delivered very less compactness. We also measure the extent of annotation of the databases produced by this method, i.e., the number of annotated elements by the total number of elements(tables, column, lookup values) in the database. On an average, 39% of the databases were annotated. It should be noted this value is different, and about 33% lesser than the concept mappability measures reported in the Table 14.2. This implies the redundancy created by the linguistically matching elements in the database. Since, we used the exact concept matching method to consider the potential mergers, the unmappable and yet semantically matching terms were not merged and hence duplicated several times in the generated database. It should also be noted that certain annotations were incorrect, as the **hybrid++** annotation method generates unto 89% of precision. However, this does not affect the mapping results as the incorrect

annotations are consistent throughout the datasets.

For the hybrid discovery method, 4 databases had at least 80% compactness. In databases 4 and 6, at least 30% of the situations were not turned into actual mergers because of the peculiar form characteristics as discussed before. The few (4%) undetected situations represent the ones involving the form terms that had corresponding concept matching element in the database, as well as another linguistically (and semantically) matching element in the database. The former correspondence is detected and invalidated first, and hence the latter and more correct correspondence went undetected. On an average, 43% of the databases were annotated, which is about 29% lesser than the original form concept mappability measures.

Comparison with Gold Databases

We compare the first 3 databases, evolved using the linguistic discovery version, with the gold databases. We performed a table-level comparison between the algorithm generated database and the two gold databases and looked for match/mismatch. As shown in Figure 17.7, we find that 74%(avg.) of the system generated tables "perfectly match" with one of the tables in the gold databases. A perfect match occurs between two tables when all the columns and the foreign keys perfectly match. Two columns match if they have matching names, null constraints, and data types. Two foreign keys match if they both reference the "matching" tables in the respective databases.

We manually analyzed the mismatched tables and found different variations in the schema design. In the light of the quality and optimization principles, the mismatches could be assigned to two classes:

- *Positive Mismatch:* A positive mismatch occurs when the system generated table is superior to its gold standard counterpart in terms of the desirable properties of high quality and optimization. This is depicted by two patterns (extended radiobuttons, or hierarchical segments) as depicted by patterns A and E in Figure 17.8.

- *Negative Mismatch:* A negative mismatch occurs when the system generated table is inferior to its gold standard counterpart in terms of the desirable properties of high quality and op-

timization. This occurred due to the following reasons. The mapping algorithms resulted in some extraneous columns or tables when encountering certain patterns like extended checkboxes, yes/no radiobuttons, and repeated radiobuttons, illustrated by the patterns B, C, and D, respectively in Figure 17.8.

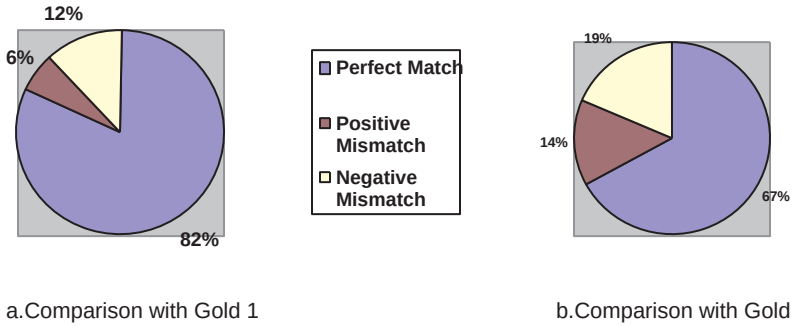


Figure 17.7: Table Comparison - System Generated Vs Gold Databases

For all the discrepancies illustrated in Figure 17.8, we also selected a "winner" method based on the principle compliance. We discuss each pattern in detail here:

1. *Pattern A- Extended Radiobutton.* The main difference between the algorithm and the first gold standard is that while the algorithm creates a common column for all extended options, the gold standard creates a separate column, e.g., *Fairmount*, for each extended option. This increases the possibilities of having null valued columns. Hence, we pick the algorithm as the intermediate winner and compare it to the second gold standard. In the second gold standard, the expert chooses to place the column corresponding to the extended option in the parent table itself. This again increases the chances of NULLs. Hence, we declare the algorithm as the final winner.

2. *Pattern B- Extended Checkboxes.* On comparing the algorithm result with the first gold standard, we pick the former as it is more optimal in terms of minimizing the number of elements. However, the second gold standard results in even more optimal result by merging the option

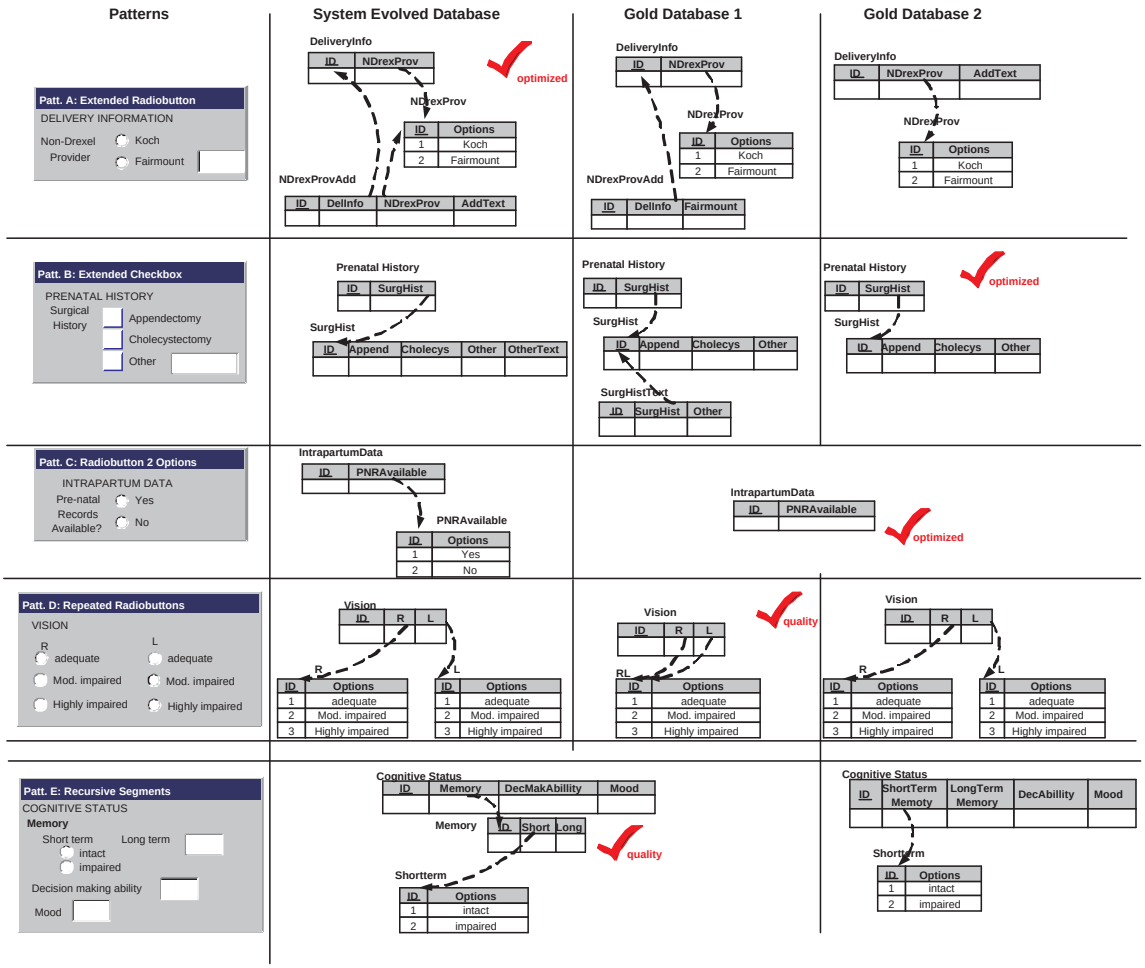


Figure 17.8: Discrepancy Scenarios - System Vs Gold Standard Databases

and the textbox into a single column without affecting the quality of the mappings.

3. *Pattern C- Radiobuttons with boolean options.* Both the gold standards result in a superior solution while translating the radiobutton options into a single yes/no column and hence leading to a more optimized result.

4. *Pattern D- Repeated Radiobuttons.* Unlike the other two methods, the first gold standard combined the similar look up tables into one, and won the case on the grounds of compactness.

5. *Pattern E- Grouping.* The second gold standard had several categories and subcategories missing from the database. In particular, the expert decided to eliminate the tables (other than the join tables) which only had foreign keys to other tables. This clearly violates the completeness

and the normalization principles. Hence, the algorithm and the first gold standard are the joint winner for this pattern.

17.3.3 Measuring User Interventions

To provide an account of the user interventions required to carry out the mapping process, we measure the following:

- Percentage reduction in screens: We conduct every experiment twice, with as well as without including the correspondence validation module. We measure the percentage reduction in the number of validation screens generated upon using the validation algorithm. This measure denotes the impact of validation algorithm in controlling the interventions.
- Average number of screens per form: This denotes the number of interventions required to map a form from a particular dataset.
- Options/screen: This denotes the number of options presented to the user in a validation screen.
- Screen relevance: This denotes the relevance of the screens presented as perceived by the user. It is calculated as the total number of screens wherein the user suggested to merge the elements over the total number of screens generated as a result of executing the validation algorithm.

Table 17.2 summarizes the user intervention results for all the experiment versions conducted across all the 6 datasets. The percentage reduction in screens for most cases is at least 50%. This denotes that the validation algorithm does help in controlling and minimizing the required user interventions. Basically, the algorithm helped in automatically validating or eliminating certain discovered correspondences in advance while leveraging the semantic structure of form as well as the connections in the database. The only exception is the dataset 3 for the concept-discovery version, wherein less validation scenarios were encountered.

The next column denotes the average number of screens generated per form. Herein, we make two key observations. First, datasets 4 and 5 required more number of user interventions. This

Table 17.2: Intervention Results (Outliers in bold or italics)

Version	Dataset	Red. Screens (%)	Avg. Screens	Options/screen	Screen Relevance (%)
Linguistic	1	50	4	2	15.39
	2	77	2	5	42.86
	3	69	2	5	**50.00**
	4	55	*10*	3	39.79
	5	76	*21*	1	**94.18**
	6	62	5	4	32.14
Concept	1	77	1	1	**75**
	2	62	3	1	**68.75**
	3	*18*	5	1	46.87
	4	54	*8*	2	**45.45**
	5	65	*15*	5	73.57
	6	65	4	*9*	42.86
Hybrid	1	52	4	2	15.38
	2	75	3	3	50
	3	57	4	2	29.63
	4	51	*13*	4	43.29
	5	69	*27*	2	86.04
	6	59	8	3	**45**

is because of the relatively larger size of these datasets, and hence more possibilities of mergers between forms and databases. Secondly, the hybrid discovery method required more number of user interventions than the other two methods. However, it also helped in identifying more merging scenarios as denoted by the Figure 17.5.

The next column denotes the average number of options (per validation screen) presented to the users. For most cases, this varied from 1 through 5 average options per screen, which is easily manageable for any user to process[98]. The dataset 6, for the concept-discovery version, was an exception wherein several SNOMED CT concepts matching a particular form element were found in the existing database. This is particularly because of the presence of the *Other* and *Comments* fields in multiple sections of a given child visit encounter forms. These terms mapped to the same SNOMED CT concepts irrespective of their container section, e.g., both the sections *Pertinent ROS* and *Teaching* contain the field *Other* but were mapped to different tables since the concepts belong to different places in the database.

The last column denotes the screen relevance as perceived by the user. This followed no fixed pattern. We hence spotted the winning method for each dataset, as marked in the bold font. In

general, the screens presented by the hybrid discovery method are found to be less relevant than the other two methods. This is because the hybrid method combines the shortcomings of both the methods and returns the irrelevant screens generated by both the methods. It is interesting to note that the screen relevance was particularly higher (94%) for the dataset 5 (linguistic-discovery version) that represents the family practice forms. In these forms, the linguistically matching and yet semantically differing terms were not very prevalent. On the other hand, the dataset 1 had many such terms that resemble linguistically but differ semantically. Hence, the relevance of screens for this dataset is very low (15%) for both the linguistic-discovery and the hybrid discovery methods. Also, an outlier dataset is the dataset 6 wherein the screen relevance of the hybrid method is more than each of the constituent methods. This is because of the relatively higher overlap between the correspondences discovered by the constituent methods. This in turn helped in improving the overall screen relevance generated by the hybrid method.

17.4 Experiment Summary and Implications

We conducted 3 experiments to test the effectiveness of the framework in evolving a principle compliant database, and in minimizing the user interventions. We now present the implications of these experiments.

17.4.1 Form Semantics Experiments

We conducted the first experiment to test the form tree extraction module and used the module to derive semantic trees from 52 data-entry forms. The resultant form trees were approximately 98% accurate. In the larger picture, these form trees contribute to the correctness, completeness, and normalization of the derived databases. An average form tree with 135 edges required only 0.08 seconds to be generated. This suggests the real-world applicability of this module. A limitation is that it requires supervised learning to train the Hidden Markov models. In the future, we intend to explore certain unsupervised methods to train the learning models. Another limitation is that it requires certain human intervention to derive the cardinalities among the form elements. On an average, 10 interventions per form were needed to disambiguate among 1:1,1:M, or M:M cardinalities. One

solution is to bypass the user intervention layer and assume M:M cardinality for each relationship. This, however would produce less optimized database due to the increased chances of NULL values in the database. Another solution is to maintain a repository of the expected cardinalities of some frequently found pair of clinical entities, e.g., *Patient* and *Primary Care Physician* would always have an M:M relationship, whereas *Patient* and *History* would have a 1:M relationship.

We conducted the second set of experiments to test the hybrid approach for annotating forms that leverages semantic structure as well as linguistic properties of the form elements. When tested on around 2500 form terms belonging to 52 form trees, the proposed **hybrid++** method with term processing led to a precision of 0.89 and a recall of 0.76. The duration of annotation ranged from 1 to 11 seconds per form. The results imply that upon leveraging the semantic structure of the form tree and the linguistic properties of the terms, the precision improved over by 43% and the recall improved by at least 29%. The performance could be further improved using sophisticated term processing techniques and leveraging other relationships present in the SNOMED CT services. A limitation of this approach is that it is based the Naive Bayes classification algorithm that requires manual tagging for training. In the future, we intend to explore whether an unsupervised classification algorithm could be used to produce a competent term mapping performance.

17.4.2 Mapping Experiments

The mapping experiments were performed to evaluate the overall effectiveness of the framework. The experiments were conducted separately on the 6 datasets using 3 mapping versions, i.e., linguistic-based, concept, and hybrid discovery. This resulted into 18 cases of evolving databases. The generated databases were of varying scales randing from 35 through 450 tables.

Result Summary

The evolved databases were at least 70% compact in 11 out of 18 cases. The linguistic-based and concept-based methods generated databases with average 69% and 62% compactness, respectively. The hybrid approach produced 74% compact databases. Also, while the first two methods detect 79% and 81% of the merging scenarios, respectively, the hybrid approach detects up to 96% of the

merging scenarios. On comparing the small-scale databases with the gold databases, we find that 84.5% of the tables generated by the system are similar or superior to the gold standard databases in terms of principle compliance.

In 17 out of 18 cases, the validation algorithm helped in reducing at least 50% of the user interventions. Overall, the validation algorithm led to about 61% reduction in the number of screens. On an average, 10, 8, and 13 screens per form were generated for user approval using the linguistic, concept, and hybrid discovery methods, respectively. Most of the screens had 1 to 5 options for user to choose from. The user found only 50% of the screens to be relevant.

Implications

The results highlight various abilities of the individual components of the framework. The close resemblance of the evolved databases with the gold databases underlines the abilities of the birthing algorithm and the embedded patterns. It suggests the compliance of the birthing algorithm with the correctness, the completeness, and the normalization principles. While the experts required several hours of careful analysis to prepare the gold standards, the birthing algorithm executes within few seconds. The percentage reduction in the number of screens clearly demonstrates the ability of validation algorithm in minimizing the number of user interventions.

The results also depict the synergy among various components of the framework. The compactness of the generated databases is very promising. This indicates the effectiveness of the framework in leveraging the semantic structure of forms and term annotations, and thereby in merging the semantically matching elements. The close resemblance of the evolved databases with the gold databases confirms the accuracy of the semantics captured by the form trees generated using the HMM-based extraction method.

The analysis of the results also helped in identifying the quantitative influence of one component on another. Figure 17.5 makes it very apparent that the hybrid approach outshines the linguistic and concept match approaches in terms of principle compliance. We find that the hybrid approach improves the other two approaches, by an average 19% in terms of identifying the merging scenarios, and by an average 13% in terms of ensuring the compactness of the evolved databases. This suggests

that annotation helps in improving the quality of the evolved databases. However, the hybrid method is less effective in terms of the screen relevance, and the number of screens generated than both the constituent methods. Our experiments and analysis could not decode certain correlations and influences, such as the impact of annotation on the performance of validation algorithm.

Lessons Learned

The experiments also helped in providing many guidelines in improving the performance of the framework. The main outliers in terms of the compactness were datasets 4 and 6. These forms had some peculiar properties that led to a higher for the null value ratio while merging. To further improve the compactness, it is required to customize the approach based on the nature of the forms. The gold databases highlighted three major limitations of the birthing algorithm in the form of patterns *B*, *C*, and *D* in Figure 17.8. The pattern *C* of repeated radiobuttons contributes toward redundancy in the databases. The other two patterns affect the optimization of the databases by increasing the chances of NULL values and creating extraneous database elements. In the future, we intend to work on these patterns, particularly to find a more optimal way to represent extended radiobuttons and checkboxes, and to compile a dictionary of variations of radiobutton options with "yes/no" values (such as *Heart Rate:* "regular/irregular", etc). To improve the performance in terms of the user interventions, it is required to further enhance the validation algorithm by identifying more validation scenarios. In the future, it is needed to investigate the irrelevant screens and options generated and study whether more patterns for elimination could be derived from them. One challenge is to identify the correspondences that match semantically and yet differ in terms of their placement in the database.

Part V

Final Remarks

Chapter 18: Contributions and Conclusions

Jagadish et al.[11] have illustrated the five painful issues in database usability. Among them, the "birthing pain" is related to the difficulties of creating a database and putting information into a database. We are motivated to study an easy and flexible way for users to use a database for storing information. Forms are a user-friendly way for interacting with databases. In this thesis, we develop a framework for automatically mapping data-entry forms into existing relational databases. In a way, this framework can be viewed as an inverse process for automatically generating query forms from databases[99]. With a thorough empirical analysis in the healthcare domain, we show that with the availability of such a highly automated framework, users do not need a clear knowledge of the final structure of a database. As users create and map more forms for evolving needs, the structure of the database grows automatically, however, in a principled way, with predictive characteristics.

The overall mapping approach can be summarized in the following manner. The goal of the system is to map a given user-designed form into an existing relational database while maintaining the quality and optimization of the resultant database, and while ensuring minimal user intervention. The input to the process is an HTML form, imported into the system, or designed using the DIY interface of the system. An HMM-based extraction component represents the form into an equivalent tree structure. In addition, a classification based annotation component tags the form terms with respect to standard concepts. The discovery component discovers the correspondences between the form tree and the existing database. The validation component validates and eliminates certain discovered correspondences, and presents the remaining for user intervention. The birthing component then translates the form tree into a new database in the light of the database design principles. The validated correspondences are transfered to this new database. The merging component studies the fitness of the discovered correspondences and integrates the two databases while ensuring compactness. This framework makes the following research contributions.

- **Understanding Forms:** We have introduced the 2-layered HMM approach for automatic derivation of a semantic tree from a given form template. This approach is motivated by the probabilistic nature of the form design process. We encoded the implicit knowledge required for form understanding into an HMM-based artificial designer. This is the first work to employ HMMs for extracting information from user-designed forms. When applied on 52 clinical forms, the approach leads to close to 98% accuracy, and derives an average tree with 135 parent-child relationships in 0.08 seconds.

- **Term Annotation:** We have introduced and addressed a new problem of mapping a form term to a SNOMED CT concept. While the existing linguistic-based methods are solely based on term-level matching, the proposed method performs a context-level matching followed by a term-level matching. Herein, the context of a given term is systematically extracted from the semantic structure of the form, and the context of a SNOMED CT concept is assumed to be its predefined semantic category. The proposed approach first uses a structure-based model to determine the semantic category for a given term, and then maps the term to the most linguistically matching clinical concept. We have conducted an empirical study on 52 clinical forms. Compared to an existing linguistic based approach, the proposed method achieves a performance improvement of 43% in terms of precision, and 29% in terms of recall. The method helps achieve an average precision of 0.89, and an average recall of 0.76. In addition, the approach requires at least 1, and at most 11 seconds to annotate a given form.

- **Correspondence Validation Algorithm:** Based on certain frequent correspondence validation scenarios, we have developed a validation algorithm to automatically validate or eliminate certain discovered correspondences. This algorithm helps reduce the average number of user intervention screens by 61%.

- **Birthing and Merging Algorithm:** We have proposed two algorithms for database design and evolution. The birthing algorithm automatically derives a new database corresponding to a given form based on the database design principles. The merging algorithm integrates two

given databases based on the specified discovered correspondences and the desired trade-off between compactness and reduction of NULL values. The 6 experimental dataset containing 52 forms result into 4 medium-scale (up to 65 tables) and 2 large-scale (up to 500 tables) databases. The evolved medium-scale databases intersect with the expert-designed databases by 84.5%. We have experimented using 3 different versions (linguistic-based discovery, concept-based discovery, and hybrid discovery) of the framework, leading to 18 mapping experiments in all. The algorithms lead to at least 70% compact databases, in 11 out of the 18 cases.

In sum, we learn the following lessons from the experiment results.

- We have studied the individual impact of the structure and the linguistics on the annotation performance. We find that while the term linguistics can only influence the recall performance, the semantic structure has the potential to improve the overall mapping performance, i.e., recall as well as precision. In the future, it is desirable to develop hybrid approaches that can address various annotation challenges, and lead to a superior performance.

- The use of annotation positively impacts the quality of the database. The hybrid discovery technique, which leverages both the annotation and the linguistic properties of a term, leads to 19% improvement in identification of merging situations, and 13% improvement in the compactness of the databases.

- The birthing algorithm could be further refined in terms of handling radio-button groups and extended check-boxes, while ensuring further compactness and optimization.

- The hybrid discovery approach leads to more user interventions, and lesser screen relevance, than the linguistic and concept-based discovery methods. The number of user interventions and the screen relevance could be further improved by enhancing the validation algorithm to include more validation patterns.

- Given the experiments conducted with a functional prototype, and the promising results, we conclude that it is technically feasible to implement such a mapping framework in a real-world setting. This system can be used in any small to large-scale application that relies on forms

for data collection. While the dissertation focused on the healthcare domain, this system can also improve the usability of other applications such as vehicle registration systems, student registration systems, and on line selling systems, e.g., craigslist [100] and ebay [101].

Chapter 19: Limitations

This study had certain limitations which we classify in terms of the techniques, the technique evaluation methods, the experimental design, and the entire study.

19.1 Techniques

The proposed techniques can be expanded in several ways. The use of HMMs for form tree extraction poses certain challenges. Currently, the training data for the HMMs is prepared by manual tagging. Our experience of tagging 52 data-entry forms suggests that the training samples can be constructed quickly and easily, as compared to the construction of exhaustive set of rules or heuristics. However, to minimize human intervention, we intend to explore the use of unsupervised training methods such as Baum Welch algorithm. Another limitation of the form understanding approach is that it does not detect weak entities, cardinalities, and participation constraints of the semantic associations. One possible solution is to maintain a repository of frequently found weak entities and constraints between clinical entities[102]. Addressing these issues would further improve the correctness and optimization of the resultant database.

The classification model used for term annotation also had certain limitations. It cannot handle the missing and inapplicable values in the training data. Also, it can be improved by leveraging other defining relationships and the compositional nature of the SNOMED CT to derive post coordinated mapping expressions, and to further improve the annotation performance[103].

Both the mapping discovery and the merging algorithms could be refined to incorporate concatenated matches, e.g., the form element *name* collectively corresponds to the database columns firstname and lastname. We also intend to improve the birthing algorithm by incorporating more complicated form features such as the conditions embedded in forms as javascript code, and the table widgets. In the future, it is also important for the merging algorithm to detect and eliminate circular references.

19.2 Technique Evaluation

Another limitation of the proposed techniques is that they are not well evaluated. We intend to compare the employed learning models with other suitable models such as support vector machines and conditional random fields, Bayesian networks, and classification association rules. We also intend to study the validity of the assumptions adopted by the term annotation method, e.g., that class conditional independence holds true, and that the most linguistically matching concept returned by the category-specific mapping is the desired one.

The validation, birthing, and merging algorithms mainly rely on heuristics. The completeness and correctness of these heuristics is yet to be validated. We need a mechanism to theoretically verify the tree design rules, the heuristics used for validation and merging, the birthing form patterns, and the classification attributes.

19.3 Experimental Design

In terms of the experiments, several aspects are yet to be tested. Experiments involving both the automatic form tree extraction method and the term annotation method are yet to be performed. To test the performance of the mapping framework in a heterogeneous environment, it is important to map and merge forms belonging to different datasets.

19.4 Study

Since the entire study focused on the technical aspects of the framework and aimed toward evolving a principled database, the main limitation is the lack of thorough user studies. It remains an open question whether the users can understand and select the right correspondences. Also, the clinical annotations were not prepared by domain experts, and hence may not be 100% accurate. A separate user study could be conducted with domain experts to understand their process of form annotation, and measure the efforts involved. Another limitation of this study was the limited time available for implementation and experimentation.

A limitation of the study is the lack of availability of clinical forms. Unlike the search forms which are widely and freely available[50], it was a challenge to collect the real-world data-entry forms.

We collected 52 data-entry forms. Although the size of the dataset is limited, an average form had about 135 form elements providing a decent scale for testing the proposed framework. In the future, we intend to prepare a much bigger benchmark repository of such clinical forms.

The large scale of databases posed some challenges in terms of evaluation of the results and the preparation of gold standards. Hence, the compliance of the database with the principles could only be projected. The comparison with the small-scale gold databases resulted into some positive and negative mismatches. This suggests that the gold databases do not represent the ideal cases, but are only the representatives of the expert's approach to real-world database design. This in turn suggests that it is possible to create an ideal gold, which when compared to the system generated databases, would not lead to any positive mismatches. In the future, we intend to manually create a repository of the ideal gold databases for some frequently used form datasets.

Chapter 20: Future Research Directions

Several new research directions spawn from the study conducted in this thesis. We categorize them into two main categories; health informatics and computer science.

20.1 Health Informatics

Given the promising performance of the framework in the healthcare domain, one direction is to investigate whether this framework can be evolved into a flexible Electronic Health Record(fEHR) system. This direction is motivated by the vision of the US government to effectively induce health information technologies (HITs) into healthcare by 2015[71], and by the challenges faced by the clinicians while working with the rigidly designed HITs. Using the fEHR, the clinicians can easily and quickly extend an existing EHR system as per their needs. The fEHR system would take as input the clinician-designed form corresponding to a given set of user requirements, and would induce the form into an existing database. To investigate the willingness of clinicians to work with such systems, we conducted a user study with some clinicians working in a nurse-managed health services center. The goal was to investigate whether they can design forms using an interface. The clinicians could perform the given tasks of modeling and building forms with 100% accuracy in all but one case. They could use the system for designing the databases based on short and simple as well as long and advanced needs within a span of few minutes in most cases. Also, there were signs of improvement in clinicians' levels of efficiency, confidence, and understanding in using the system. This suggests that the system has the potential to reduce the current problems of HITs, particularly, the inefficiency faced by clinicians, and the inconsistency between clinician's needs and databases. In addition to this, the system is adoptive in that it helps the clinicians to learn and improve their need modeling and form building skills. The user study helped in identifying several future directions for improving the system. Considering the main challenges faced by the participants, we intend to re-design fEHR's interface such that it helps clinicians in taking modeling decisions and suggests design

alternatives to them. The case study participants suggested addition of new features like calculated fields, table widgets, etc. While addition of such advanced features is technically possible, what is challenging is to introduce them without imposing any learning burden on the clinicians.

In the future, we intend to expand this fEHR user study to see whether users can comprehend and identify correspondences between forms and databases. We also intend to study whether this framework helps in improving data quality and patient diagnosis while managing other unforeseen implications[104]. Another direction is to see how this framework could be integrated with the proprietary health information systems such as Allscripts[105], and how well does this fit in with regard to HIPPA regulations[106]. Furthermore, the mapping algorithms could be customized for specific form categories, such as encounter form, admission form, data-entry form, etc. We also intend to explore the use of other UMLS terminologies[107] for performing form term annotation.

20.2 Computer Science

In terms of general computer science research, this framework can be enhanced in many ways. One direction is the maintenance of the transformation correspondences, also known as the *mapping maintenance* problem. The merging algorithm leads to the modification of the existing database. For instance certain tables are split, and certain columns are shifted from one table to the other, and so on. Mapping maintenance implies that any change in the database should be reflected in the associated mappings of all the related forms. In addition, this change should also be propagated to views, queries, and other related applications[108–110].

Another direction is to improve the process of data collection through the mapped forms. The current version of the framework does not support automatic pulling of certain form fields based on user input. For example, when the user fills out patient's basic information, such as first name and last name, other fields like DOB, and MRN should automatically get filled out in the form. Also, record conflict resolution is yet another area of research. For instance, multiple patient's can share the same basic information (such as name). Automatic disambiguation of records while data collection is still an open question.

We find that forms are still quite under-explored. The validation and merging algorithms could

be based on several other form related information, such as its frequency of use, or the domain expertise of the designer, or the target user(e.g., physician, nurse, patient, data-entry staff, etc). Also the decisions regarding correspondence validation could be based or learned on the existing transformation mappings with previously mapped forms. Finally, we also intend to explore whether this framework could be turned into an application programming interface, or could be used for storing data in the "cloud," in the lines of the Amazon SimpleDB[111], and the Google Datastore[112].

Bibliography

[1] Pete Sawyer. Database systems: Challenges and opportunities for graphical hci. *Interacting With Computers*, 7:273–303, 1995.

[2] Fang Liu, Clement Yu, Weiyi Meng, and Abdur Chowdhury. Effective keyword search in relational databases. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 563–574, New York, NY, USA, 2006. ACM. ISBN 1-59593-434-0. doi: http://doi.acm.org/10.1145/1142473.1142536. URL `http://portal.acm.org/ft_gateway.cfm?id=1142536&type=pdf&coll=Portal&dl=GUIDE&CFID=47678283&CFTOKEN=22667157`.

[3] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, pages 850–861. VLDB Endowment, 2003. ISBN 0-12-722442-4. URL `http://portal.acm.org/ft_gateway.cfm?id=1315524&type=pdf&coll=Portal&dl=GUIDE&CFID=47678283&CFTOKEN=22667157`.

[4] Vagelis Hristidis and Yannis Papakonstantinou. Discover: keyword search in relational databases. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 670–681. VLDB Endowment, 2002. URL `http://portal.acm.org/ft_gateway.cfm?id=1287427&type=pdf&coll=Portal&dl=GUIDE&CFID=47678283&CFTOKEN=22667157`.

[5] Tiziana Catarci. What happened when database researchers met usability. *Inf. Syst.*, 25(3): 177–212, 2000. ISSN 0306-4379. doi: http://dx.doi.org/10.1016/S0306-4379(00)00015-6.

[6] Magesh Jayapandian and H. V. Jagadish. Automating the design and construction of query forms. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 125, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2570-9. doi: http://dx.doi.org/10.1109/ICDE.2006.29.

[7] Kenneth J. Mitchell, Jessie B. Kennedy, and Peter J. Barclay. A framework for user-interfaces to databases. In *AVI '96: Proceedings of the workshop on Advanced visual interfaces*, pages 81–90, New York, NY, USA, 1996. ACM. ISBN 0-89791-834-7. doi: http://doi.acm.org/10.1145/948449.948462.

[8] Tiziana Catarci and Giuseppe Santucci. Query by diagram: a graphical environment for querying databases. In *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, page 515, New York, NY, USA, 1994. ACM. ISBN 0-89791-639-5. doi: http://doi.acm.org/10.1145/191839.191976. URL `http://portal.acm.org/ft_gateway.cfm?id=191976&type=pdf&coll=Portal&dl=GUIDE&CFID=59442550&CFTOKEN=84545770`.

[9] Cong Yu and H. V. Jagadish. Querying complex structured databases. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 1010–1021. VLDB Endowment, 2007. ISBN 978-1-59593-649-3.

[10] Yunyao Li, Cong Yu, and H. V. Jagadish. Schema-free xquery. In *In VLDB*, pages 72–83, 2004.

[11] H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. Making database systems usable. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 13–24, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-686-8. doi: http://doi.acm.org/10.

1145/1247480.1247483. URL http://portal.acm.org/ft_gateway.cfm?id=1247483&type=mov&coll=Portal&dl=GUIDE&CFID=59442550&CFTOKEN=84545770.

[12] Keith Kowalczykowski, Kian Win Ong, Kevin Keliang Zhao, Alin Deutsch, Yannis Papakonstantinou, and Michalis Petropoulos. Do-it-yourself custom forms-driven workflow applications. In *4th Conference on Innovative Data Systems Research (CIDR 2009)*, 2009.

[13] Fan Yang, Nitin Gupta, Chavdar Botev, Elizabeth F Churchill, George Levchenko, and Jayavel Shanmugasundaram. Wysiwyg development of data driven web applications. *Proc. VLDB Endow.*, 1(1):163–175, 2008. ISSN 2150-8097. doi: http://doi.acm.org/10.1145/1453856.1453879. URL http://portal.acm.org/ft_gateway.cfm?id=1453879&type=pdf&coll=Portal&dl=GUIDE&CFID=60726750&CFTOKEN=33144483.

[14] J. Choobineh, M. V. Mannino, J. F. Nunamaker, Jr., and B. R. Konsynski. An expert database design system based on analysis of forms. *IEEE Trans. Softw. Eng.*, 14(2):242–253, 1988. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/32.4641.

[15] David W. Embley. Nfql: the natural forms query language. *ACM Trans. Database Syst.*, 14(2):168–211, 1989. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/63500.64125. URL http://portal.acm.org/ft_gateway.cfm?id=64125&type=pdf&coll=Portal&dl=GUIDE&CFID=60726750&CFTOKEN=33144483.

[16] Michael V. Mannino and Joobin Choobineh. Research on form driven database design and global view design. *IEEE Database Eng. Bull.*, 7(4):58–63, 1984.

[17] Online web forms, surveys & questionnaires - formassembly.com. http://www3.formassembly.com/.

[18] *Zoho Creator*. http://creator.zoho.com, .

[19] Jotform - easiest form builder. http://www.jotform.com/.

[20] Wufoo: Online html form builder - contact, survey and payment forms. http://wufoo.com/.

[21] Ayse P. Gurses, Yan Xiao, and Peter Hu. User-designed information tools to support communication and care coordination in a trauma hospital. *J. of Biomedical Informatics*, 42(4):667–677, 2009. ISSN 1532-0464. doi: http://dx.doi.org/10.1016/j.jbi.2009.03.007.

[22] Yuan An, Prudence W. Dalrymple, Michelle Rogers, Patricia Gerrity, Jennifer Horkoff, and Eric Yu. Collaborative social modeling for designing a patient wellness tracking system in a nurse-managed health care center. In *DESRIST '09: Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, pages 1–14, Philadelphia, PA, USA, 2009. ACM. ISBN 978-1-60558-408-9. doi: http://doi.acm.org/10.1145/1555619.1555622. URL http://portal.acm.org/ft_gateway.cfm?id=1555622&type=pdf&coll=GUIDE&dl=GUIDE&CFID=59711119&CFTOKEN=71959739.

[23] Joan S. Ash, Marc Berg, and Enrico Coiera. Some unintended consequences of information technology in health care: The nature of patient care information system-related errors. *JAMIA*, 11(2):110–112, 2004. doi: 10.1197/jamia.M1471.

[24] T. Lee. Nurses' experiences using a nursing information system: early stage of technology implementation. *Comput Inform Nurs*, 25(5), 2007.

[25] A. Halevy. Why Your Data Won't Mix. *ACM Queue*, 3(8):50–58, 2005.

[26] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 49–58, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-804-4.

[27] J. Euzenat P. Shvaiko. A Survey of Schema-based Matching Approaches. *Journal on Data Semantics*, 2005.

[28] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB JOURNAL*, 10:2001, 2001.

[29] Y. An, A. Borgida, and J. Mylopoulos. Inferring Complex Semantic Mappings between Relational Tables and Ontologies from Simple Correspondences. In *Proceedings of International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE)*, pages 1152–1169, 2005.

[30] Y. An, J. Mylopoulos, and A. Borgida. Building Semantic Mappings from Databases to Ontologies. In *Proceedings of American Association for Artificial Intelligence (AAAI)*, 2006.

[31] Partha Pratim Talukdar, Zachary G. Ives, and Fernando Pereira. Automatically incorporating new sources in keyword search-based data integration. In *SIGMOD '10: Proceedings of the 2010 international conference on Management of data*, pages 387–398, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0032-2. doi: http://doi.acm.org/10.1145/1807167.1807211.

[32] Xia Yang, Mong Li Lee, Tok Wang Ling, Lee Tok, and Wang Ling. Resolving structural conflicts in the integration of xml schemas: A semantic approach. In *In Proc. ER03*, pages 520–533, 2003.

[33] Xin Luna Dong and Felix Naumann. Data fusionresolving data conflicts for integration. pvldb, 2009.

[34] Richard Y. Wang and Diane M. Strong. Beyond accuracy: what data quality means to data consumers. *J. Manage. Inf. Syst.*, 12:5–33, March 1996. ISSN 0742-1222. URL `http://portal.acm.org/citation.cfm?id=1189570.1189572`.

[35] R. Ramakrishnan and M. Gehrke. *Database Management Systems (3rd ed.)*. McGraw Hill, 2002.

[36] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database Systems Concepts*. McGraw-Hill Higher Education, 2001. ISBN 0072283637.

[37] Carlo Batini and Monica Scannapieco. *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 3540331727.

[38] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems, 3rd Ed.* Addison-Wesley, 2000.

[39] S. M. Benslimane, M. Malki, M. K. Rahmouni, and D. Benslimane. Extracting personalised ontology from data-intensive web application: An html forms-based reverse engineering approach. *Informatica*, 18(4):511–534, 2007.

[40] Ritu Khare, Yuan An, and Il-Yeol Song. Understanding search interfaces: A survey. *SIGMOD Record*, 39(1):33–40, 2010.

[41] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 129–138, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-804-4.

[42] Hai He, Weiyi Meng, Yiyao Lu, Clement Yu, and Zonghuan Wu. Towards deeper understanding of the search interfaces of the deep web. *World Wide Web*, 10(2):133–155, 2007. ISSN 1386-145X. doi: http://dx.doi.org/10.1007/s11280-006-0010-9.

[43] Zhen Zhang, Bin He, and Kevin Chen-Chuan Chang. Understanding web query interfaces: best-effort parsing with hidden syntax. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 107–118, New York, NY, USA, 2004. ACM. ISBN 1-58113-859-8. doi: http://doi.acm.org/10.1145/1007568.1007583. URL `http://portal.acm.org/ft_gateway.cfm?id=1007583&type=pdf&coll=GUIDE&dl=GUIDE&CFID=62954640&CFTOKEN=46271710`.

[44] Ritu Khare and Yuan An. An empirical study on using hidden markov model for search interface segmentation. In *Proceedings of 18th ACM Conference on Information and Knowledge Management (CIKM)*, 2009.

[45] Oliver Kaljuvee, Orkut Buyukkokten, Hector Garcia-Molina, and Andreas Paepcke. Efficient web form entry on pdas. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 663–672, New York, NY, USA, 2001. ACM. ISBN 1-58113-348-0. doi: http://doi.acm.org/10.1145/371920.372180.

[46] Hoa Nguyen, Thanh Nguyen, and Juliana Freire. Learning to extract form labels. *Proc. VLDB Endow.*, 1(1):684–694, 2008. ISSN 2150-8097. doi: http://doi.acm.org/10.1145/1453856.1453931. URL `http://portal.acm.org/ft_gateway.cfm?id=1453931&type=pdf&coll=GUIDE&dl=GUIDE&CFID=62954640&CFTOKEN=46271710`.

[47] Denis Shestakov, Sourav S. Bhowmick, and Ee-Peng Lim. Deque: querying the deep web. *Data Knowl. Eng.*, 52(3):273–311, 2005.

[48] Eduard Constantin Dragut, Thomas Kabisch, Clement T. Yu, and Ulf Leser. A hierarchical approach to model web query interfaces for web source integration. *PVLDB*, 2(1):325–336, 2009. URL `http://dblp.uni-trier.de/db/journals/pvldb/pvldb2.html#KabischDYL09`.

[49] Wensheng Wu, AnHai Doan, Clement Yu, and Weiyi Meng. Modeling and extracting deep-web query interfaces. *Advances in Information and Intelligent Sys.*, pages 65–90, 2009.

[50] Ritu Khare and Yuan An. *A Library of Form Interfaces*. http://cluster.ischool.drexel.edu:8080/ibiosearch/ datasets.html, 2010.

[51] Jelena Pavicevic, Ivan Lukovic, Pavle Mogin, and Miro Govedarica. Information system design and prototyping using form types. In *ICSOFT (2)*, pages 157–160, 2006.

[52] Ivan Luković, Pavle Mogin, Jelena Pavićević, and Sonja Ristić. An approach to developing complex database schemas using form types. *Softw. Pract. Exper.*, 37(15):1621–1656, 2007. ISSN 0038-0644. doi: http://dx.doi.org/10.1002/spe.v37:15.

[53] Joobin Choobineh and Santosh S. Venkatraman. A methodology and tool for derivation of functional dependencies from business forms. *Inf. Syst.*, 17(3):269–282, 1992.

[54] Zoho creator. https://creator.zoho.com/, .

[55] The model-driven tool for microsoft visual studio 2008. http://www.deklarit.com/portal/hgxpp001.aspx?12.

[56] Microsoft office infopath. http://office.microsoft.com/en-us/infopath/default.aspx.

[57] Paul A. Harris, Robert Taylor, Robert Thielke, Jonathon Payne, Nathaniel Gonzalez, and Jose G. Conde. Research electronic data capture (redcap)–a metadata-driven methodology and workflow process for providing translational research informatics support. *Journal of Biomedical Informatics*, 42(2):377 – 381, 2009. ISSN 1532-0464. doi: DOI:10.1016/j.jbi.2008.08.010. URL `http://www.sciencedirect.com/science/article/pii/S1532046408001226`.

[58] Perfect forms, a smarter way to manage and monitor your business. http://www.perfectforms.com/.

[59] Appnowgo- the easy web application builder. build online database applications with ease. http://www.appnowgo.com/index.php.

[60] Edit form - google docs. http://spreadsheets.google.com, .

[61] Openmrs: Open source health it for the planet. http://openmrs.org/.

[62] S B Henry, K E Campbell, and W L Holzemer. Representation of nursing terms for the description of patient problems using snomed iii. *Proceedings of the Annual Symposium on Computer Application in Medical Care*, pages 700–704, 1993.

[63] Randolph C. Barrows Jr., James J. Cimino, and Paul D. Clayton. Mapping clinically useful terminology to a controlled medical vocabulary. In *Proceedings of Annual Symposium of Computing Applications in Medical Care*, pages 211–215, 1994.

[64] Jon Patrick, Yefeng Wang, and Peter Budd. An automated system for conversion of clinical notes into snomed clinical terminology. In *In Proc. of HKMD-07, volume 68 of CRPIT*, pages 219–226, 2007.

[65] Alan Rector Rahil Qamar and. Most: A system to semantically map clinical model data to snomed-ct. In *In the proceedings of Semantic Mining Conference on SNOMED-CT*, pages 38–43, 2006.

[66] Lawrence W. Wright, Holly K. Grossetta Nardini, Alan R. Aronson, and Thomas C. Rindflesch. Hierarchical concept indexing of full-text documents. *J. Am. Soc. Inf. Sci*, 50:514–523, 1999.

[67] Ying Li, Sharon Lipsky Gorman, and Noémie Elhadad. Section classification in clinical notes using supervised hidden markov model. In *Proceedings of the 1st ACM International Health Informatics Symposium*, IHI '10, pages 744–750, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0030-8. doi: http://doi.acm.org/10.1145/1882992.1883105. URL `http://doi.acm.org/10.1145/1882992.1883105`.

[68] Suhit Gupta, Gail E. Kaiser, Peter Grimm, Michael F. Chiang, and Justin Starren. Automating content extraction of html documents. *WORLD WIDE WEB - INTERNET AND INFORMATION SYSTEMS*, pages 179–224, 2005.

[69] Ritu Khare, Yuan An, Il-Yeol Song, and Xiaohua Hu. Can clinicians create high-quality databases? a study on a flexible electronic health record (fehr) system. In *Proceedings of 1st ACM International Health Informatics Symposium (IHI)*, 2010.

[70] Yuan An, Ritu Khare, Il-Yeol Song, and Xiaohua Hu. Automatically mapping and integrating multiple data entry forms into a database. In *In the proceedings of 30th International Conference on Conceptual Modeling*, 2011.

[71] Subhashish Karmakar. How to design a usable and meaningful emr application. Technical report, eMids Technologies, 2010.

[72] Nicholas Kushmerick. Finite-state approaches to web information extraction. In *Extraction in the Web Era*, volume 2700 of *Lecture Notes in Computer Science*, pages 77–91. Springer Berlin / Heidelberg, 2003. URL `http://dx.doi.org/10.1007/978-3-540-45092-4_4`. 10.1007/978-3-540-45092-4_4.

[73] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, New York, 2001.

[74] M. A. Hernández, L. Popa, H. Ho, and F. Naumann. Clio: A schema mapping tool for information integration. In *Proceedings of ISPAN*, 2005.

[75] Vijayan Sugumaran and Veda C. Storey. An ontology-based framework for generating and improving database design. In *Proceedings of the 6th International Conference on Applications of Natural Language to Information Systems-Revised Papers*, NLDB '02, pages 1–12, London, UK, 2002. Springer-Verlag. ISBN 3-540-00307-X. URL `http://portal.acm.org/citation.cfm?id=645757.758149`.

[76] Stéphane Jean, Hondjack Dehainsala, Dung N. Xuan, Guy Pierra, Ladjel Bellatreche, and Yamine Aït-Ameur. OntoDB: It is Time to Embed your Domain Ontology in your Database. Demo presentation, 2007.

[77] Holger Stenzhorn, Edson José Pacheco, Percy Nohama, and Stefan Schulz. Automatic mapping of clinical documentation to snomed ct. In *MIE*, pages 228–232, 2009.

[78] S. Hina, E. Atwell, and O. Johnson and. Secure information extraction from clinical documents using snomed ct gazetteer and natural language processing. In *Proceedins of International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 1–5, 2010.

[79] J Rogers and O Bodenreider. Snomed ct: Browsing the browsers. In R. Cornet and K. A. Spackman, editors, *Proceedings of Conference in Knowledge Representation in Medicine*, 2008.

[80] Dataline software ltd. http://www.dataline.co.uk.

[81] Ihtsdo: International health terminology standards development organisation. http://www.ihtsdo.org/.

[82] SNOMED Clinical Terms User Guide. Technical report, The International Health Terminology Standards Development Organisation, 07 2009.

[83] Ritu Khare, Yuan An, Jiexun Jason Li, Il-Yeol Song, and Xiaohua Hu. Exploiting semantic structure for mapping user-specified form terms to snomed ct concepts. In *Proceedings of 2nd ACM International Health Informatics Symposium (IHI)*, 2012.

[84] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques, 2nd ed.* Morgan Kaufmann, 2006.

[85] Erik Hatcher and Otis Gospodnetic. *Lucene In Action.* Manning Publications, 2004.

[86] R. J. Miller, L. M. Haas, and M. A. Hernandez. Schema Mapping as Query Discovery. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 77–88, 2000.

[87] Lucian Popa, Yannis Velegrakis, Renée J. Miller, Mauricio A. Hernández, and Ronald Fagin. Translating web data. In *VLDB*, pages 598–609, 2002.

[88] Y. An, A. Borgida, and J. Mylopoulos. Discovering the Semantics of Relational Tables through Mappings. *Journal on Data Semantics*, VII:1–32, 2006.

[89] Y. An, A. Borgida, R. J. Miller, and J. Mylopoulos. A Semantic Approach to Discovering Schema Mapping Expressions. In *Proceedings of International Conference on Data Engineering (ICDE)*, pages 206–215, 2007.

[90] Convenient care center `walk in clinic` drexel medicine. http://www.drexelmed.edu/Home/DrexelUniversityPhysicians/MedicalPractices/InternalMedicine/PatientSer .

[91] Drexel university college of nursing and health professionals. http://www.drexel.edu/cnhp/, .

[92] Drexel university college of medicine - obstetrics and gynecology. http://www.drexelmed.edu/Home/AboutTheCollege/DepartmentsCentersandInstitutes/ClinicalDepts/Obstet .

[93] Logician: Knowledge bank. http://knowledge.medicalogic.com/index.jsp.

[94] Family practice management – american academy of family physicians. http://www.aafp.org/online/en/home/publications/journals/fpm.html.

[95] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[96] Ehren Myers. *Rnotes: Nurse's Clinical Pocket Guide*. F. A. Davis Company, second edition, 2006.

[97] Roy Rada, Bruce Blum, Edith Calhoun, Hafedh Mili, Helmuth Orthner, and Sarah Singer. A vocabulary for medical informatics. *Comput. Biomed. Res.*, 20:244–263, June 1987. ISSN 0010-4809. doi: 10.1016/0010-4809(87)90057-7. URL `http://portal.acm.org/citation.cfm?id=28107.28111`.

[98] George Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information, 1956. URL `http://cogprints.org/730/`. One of the 100 most influential papers in cognitive science: http://cogsci.umn.edu/millennium/final.html.

[99] Magesh Jayapandian and H. V. Jagadish. Automated creation of a forms-based database query interface. *Proc. VLDB Endow.*, 1(1):695–709, 2008. doi: http://doi.acm.org/10.1145/1453856.1453932. URL `http://portal.acm.org/ft_gateway.cfm?id=1453932&type=pdf&coll=GUIDE&dl=GUIDE&CFID=47678283&CFTOKEN=22667157`.

[100] craigslist: Philadelphia classifieds for jobs, apartments, personals, for sale, services, community, and events. http://philadelphia.craigslist.org/.

[101] ebay - electronics, cars, clothing, collectibles and more online shopping. http://www.ebay.com/.

[102] Ornsiri Thonggoom, Il-Yeol Song, and Yuan An. Semi-automatic conceptual data modeling using entity and relationship instance repositories. In Manfred Jeusfeld, Lois Delcambre, and Tok-Wang Ling, editors, *Conceptual Modeling ER 2011*, volume 6998 of *Lecture Notes in Computer Science*, pages 219–232. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-24605-0. URL `http://dx.doi.org/10.1007/978-3-642-24606-7_17`. 10.1007/978-3-642-24606-7_17.

[103] Rafael Berlanga Llavori, Ernesto Jim?nez-Ruiz, Victoria Nebot, and Ismael Sanz. Faeton: Form analysis and extraction tool for ontology construction. *IJCAT*, pages 224–233, 2010.

[104] Ross Koppel and David Kreda. Health care information technology vendors' "hold harmless" clause: Implications for patients and clinicians. *Journal of the American Medical Association*, 301(12):1276–1278, 2009.

[105] Allscripts - home. http://www.allscripts.com/.

[106] Health information privacy. http://www.hhs.gov/ocr/privacy/.

[107] Olivier Bodenreider. The unified medical language system (umls): Integrating biomedical terminology. *Nucleic Acids Research*, 32:D267–D270, 2004. doi: 10.1093/nar/gkh061.

[108] Eladio Domínguez, Jorge Lloret, Ángel L. Rubio, and María A. Zapata. Medea: A database evolution architecture with traceability. *Data Knowl. Eng.*, 65(3):419–441, 2008. ISSN 0169-023X. doi: http://dx.doi.org/10.1016/j.datak.2007.12.001.

[109] George Papastefanatos, Panos Vassiliadis, and Yannis Vassiliou. Adaptive query formulation to handle database evolution. In *Forum of the 18th Conference on Advanced Information Systems Engineering (CAiSE 2006), Luxembourg*, pages 5–9. Springer, 2006.

[110] George Papastefanatos, Panos Vassiliadis, Alkis Simitsis, Konstantinos Aggistalis, Fotini Pechlivani, and Yannis Vassiliou. Language extensions for the automation of database schema evolution. In Jos Cordeiro and Joaquim Filipe, editors, *ICEIS (1)*, pages 74–81, 2008. ISBN 978-989-8111-36-4. URL `http://dblp.uni-trier.de/db/conf/iceis/iceis2008-1.html#PapastefanatosVSAPV08`.

[111] Amazon simpledb. http://aws.amazon.com/simpledb/.

[112] Datastore overview - google app engine - google code. http://code.google.com/appengine/docs/python/datastore/overview.html, .

Part VI

Appendix

Appendix A: Sample Clinical Forms

Walk-in Clinic Encounter Forms

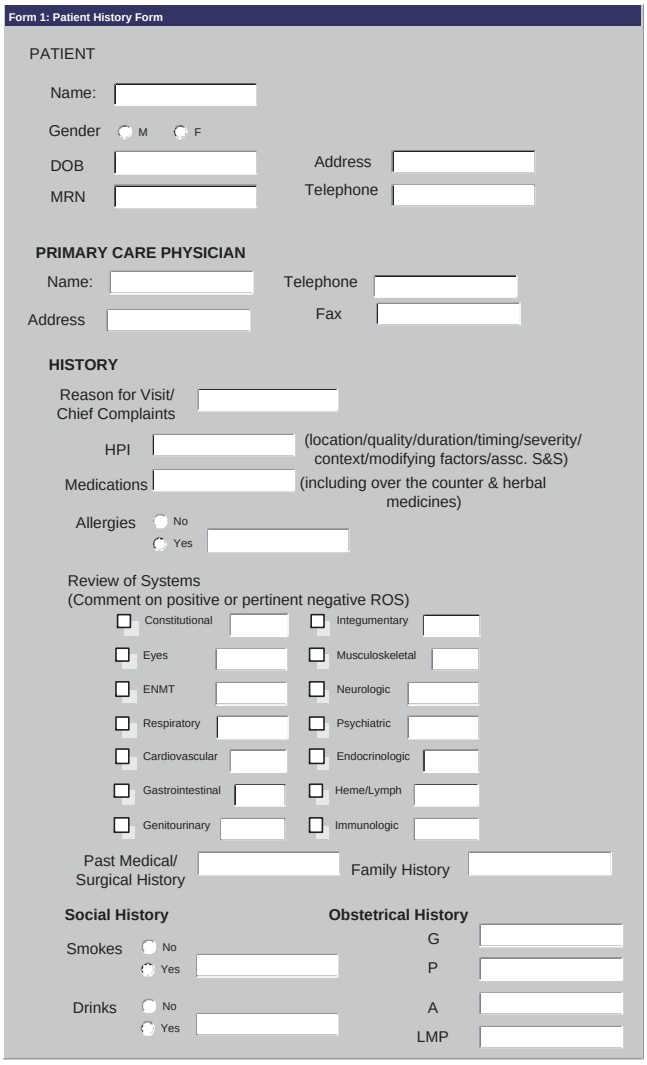


Figure A.1: Dataset 1: Form 1

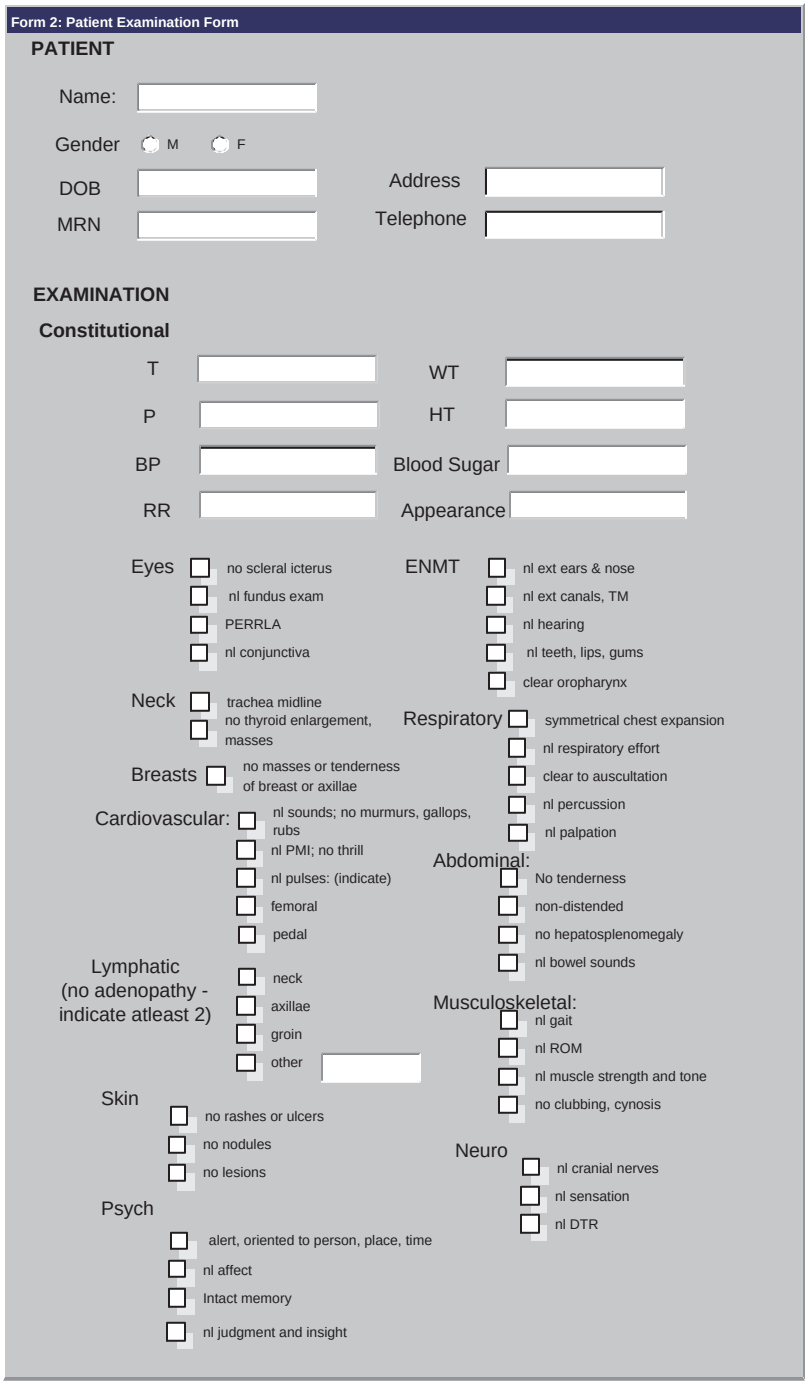


Figure A.2: Dataset 1: Form 2

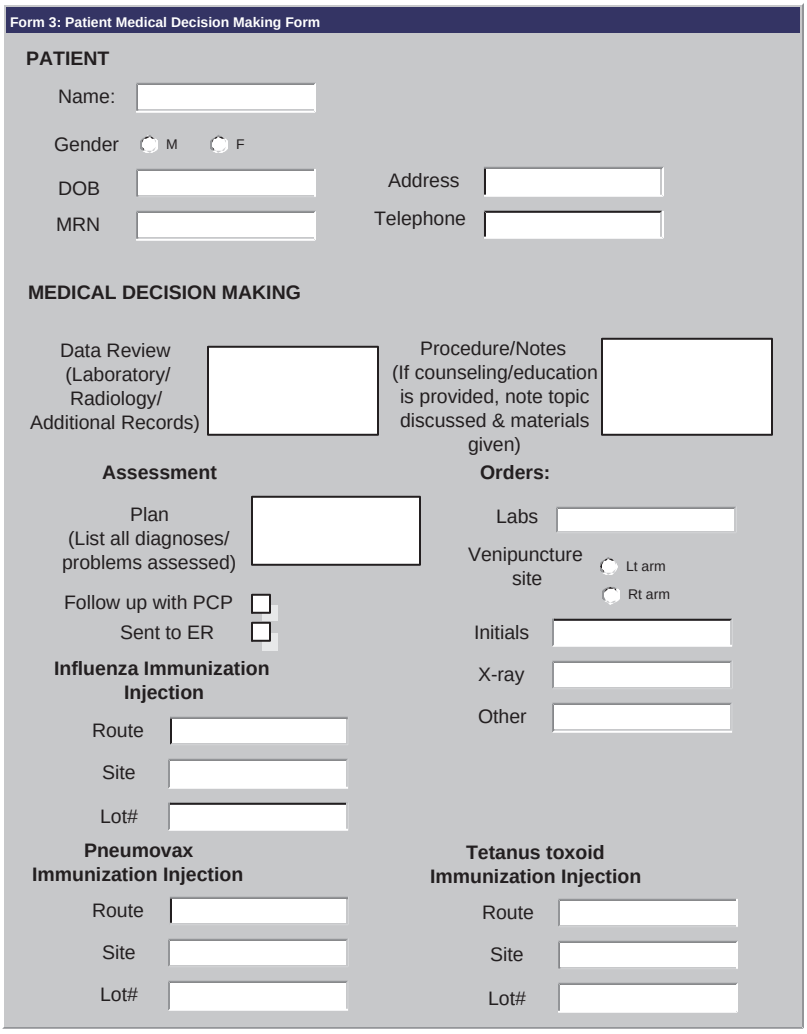


Figure A.3: Dataset 1: Form 3

Patient Admission Forms

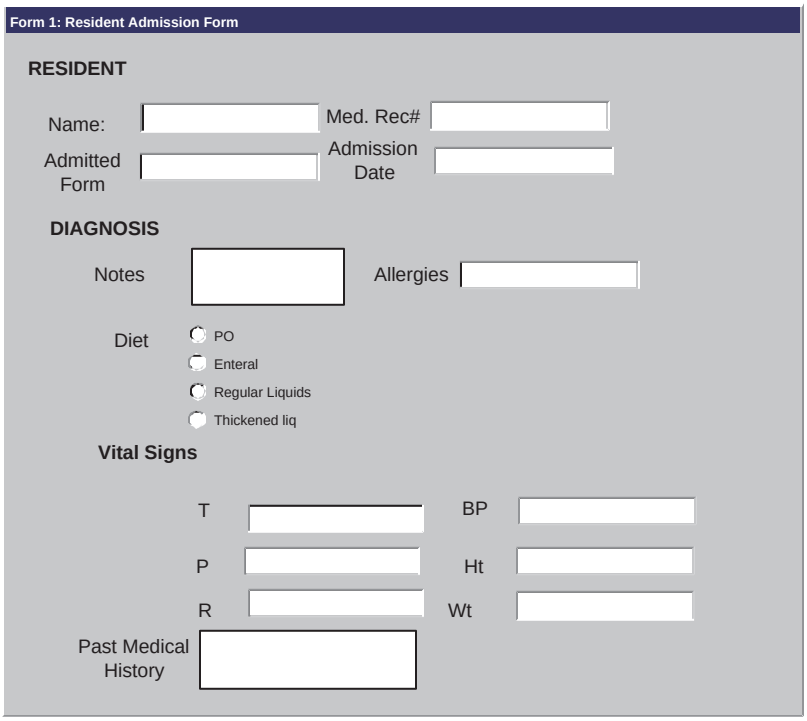
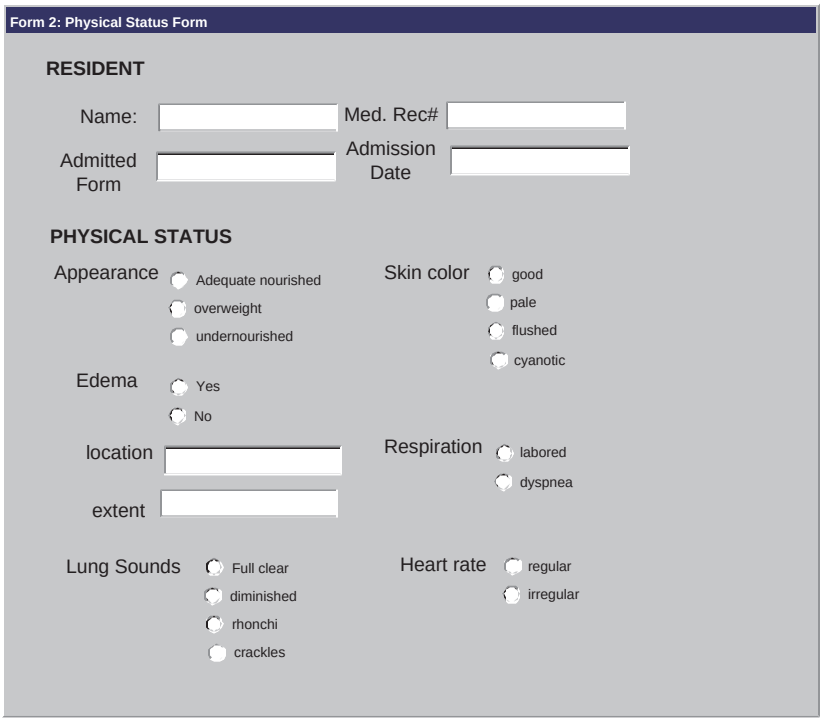


Figure A.4: Dataset 2: Form 1



Condition: The fields 'location' and "extent" are available for data entry only when the user selects "Yes" for "Edema"

Figure A.5: Dataset 2: Form 2

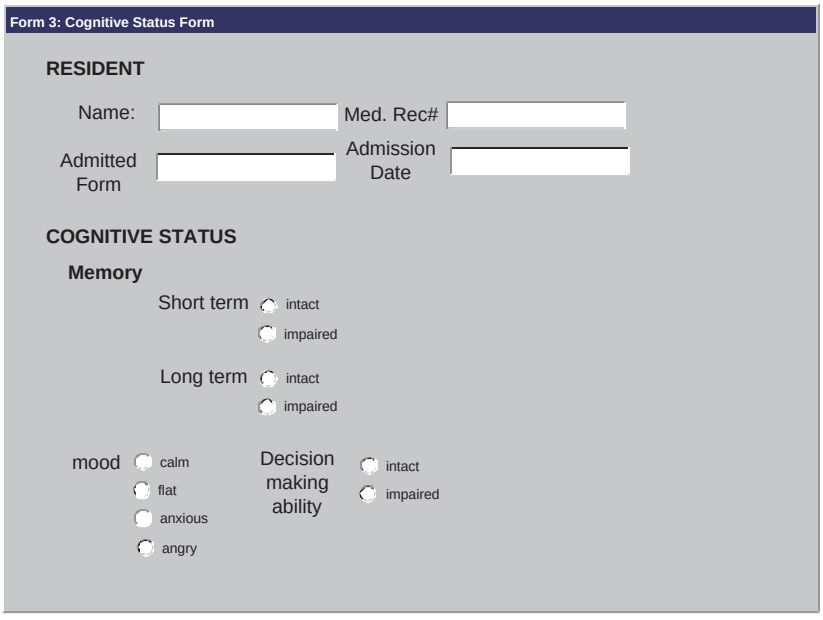


Figure A.6: Dataset 2: Form 3

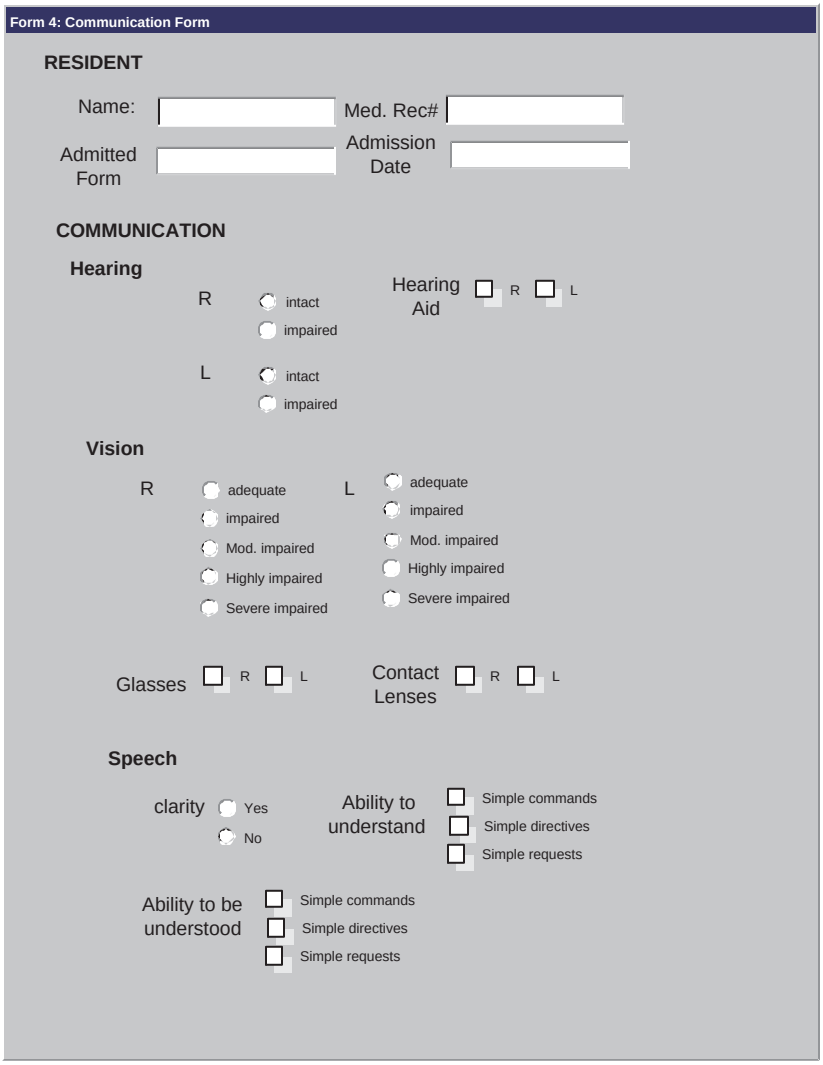
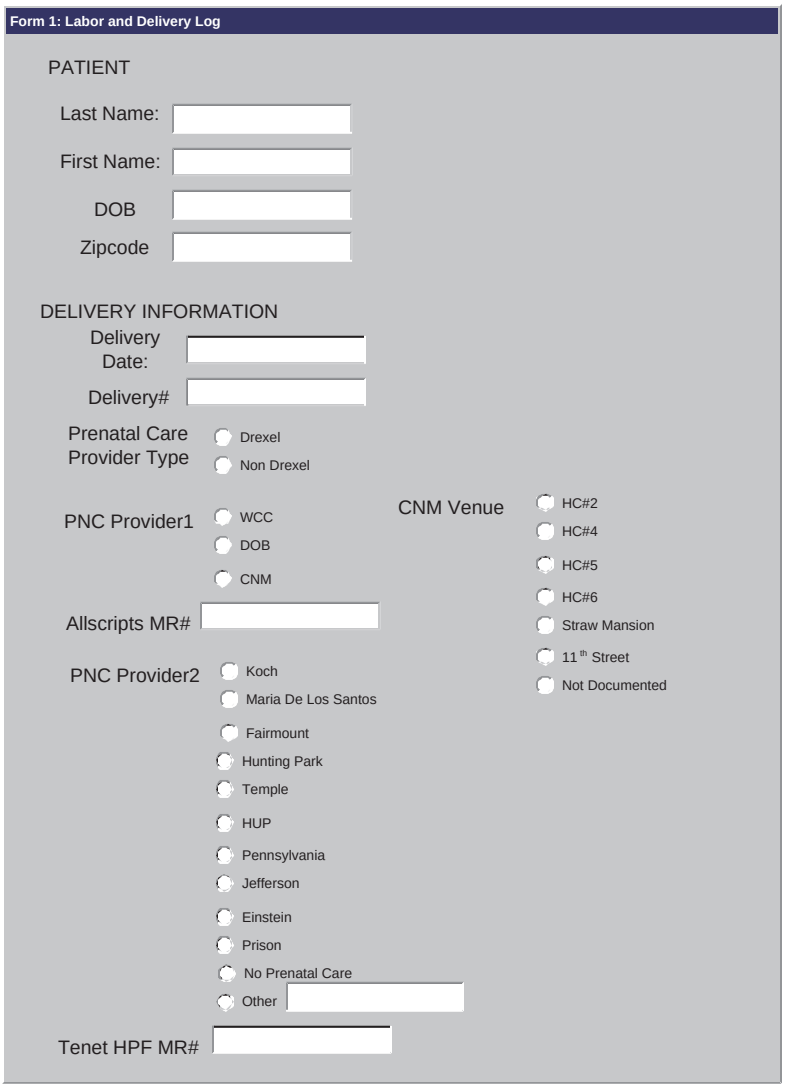


Figure A.7: Dataset 2: Form 4

Labor and Delivery Data-Entry Forms



Conditions:
1. PNC Provider1 and Allscripts MR# are available for data-entry only when the user selects “Drexel” as the PreNatal Care Provider Type
2. PNC Provider2 is available only when the user selects “Non-Drexel” as the Prenatal care provider type
3. Venue is available only when user picks “CNM” as the PNC provider1 value

Figure A.8: Dataset 3: Form 1

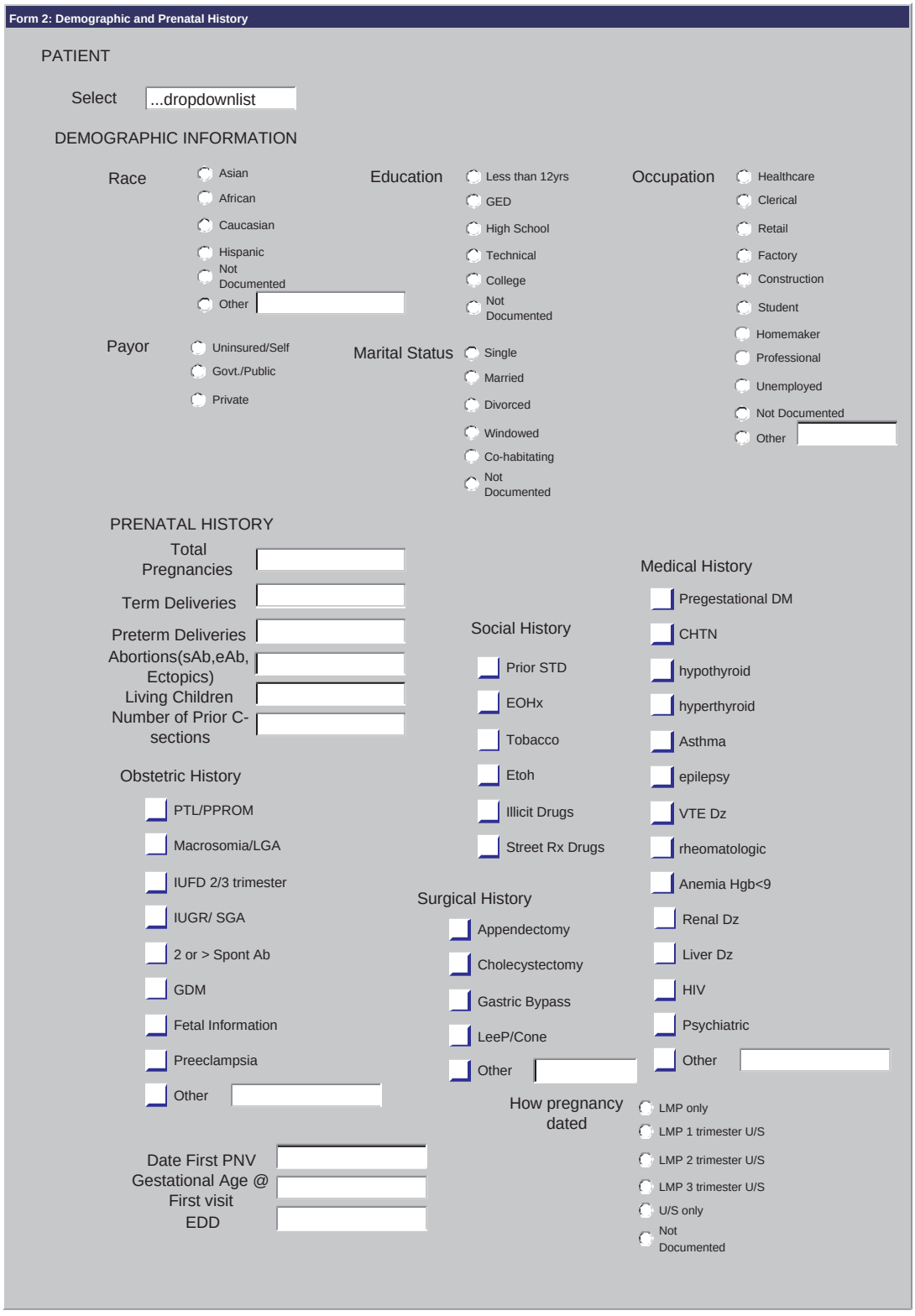
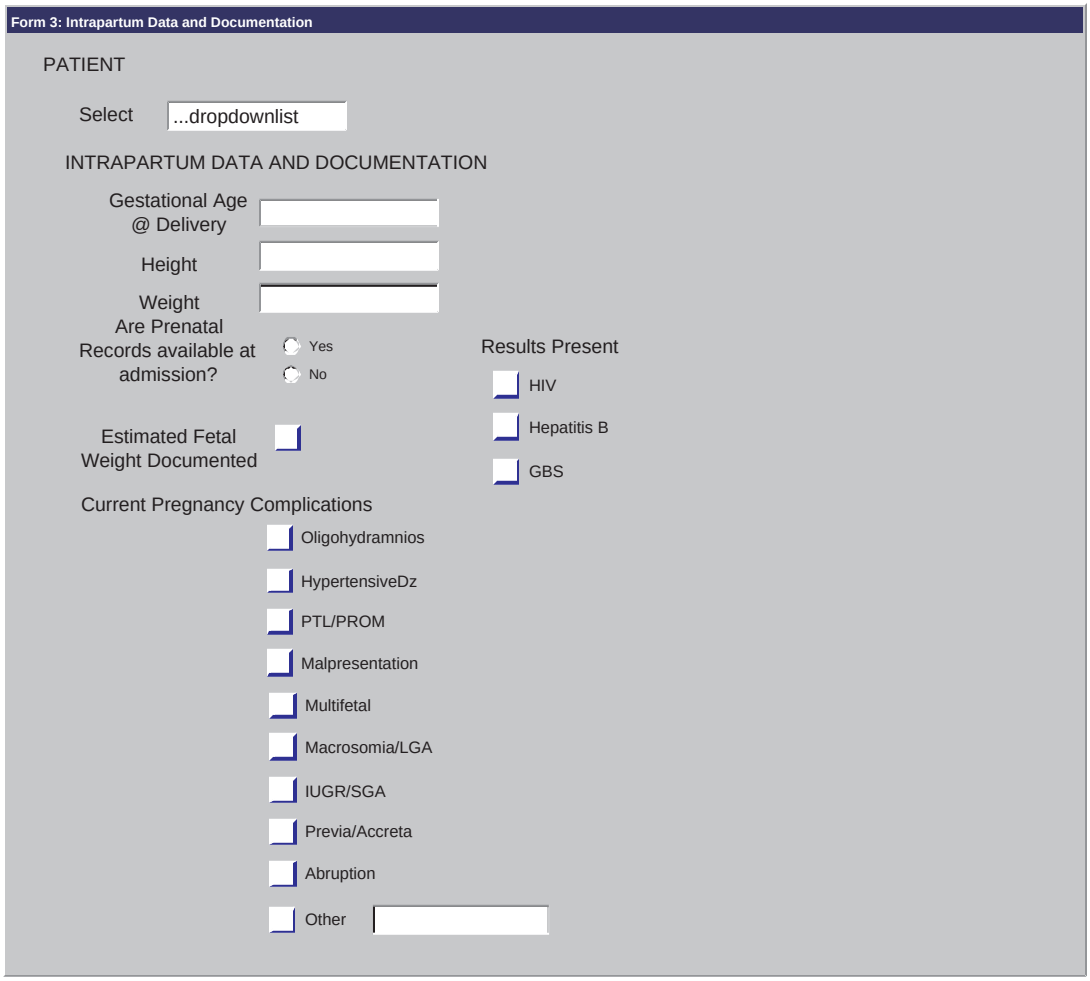


Figure A.9: Dataset 3: Form 2



Conditions:
1. Results present is available for data-entry only when the user selects “Yes” for “Are prenatal records available?”

Figure A.10: Dataset 3: Form 3

Appendix B: Sample Database for a Walk-in Clinic

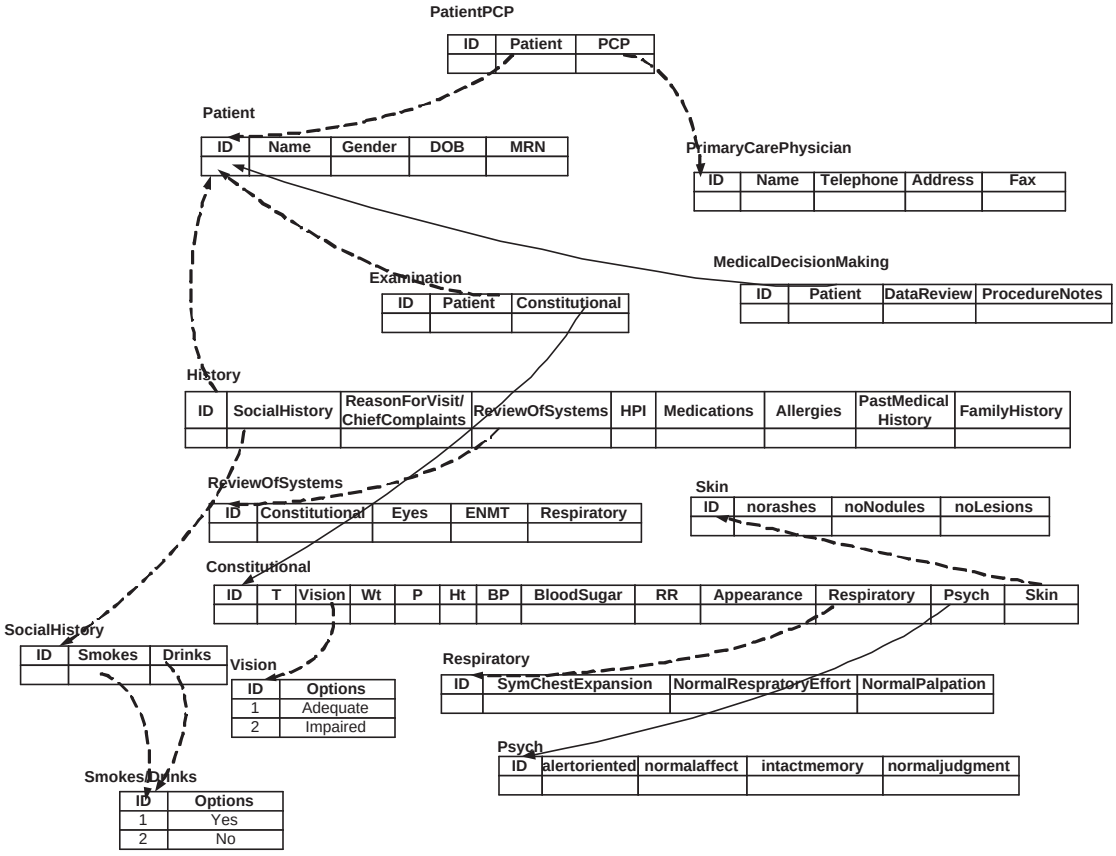


Figure B.1: Part the Database Generated Using the Forms in Figures A.1, A.2, A.3

Appendix C: List of Acronyms and Abbreviations

Table C.1: Medical Acronym List Part 1

Acronym	Expansion
A	abortus
AMT	amount
ASA	Acetylsalicylic Acid
BP	Blood Pressure
BTL	Bilateral Tubal Ligation
BUN	Blood Urea Nitrogen
C-SECTION	Cesarean Section
C/S	Cesarean Section
CA	Cancer
CBC	Complete Blood Count
CC	Chief Complaints
CHTN	Chronic Hypertension
CNM	Certified Nurse-Midwife
CV	Cardiovascular
CVE	Comprehensive Visual Examination
CXR	Screening Chest X-ray
CX	Culture
DERM	Dermatological
DM	Diabetes Mellitus
DTR	Deep Tendon Reflex
DOB	Date of Birth
DX	Diagnosis
DZ	Disease
E	Expiratory
EAC	External Auditory Canal
ECD	Endothelial cell density
EDD	Expected Date of Delivery
EKG	Electrocardiogram
ENMT	Ear Nose Mouth Throat
ENT	Ear Nose Throat
ER	Emergency Room
ESR	Erythrocyte sedimentation rate
ETOH	Ethyl Alcohol
EXP	Expiration
EXT	External
F	Female
FBS	Fetal Blood Sampling
FEVI PEF	Embey Index
FHX	Family History
FSE	Fetal Scalp Electrode
FVC	Forced Vital Capacity
F/U	Follow-Up
G	Gravida
GASTRO	Gastrointestinal
GDM	Gestational Diabetes Mellitus
GI	Gastrointestinal
GU	Genitourinary
HBA1C	Glucose measurement estimated from glycated haemoglobin
HCG	Human Chorionic Gonadotropin
HGBA1C	Glucose measurement estimated from glycated haemoglobin

Table C.2: Medical Acronym List Part 2

Acronym	Expansion
HPI	History of Present Illness
HT/WT	Height Weight
HT	Height
HTN	Hypertension
HX	History
ICU	Intensive Care Unit
I	Inspiratory
IOL	Induction of Labor
IUFD	Intrauterine fetal death
IUD	Intrauterine death
IUGR	Intrauterine Growth Retardation
IV	Intravenous
KUB	Kidney-ureter Bladder
L	Left
LFT	Liver Function Test
LMP	Last Menstrual Period
LGA	Large for Gestational Age
LPL	Lipoprotein Electroph
LT	Left
LTCS	Low-Transverse Cesarean Section
M	Male
MDI	Metered Dose Inhaler
MED	Medical
MEDS	Medicines
MFR	Mass Fraction
MS	Musculoskeletal
MSK	Musculoskeletal
MRN	Medical Record Number
N	No
N/A	Not applicable
NL	Normal
NP	Nurse Practitioner
NEURO	Neurological
O	Objective
O2	Oxygen
OM	Otitis Media
OP	Outpatient
P	Pulse Rate
PAP	Papanicolaou
PCP	Primary Care Physician
PE	Physical Examination
PEF	Peak Expiratory Flow Rate

Table C.3: Medical Acronym List Part 3

Acronym	Expansion
PERRLA	Pupil Equal Round Reacting to Light
PM	post meridiem
PMD	Private Medical Doctor
PMH	Past Medical History
PMHX	Past Medical History
PMI	Postoperative Myocardial Infarction
PRN	as required
PROM	Premature Rupture of Membranes
PSYCH	Psychiatric
PT	Patient
PTL	Preterm Labor
PULM	Pulmonary
RR	Respiratory Rate
R	Right
REC	Record
RESP	Respiratory
ROM	Range of Motion
ROS	Review of Systems
RRR	Regular Rate and Rhythm
RT	Right
RX	Prescription
S	Subjective
SAT	Saturation
SBG	Sensor Blood Glucose
SGOT	Serum Glutamic-Oxaloacetic Transaminas
SHX	Social History
SOB	Shortness of Breath
SVD	Spontaneous Vaginal Delivery
STD	Sexually Transmitted Disease
T	Temperature
TB	Tobacco
TEMP	Temperature
TC/HDL	High density lipoprotein/ total cholesterol ratio
TG	Triglyceride level
TOL	Trial of Labor
TM	Tympanic Membrane
U	Upper
U/A	Unavailable
UGI	Upper Gastrointestinal
UO	Urinary Output
VAVD	Vacuum-Assisted Vaginal Delivery
VTE	Venous Thromboembolism
X-RAYS	Radiographic Imaging Procedure
X-RAY	Radiographic Imaging Procedure
Y	Yes
WCC	White Cell Count
WNL	Within Normal Limits
WT	Weight
#	identifier

Table C.4: List of Thesis Abbreviations

Abbreviation	**Expansion**
API	Application Programming Interface
CASE	Computer Aided Software Engineering
DB	Database
DDL	Data Definition Language
DIY	Do-it-yourself
DOM	Document Object Model
EHR	Electronic Health Record
IR	Information Retrieval
ER	Entity Relationship
fEHR	Flexible Electronic Health Record
FK	Foreign Key
HIT	Health Information Technologies
HMM	Hidden Markov Models
HTML	Hypertext Markup Language
nvr	Null Value Ratio
PK	Primary Key
qtf	Quality Tuning Factor
SNOMED CT	Systematized Nomenclature of Medicine–Clinical Terms
SQL	Structured Query Language
UMLS	Unified Medical Language System
WYSIWYG	What you see is what you get
XML	Extensible Markup Language