Chapter 5

The Implementation and Evaluation of TransFER

All computer applications are tools intended to perform a function or to help users perform a function. As such, a new computer technology should be judged by its ability to (1) perform a useful function that could not be performed previously, (2) perform a function more effectively than it could be performed previously, or (3) perform a function more efficiently than it could be performed previously. By some criterion, the new technology should constitute an *improvement* over the status quo. TransFER is a computer tool intended to facilitate the sharing of decision-support applications that require clinical data. Although currently available methods of sharing decision-support applications leave much to be desired, as discussed in Chapter 2, the TransFER methodology must demonstrate improved performance before one can advocate its use. This chapter describes the general criteria by which the methodology of TransFER was judged and the specific means by which an implementation of TransFER was evaluated.

5.1 A Controlled Experiment

I conducted an experiment to evaluate the utility and generality of the TransFER query-translation methodology. The hypotheses of the experiment were (1) that the TransFER methodology could map a reference schema of clinical data to three independently designed relational database schemas, all of which represent the same data set, and (2) that the TransFER methodology automatically could translate queries formulated against the single reference schema to queries that could be executed against each of the relational schemas. Specifically, the experiment entailed the following steps:

1. Three groups, each consisting of one clinician and one database expert, designed a relational database schema based on the same clinical data set.

2. I designed a global conceptual schema in the FER conceptual modeling language to represent the same clinical data set.

3. A small group of physicians informally formulated a set of queries to retrieve clinically relevant information represented in the data set. I encoded these queries in the ReFER query language in a manner consistent with the conceptual FER schema.

4. I mapped the conceptual FER schema to each relational schema using the extended relational algebra mapping language.

5. The TransFER compiler translated the ReFER queries to three sets of SQL queries such that each set was consistent with one of the relational database schemas.

6. I manually translated the ReFER queries to three sets of "gold-standard" SQL queries, which I compared to the TransFER-generated queries with respect to correctness and efficiency.

The experiment allowed me to evaluate the TransFER methodology in the context of a realistic clinical data set and with respect to heterogeneously implemented databases. The experiment addressed several specific questions:

1. Is the ReFER query language sufficiently powerful to capture the semantics of the specified clinical queries?

2. Is the relational algebra mapping language sufficiently powerful to map the FER schema of the test data set to each relational schema? If it is not, what are the sources of mapping "failures?"

3. Can one specify the mappings between the FER schema and the relational database schemas declaratively, or must one resort to procedural specifications (that is, external functions)?

4. Does the TransFER compiler accurately translate the ReFER queries to semantically equivalent SQL queries?

5. Are the SQL queries that the TransFER compiler generates comparable in efficiency to manually encoded SQL queries?

The high-level goal of the evaluation was to determine whether the TransFER methodology is superior to existing methods for sharing applications that query clinical databases. To answer this question, I characterized the existing methods, I defined specific performance criteria that could differentiate TransFER from the existing methods, I built a prototype implementation of TransFER, and I assessed the prototype with respect to these performance criteria. The remainder of this chapter describes each of these steps.

Section 5.2 discusses deficiencies of current methods for sharing decision-support applications and defines criteria by which one should evaluate a competing technology. Section 5.3 describes the prototype implementation of TransFER that I used to evaluate the methodology. Section 5.4 describes fully the design of the evaluation experiment that I performed and Section 5.5 presents the results of that experiment.

5.2 Evaluation Criteria for TransFER Technology

Medical decision-support applications that rely upon clinical data currently can be shared in one of several general ways. As discussed in Chapters 1 and 2, none of these methods is satisfactory. In order to outline the criteria by which TransFER may be judged superior to existing methods, it is useful to review the existing methods briefly.

> **Method 1**: Manual query execution. In this strategy, used by MYCIN [Buchanan and Shortliffe 1984]and QMR [Miller and Masarie 1990], the physician user interprets all of the application's information requests and responds to them manually by entering patient-specific data that the user knows or that is documented in the medical record.

Method 2: Manual query translation. In this strategy, used by the Arden medical knowledge-representation language [Hripcsak, et al. 1990], each clinical data query that appears in an application is manually translated by a database programmer so that the new query is consistent with the local clinical database implementation.

Method 3: Automated procedural query translation. This strategy, used by the Physician Workstation (PWS) environment [Annevelink, et al. 1992], automatically translates queries issued against a reference data model to a set of operations that retrieves data from an existing database. This method is similar to the TransFER methodology, except that the query translations in the PWS environment are based on *procedural* mappings, whereas TransFER query translations are based on declarative mappings (recall that declarative mappings allow queries to be optimized automatically).

All three methods have deficiencies with respect to at least one of three general properties: *automation*, *efficiency*, or *latitude*. The definition of each property and the trade-offs among the properties are important to consider in comparing a new technology for sharing clinical queries to existing technologies.

5.2.1 Automation

When a mismatch exists between the query model of a decision-support application and the query model of a local database implementation, several existing methods of reconciliation require that the queries be *manually* transformed. Manual query execution and manual query translation both involve labor-intensive translations: Manual query execution requires that users interpret and manually respond to all application queries; manual query translation requires that programmers manually translate all queries that appear in an application. The goal of automation is to relieve users and programmers of tedious and error-prone labor. The manual translation task is especially onerous when it must be performed frequently. Manual query execution and manual query translation require human intervention each time an existing query is issued and each time a new query is added, respectively. Thus, both methods entail a relatively high *variable cost* for reconciling query-model mismatches. A query-translation method would constitute an improvement in automation if it either reduced the variable cost by facilitating the transformation of queries or if it changed the variable cost to a fixed cost by automating the translation of queries based on encoded mappings.

An important assumption of the automation property is that the query translations are *correct*. Correct translations may be defined informally as those translations that preserve the semantic equivalence of source and target queries. In general, semantic equivalence between alternative representations of queries is difficult to verify, but I propose an empirical method for testing equivalence in Section 5.5.3. Although automation does not necessarily preserve semantic equivalence, manual methods do not guarantee correct translations either. Because “source” queries within shared applications often are encoded informally and idiosyncratically, the meanings of the queries as intended by their authors may be unclear or ambiguous. In these cases, personnel who are translating the queries manually cannot be certain that they have preserved the semantics of the original queries because they may not know definitively the semantics of the original queries [Pryor and Hripcsak 1994].

5.2.2 Efficiency

When disparate query models are reconciled via the translation of queries, either manual or automated, it is desirable that the resulting queries be as efficient as possible. In some cases, the time cost of evaluating queries is critical to the objectives of the application. For example, ICU monitors that adjust ventilator settings in real time cannot spend tens of seconds performing complex queries against disk-resident data files. In general, the costs of executing a query depend on where the data resides, how the data is represented, how the query is decomposed into subparts, which agents execute which subparts, and in which sequence the subparts are executed [Ullman 1989 Chapter 11].

The existing methods for sharing clinical queries all vary with respect to the efficiency with which the resulting queries can be executed. Manual query execution, which requires that physician users personally “execute” queries by entering data—possibly after looking it up in the patient record—is clearly the least efficient. Manual query translation, which relies upon programmers to manually translate queries that are subsequently executed by a database query engine, takes advantage of DBMS query facilities, but relies upon the ability of programmers to formulate efficient queries. Although professional programmers are the best personnel for this task, certain clinical queries must be translated into multiple subqueries and the optimal ordering of the subqueries may depend upon knowledge that the programmers do not have or that is subject to change, such as the sizes of data files and indexes. Automated procedural query translation requires that programmers specify the mappings between the query model known to decision-support applications and the query models of underlying databases; the query engine subsequently uses the mappings to translate queries from one model to another automatically. However, the query engine cannot perform certain optimizations because the mappings between query models are specified as *procedural* functions that cannot be decomposed, reordered, and recombined to achieve maximum efficiency [Annevelink, et al. 1992]. To maximize the potential for optimization, a *declarative* representation of query-model mappings is necessary. Because a query optimizer can inspect and manipulate declarative mappings based on an encoded understanding of the operations within the mappings, the optimizer can apply information about the semantics of the data, the state of the database, and the capabilities of the DBMS to determine an optimal sequence and allocation of the operations.

5.2.3 Latitude

A large variety of legacy clinical databases exists. The objective of any method for sharing clinical queries is to support the mapping of queries to as many of these databases as possible. I refer to the proportion of legacy database implementations that a method can support as the method's *latitude*. The latitudes of the existing methods for sharing decision-support applications are quite large, because translation techniques that rely on manual intervention or procedural programming are quite powerful. However, the existing methods achieve wide latitude by sacrificing automation or efficiency (as I discussed in Sections 5.2.1 and 5.2.2).

It is unlikely that an automated query-translation technique based on declarative mappings (such as TransFER) can achieve wider latitude than the existing query-translation methods. Because the ERA mapping language is confined to a small set of relational algebra operators, the transformations supported by these operators are limited to relational database implementations. However, limiting the mapping language to a small set of formal relational operators confers a theoretical foundation to the query-translation process, which enables the process to validate and to optimize automatically the query translations that it produces. This theoretical foundation results in greater automation and efficiency *for relational database implementations* than is achieved by existing methods for sharing clinical queries. The database implementations that TransFER does not support (that is, nonrelational implementations) must default to one of the existing query-translation methods (such as manual query translation), but, in doing so, remain no worse off than they are currently. Relational database implementations, meanwhile, reap whatever automation and efficiency benefits are conferred by TransFER's formal methodology. Therefore, if I can show that TransFER is superior to existing query-translation methods for just relational databases, I will show that TransFER provides an improvement over the status quo.

5.2.4 Performance criteria for TransFER

Based on the demonstrated automation, efficiency, and latitude of existing methods for sharing clinical queries, one can enumerate a set of specific performance criteria by which the TransFER methodology may be compared directly to these existing methods. The criteria embody the general properties of automation, efficiency, and latitude in a manner that may be objectively evaluated in the context of a controlled experiment. The five criteria and their informal definitions are as follows:

1. **Latitude:** the ability of the TransFER mapping language to fully map the FER schema of clinical data to most legacy relational databases, and, therefore, to support automated query translation. Poor latitude with respect to relational database implementations would indicate that TransFER cannot yet replace existing query-translation methods at *any* provider sites and, therefore, constitutes little improvement over the status quo.

2. **Declarativeness:** the ability of the TransFER mapping language to map the FER schema to most legacy relational databases without requiring

extensive use of external functions. Because external functions are procedurally specified and encapsulated, they cannot be manipulated by the query optimizer, whereas combinations of relational algebra operators can be inspected and reordered by the optimizer. Also, because the evaluation of external functions must occur in the host environment, it cannot benefit from indexes and other query-optimization features of commercial relational DBMSs. The extensive use of external functions (that is, minimal declarativeness) would indicate that TransFER is no more effective for translating queries than existing methods based on procedural mappings.

3. **Correctness:** the ability of the TransFER compiler to generate target queries that are semantically consistent with the source queries. Systematic errors in query translation may be due to unsound algorithms used by the compiler. The generation of incorrect queries would indicate that the TransFER method is unreliable and cannot replace existing methods.

4. **Efficiency:** the ability of the TransFER query optimizer to generate target queries that are efficient enough for client applications. For example, the execution of certain queries must be sufficiently fast to be performed as background processes in an integrated clinical alerting system, such as the HELP system [Kuperman, et al. 1991]. Poor efficiency would indicate that manual translation of queries is still required for database access to be fast enough for decision-support applications.

5. **Expressiveness:** the ability of the FER data model and the ReFER query language to represent the semantics of clinical queries that are typically required by decision-support applications. Inadequate expressiveness necessitates that decision-support applications perform certain data-manipulation operations themselves that otherwise could be performed more efficiently by the DBMS if the ReFER query language could express the operations. Inadequate expressiveness precludes the optimal allocation of operations between an application and a DBMS because the operations are not known to the TransFER optimizer and to the TransFER query engine.

For the TransFER methodology to be an improvement over the status quo, it must satisfy these performance criteria, which effectively constitute a lower bound for improved functionality. I assert that, although satisfying the performance criteria in the context of a controlled experiment does not prove that TransFER is generally applicable, it does suggest that TransFER is superior to existing methods and that more exhaustive evaluations are warranted.

To test TransFER with respect to the performance criteria, I conducted a controlled experiment that pitted a prototype implementation of TransFER against several heterogeneous database implementations.

5.3 Implementation of the TransFER Methodology

To evaluate the TransFER methodology, I implemented a prototype that is consistent with the specifications of TransFER as described in Chapters 3 and 4. In this section, I describe the architecture of the prototype and the development environment in which it was implemented.

5.3.1 Architecture

I implemented the TransFER methodology as a set of modules that reads a ReFER query and returns data from a relational database. The implementation is functionally divided into three *compiler modules* , which translate the ReFER query to an optimized relational database program, and one *runtime module*, which executes the database program and retrieves information stored in the relational database. Figure 5.1 illustrates the architecture of the prototype.

The compiler modules consist of (1) a TransFER attribute-grammar processor, which converts the input ReFER query into a site-specific relational algebra query, (2) a query optimizer, which eliminates redundant operations from the relational algebra query, and (3) an SQL translator, which converts the relational algebra query to a database program consisting of one or more SQL queries. The site-specific attribute-grammar processor is, itself, composed of three components: the set of site-independent semantic functions (which I discussed in Section 4.3 and enumerated in Appendix B), the site-specific mapping expressions (which database programmers defined for the local relational database), and the global FER schema (which application developers used to encode the ReFER queries). Before the mapping expressions are integrated into the attribute grammar, a mapping validator confirms that they are syntactically and semantically correct. This step is necessary because the mapping expressions are *user-defined* components of the attribute-grammar module, and therefore subject to error. For example, the mapping validator checks that all tables and fields referenced in the mapping expressions appear in the relational schema and the set of mapping expressions includes mappings for all entity and relationship types in the FER schema (although not shown in Figure 5.1, the mapping compiler uses both the FER schema and the relational schema to perform these validations). The FER schema is integrated into the attribute grammar to validate that all entity types, relationship types, and relationship functions that appear in ReFER queries are, indeed, parts of the global FER schema.

The query optimizer traverses the relational algebra expression tree generated by the attribute-grammar processor and applies any simplifying transformations that are allowed (based on the theory of PROJECT-equivalence described in Section 4.4). These transformations typically replace join and union operations with simpler subexpressions. The optimizer currently applies the transformation rules in a pre-defined order and traverses the expression tree only once.

The SQL translator subsequently traverses the simplified expression tree and identifies any relational algebra operations whose results must be stored in a temporary table; these operations include all external function calls and any relational algebra operators whose results cannot appear in the "where" clause of SQL queries (such as the UNION and AGGREGATION operators). The SQL translator separates these operations into distinct relational algebra expressions and external function invocations, translates the relational algebra expressions to SQL, and specifies the names and structures of temporary tables to store intermediate results. The output of the SQL translator is a database program consisting of a correctly sequenced list of SQL queries, external function calls, and database loading operations that implements the semantics of the original ReFER query. In this sense, the database program constitutes the "compiled" ReFER query.

The runtime module executes the database program. If the database program consists of multiple operations, the runtime engine stores the intermediate results of these operations as temporary tables in the relational database; subsequent SQL queries and external-function calls operate on these tables. The last operation in the database program generates the result of the query, which is printed to the screen.

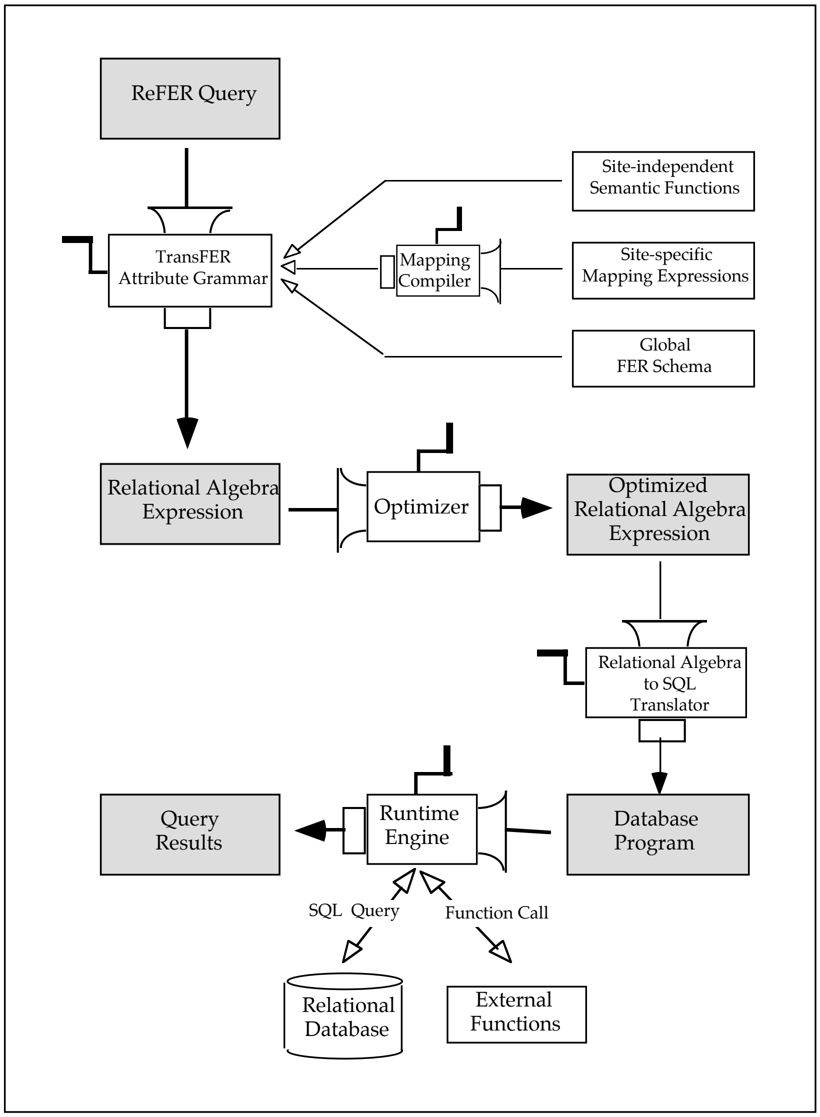

Figure 5.1: Architecture of the TransFER prototype. The translation and execution of a ReFER query may be traced by following the figure from top to bottom. The "hand crank" icons represent code modules and the boxes represent the inputs and outputs of code modules. Shaded boxes represent sequential versions of a ReFER query as it is translated and executed.

5.3.2 Development environment

I implemented the TransFER prototype in the Prolog programming language, using the Quintus Prolog[1] development environment (Version 3.2). Quintus Prolog is an implementation of Edinburgh Prolog that includes a compiler, a debugger, and many built-in utility and input/output functions. The implementation runs on a Sun SPARCstation under the SunOS operating system (Release 4.1). The prototype accesses relational databases implemented in the Sybase[2] relational DBMS (Server Release 10.0), also running on a Sun SPARCstation under SunOS 4.1. A client–server architecture is employed, such that the Prolog query-translation and runtime modules run on one processor and the Sybase DBMS on another. The Prolog application issues SQL queries to the Sybase database via the ProDBI[3] database interface (Version 3.2). ProDBI is a set of Prolog routines that connects to a Sybase server, issues SQL queries to a Sybase database, and converts retrieved query results to Prolog data structures. In addition to Sybase, ProDBI also provides access to Informix, Ingres, and Oracle relational databases.

I encountered several limitations of the development environment during my implementation of TransFER. The first was that the Sybase query engine cannot process SQL queries with more than 16 joins. Certain of the SQL queries generated by the TransFER compiler in the course of the evaluation did, in fact, contain more than 16 joins, and Sybase simply could not execute these queries . Future versions of the compiler should detect relational expressions that produce SQL queries with more than 16 joins and decompose these expressions into smaller subexpressions that yield "valid" SQL queries. Another limitation was that the ProDBI Prolog-to-Sybase interface did not import date-valued data correctly. Specifically, database values having the Sybase data type "datetime" (a data type that specifies a date and a time of day, such as "10/07/92 07:30") were returned to Prolog with their values truncated to just the date component (for example, "10/07/92"). This loss of time-of-day information caused certain temporal comparisons that appeared in ReFER queries to be incorrect. I

---

[1]Quintus, Inc., 47212 Mission Falls Court, Fremont, CA 94539

[2]Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608

[3]ProDBI is also available from Quintus, Inc.

circumvented this problem by converting all datetime-typed values to canonically formatted text strings before performing any temporal comparisons (such as "92/10/07 0730"), but this solution introduced extra runtime operations. Optimal performance will require a database-interface library that imports datetime-typed values without truncation.

Although I implemented the prototype on a Sun SPARCstation running the SunOS operating system, the application could be ported with minimal or no modifications to any environment jointly supported by Quintus Prolog, ProDBI, and Sybase. Currently, these environments include the following hardware platforms (with operating systems in parentheses): DEC Alpha (OSF/1), HP 9000–series 700/800 (HP-UX), IBM RS/6000 (AIX), SGI (IRIX 5), and Sun SPARC (Solaris). Furthermore, because the SQL-translator module generates standard ANSI SQL queries, one could use the TransFER prototype to access Informix, Ingres, or Oracle databases running on any of the supported hardware and operating-system environments. Although I have not yet attempted to run the TransFER prototype in any other hardware, operating-system, or DBMS environment, a significant degree of portability theoretically exists.

5.4 Structure of the Experiment

In this section, I describe the protocol of the experiment that I conducted using the prototype implementation. The objective of the experiment was to evaluate the TransFER query-translation methodology with respect to the performance criteria defined in Section 5.2.4. The experiment was a controlled trial that posed a realistic, though not exhaustive, set of database mapping problems to the TransFER methodology. I described the general design of the experiment in Section 5.1. I now discuss each sequential phase of the experiment in detail.

5.4.1 Phase 1: Independent clinical database design

Three groups of experimental subjects were given the same clinical data set and asked to design a relational database schema to represent that data set. Each group consisted of one clinician and one database expert, recruited from the Section on Medical Informatics or the Computer Science Department at Stanford. All subjects were volunteer graduate students. To avoid bias in the design of the database schemas, I chose volunteers who had minimal knowledge of the

TransFER query-translation methodology, including the FER data model and the ReFER query language.

The clinical data set that I provided to the subject groups represented a *subset* —in terms of both size and scope—of the patient-specific data that typically appear in clinical information systems. The data set described the hospital admissions and inpatient stays of two actual patients in the ICU of a major hospital. The data set consisted of approximately 40 distinct data elements (for example, "Medications") and 300 total data values (for example "Albuterol MDI 4 puffs q4"). The types of information represented included demographics, medical history, vital signs, laboratory results, and medication records. Obviously, many data that typically appear in electronic medical records are omitted, including orders, radiology reports, progress notes, and administrative information. However, the data elements that were included constituted most of the *structured* data that appear in electronic medical records today and that represent the difficult modeling challenges faced by designers of relational clinical databases. I assert that it is the modeling of these data that leads to representational heterogeneity among clinical databases. The complete data set and the instructions to the volunteer groups appear in Appendix C.

I provided the data set to the subject groups in a "representationally neutral" format so as to minimally influence the design of the relational schemas. To this end, I formatted the primary data much as they would appear in a paper medical record: The history and physical were presented in narrative form, the laboratory results were presented as episodic sets of test values, and the vital signs and ICU monitor data were presented as charted flow sheets. Also, all data values were represented by natural-language descriptors, such as "Blood Glucose," rather than "GLU," to avoid biasing the choice of coding schemes. In providing the data set in a representationally neutral format, I hoped to simulate the database design process that takes place at clinical institutions when paper-based medical records are modeled in clinical information systems. The intent of simulating the database design process was to achieve a representative degree of heterogeneity among the resulting schemas.

The result of the database design phase was a set of independently specified relational schemas. Each schema consisted of a set of valid SQL data-definition statements that could be processed by a relational DBMS to create the database structure. For the purposes of this experiment, all three groups used the Sybase DBMS. After implementing its database, each group also populated the database with the clinical data set, employing whatever coding scheme it chose to represent the data elements. The groups were given the flexibility to use existing codes drawn from standard terminologies, such as ICD9 and SNOMED, or to define their own ad hoc codes, provided that the meanings of all codes were documented.

The specific constraints imposed on the volunteer groups in designing the relational databases are summarized by the following rules:

1. The relational schemas had to represent all of the data provided in the clinical data set.

2. The groups could *enrich* the data set by adding types of objects that were not already represented, as well as instances of those types. However, the groups could not *expand* the data set by adding instances of any object types that were already represented.

3. No group could consult with any other group.

5.4.2 Phase 2: Design of the FER schema

In this phase, which took place concurrently with Phase 1, I modeled the same clinical data set given to the subject groups using the FER data model. The result was an FER schema that included all of the entity types, relationship types, and IS-A connections represented (explicitly or implicitly) in the clinical data set. Phase 2 took place before I had seen any of the relational schemas designed by the subject groups or any of the clinical queries formulated by the domain experts (see Section 5.4.3). This sequence prevented my design of the FER schema from being biased by knowledge of either the specific relational schemas to which the FER schema would be mapped or the data-access paths that the specific queries would require. Because I was evaluating the ability of the FER model and the ReFER query language to represent a wide range of database designs and clinical queries, the validity of the experiment depended on avoiding bias in these regards. The FER schema that I defined for the clinical data set appears in Appendix E.

5.4.3 Phase 3: Formulation of clinically relevant queries

Two physicians independently formulated a set of queries requesting clinically relevant information known to be available in the data set. I subsequently encoded the queries in the ReFER query language such that they were consistent with the conceptual FER schema I had specified.

The clinicians were instructed to generate queries that were representative of the varied uses of clinical data, including direct patient care, clinical practice guidelines, clinical alerts, and outcomes assessment. The queries were based on the data actually available in the clinical data set, rather than on the more complete data set that typically appears in medical records. The data were provided to the physicians in the same representationally neutral format as they were provided to the database-design groups, so that the formulation of the queries would not be biased toward any particular relational database design. This method of query formulation is consistent with the way in which queries are formulated in actual decision-support applications that will be shared: Query authors have no knowledge of the various relational implementations to which their queries will be translated.

The result of the query formulation phase was a set of 23 clinically relevant queries encoded in *natural language*. An example of such a query is "How many hours after Patient #2 was placed on Cefotetan did her temperature fall below 101?" Because there currently exist neither a standard reference model for clinical data nor a standard query language for clinical queries, natural language provided the only site-independent format in which to formulate queries that could be shared among clinical database implementations.

I subsequently translated each of the natural-language queries generated by the physicians to equivalent ReFER queries formulated with respect to the FER schema that I had defined for the clinical data set. Note that I translated the queries to ReFER before I had specified any mappings between the FER schema and the relational schemas so as not to bias my formulation of the queries based on the characteristics of the mappings. The complete set of natural-language queries that the physicians formulated and the ReFER translations of those queries that I specified are listed in Appendix F.

5.4.4 Phase 4: Mapping the relational schemas

Using the extended relational algebra (ERA) mapping language, I mapped the FER schema that I had defined to each of the relational database schemas that the volunteer groups had implemented. Recall that the FER schema and each of the relational schemas were based on the same clinical data set. For each relational schema, the mapping consisted of an ERA expression for each entity type and relationship type in the FER schema. External functions were specified where necessary and were implemented in a general-purpose programming language as part of the TransFER run-time system.

In a realistic application of the TransFER technology, the ERA mappings between the FER schema and a relational database would have been performed at each database site by personnel who are familiar with the semantics of the local database and the FER schema. However, it was not feasible in the context of my experiment for each database design group to perform the mappings, because a suitable user interface in which to specify mappings did not yet exist and because the volunteer subjects did not have sufficient time to learn the details of the TransFER model. For purposes of the experiment, therefore, I assumed that personnel at each site could perform the mappings as accurately as I could perform them myself. I can test the validity of this assumption in subsequent evaluations of TransFER.

5.4.5 Phase 5: Automatic translation of the ReFER queries

For each relational implementation of the clinical data set, I translated the set of ReFER queries to a set of corresponding SQL database programs by using the TransFER compiler. The compiler used the ERA mappings specified for each database implementation. The inputs to the TransFER compiler were the "source" ReFER queries, the FER schema, and the implementation-specific ERA mappings. The output of the compiler was a set of database programs consisting of SQL queries and calls to external functions (when required). The TransFER runtime system executed each set of the query programs against its respective database implementation and I collected and analyzed the results. I present the data produced by the evaluation experiment in the following section.

5.5 Results

Each step of the experiment yielded useful data for (1) understanding the nature of database schema heterogeneity in the context of clinical information systems and (2) assessing the effectiveness of the TransFER technology with respect to the performance criteria.

5.5.1 Database schema heterogeneity

The intent of TransFER is to render transparent to decision-support applications the heterogeneity of legacy clinical databases. The evaluation experiment is only valid, therefore, if the subject groups generated database schemas with realistic and representative degrees of heterogeneity. Such heterogeneity should include naming differences, structural differences, semantic differences, and data content differences [Deen, et al. 1987]. The three generated schemas manifested heterogeneities in each of these categories. The complete relational schemas for the three databases appear in Appendix D.

1. **Naming differences** are characterized by distinct syntactic labels for the same semantic constructs across schemas. Naming differences among the generated clinical databases included numerous variations in table names (for example "meds" versus "Medications") and field names (for example "dose" versus "MedAmount"). Naming differences also were manifested in the coding systems used to populate the databases. For example, one database-design team used SNOMED-III codes {{ref}} to represent all medical concepts, whereas the other two teams used nonstandard numerical codes and text strings.

2. **Structural differences** entail different decompositions of semantic objects into the tables and fields of a relational schema. A structural difference among the database schemas was variation in the normalization of medication data. One schema represented in a single table the medications that patients had received and the properties of those medications (resulting in a denormalized schema); another schema represented data regarding the dosage units and routes of administration of medications in a separate reference table from data regarding the medications that patients received (resulting in a

normalized schema). Clearly, the degree of normalization influences the query operations that are required to access and associate information. Structural heterogeneity also appeared in the representation of drug allergies: One schema recorded all of the allergies for a patient as a single text string in one field of the patient demographics table, whereas another schema represented patients' allergies in a dedicated table that included one row for each allergy (see Figure 5.2).

Schema 1

PATIENT

pt-ssn	first	last	dob	Allergies
149584223	Phil	Smythe	12-2-57	Penicillin, NSAIDs
216426077	Carol	Flynn	3-25-62	Morphine, ASA

Schema 2

PATIENT_ALLERGIES

pt-ssn	Allergy
149584223	Penicillin
149584223	NSAIDs
216426077	Morphine
216426077	ASA

and medication stop times from one of the schemas. For both data elements, the omitted values were assumed to be NULL, although this value was not necessarily consistent with the values represented in the other databases. When a data element that appears in the FER schema is omitted from a database, the database mapper either must make an assumption about the value of the missing datum or must assign a special value to the missing datum indicating that it is unknown.

Given that significant heterogeneity did exist among the clinical database schemas that were generated, the performance of the TransFER technology in the context of the experiment is a meaningful, albeit preliminary, indicator of the utility of TransFER in the "real world." The following sections discuss TransFER's performance with respect to each of the criteria that I defined in Section 5.2.

5.5.2 Latitude and declarativeness

Table 5.1 addresses the performance of TransFER with respect to latitude and declarativeness.

Schema	Mapping Yield (%)	External Function Use (%)
Schema 1	145/183 (79%)	12/145 (8%)
Schema 2	183/183 (100%)	13/183 (7%)
Schema 3	183/183 (100%)	13/183 (7%)

Table 5.1: Success rates for mapping the FER schema to three relational schemas. Mapping Yield indicates the proportion of entity types and relationship types in the FER schema that were mapped successfully to the relational schema. External Function Use indicates the proportion of entity-type and relationship-type mappings that included calls to external functions.

The "Mapping Yield" column in Table 5.1 indicates the proportion of entity types and relationship types in the FER reference schema that I successfully mapped to each of the three relational schemas using the relational algebra mapping language.

Although I was able to map Schemas 2 and 3 completely to the FER reference schema, Schema 1 proved problematic because its designers chose not to model clinical diagnoses and medical procedures in a structured relational format. Rather, the diagnoses and procedures associated with each patient were represented in a single text field as a narrative description of the patient's chief complaint and past medical history. The contents of this field resembled the following text string: "asthma since age 8, left nasal polyp removal 1985, chronic sinusitis since 1980." The extraction of distinct disease and procedure entities from such narratives was beyond the capabilities of the relational algebra or of any external functions available to me. In this sense, Schema 1 was beyond the "latitude" of TransFER. However, of the comparative query-translation techniques, only the manual inspection and interpretation of Schema 1 could have achieved a better result given current technology. In the future, natural-language processing techniques for clinical narratives [Sager, et al. 1994] may become sufficiently powerful and reliable to extract structured information from free-text descriptions. For the present, however, disease and procedure data modeled as free-text narratives are beyond the latitude of any automated retrieval techniques.

The "External Function Use" column of Table 5.1 denotes the proportion of mappings that required at least one external function call to represent accurately the semantics of FER entity types or relationship types. As the data indicate, external functions were required sparingly. Operations requiring external functions included (1) the decomposition of a single text field containing multiple allergies into a set of text fields each containing a single allergy and (2) the data-type conversion of date-valued fields to a format that readily supported temporal comparisons. Given the degree of heterogeneity among the generated relational schemas, the infrequent need for external functions indicates the power of the relational algebra as a mapping language and bodes well for the optimization potential of queries generated by the TransFER methodology.

5.5.3 Expressiveness and correctness

Table 5.2 addresses the performance of TransFER with respect to expressiveness and correctness. The ability of the ReFER query language to express the clinical queries formulated in Phase 3 of the experiment reflects the adequacy of the TransFER model in supporting the data-access needs of decision-support applications.

Parameter	Value
Physician queries encoded in ReFER (%)	70% (16/23)
Translated queries returning correct values (%)	92% (36/39)

Table 5.2: The expressiveness and correctness of TransFER. The percentage of queries successfully encoded in ReFER reflects the expressiveness of the ReFER query language. The percentage of translated queries returning correct values reflects the correctness of the TransFER query-translation methodology.

The data in the first row of Table 5.2 show that seven of the 23 queries specified by physicians as relevant to decision support (30%) could not be encoded in the ReFER query language. Five of the seven queries could not be encoded because ReFER currently lacks constructs to perform certain temporal operations, such as retrieving the three most recent values of a parameter or computing the difference in hours between two time points. For example, the following specific queries (as formulated by the physician experts) could not be supported by the ReFER query language:

1. Retrieve the last 3 arterial blood gas (ABG) measurements for Patient #1 as of 1:00 p.m. on August 27, 1993.

2. Retrieve all of the ABG measurements for Patient #2 within 6 hours before and 1 hour after mechanical ventilation.

The 30% failure rate for encoding clinically relevant queries implies that the addition of more powerful temporal operators to the query language would improve TransFER's ability to directly support certain queries. However, even in the absence of such operators, decision-support applications can use the TransFER model as an interface to clinical databases provided that the applications implement certain temporal operations themselves. For example, to derive the last three ABG measurements for a patient, an application could retrieve *all* of the ABG measurements and subsequently filter out all but the three most recent.

Each of the ReFER queries subsequently was processed by the translating compiler three times, once using each set of schema-specific ERA mappings. The result of each translation was a database program consisting of SQL queries interspersed with external function calls. When executed against the appropriate relational database, each program should have returned the data requested in the original ReFER query. Any discrepancy between the data originally requested and the data ultimately retrieved constituted a correctness failure. Of the 48 potential query translations, 39 produced "valid" database programs. The remaining nine translations produced programs that were not valid because at least one of the generated SQL queries referenced more than 16 relational tables, an arbitrary limit of the database software used for the evaluation (Sybase).

When the 39 valid translations were executed, 36 (92%) produced correct results. The gold standard for correctness was determined by manually inspecting the original data set in its paper-based format. The modest size of the data set made such a manual inspection feasible. Of the three cases in which incorrect data were returned, a different reason was responsible for each failure:

1. One query failed due to an erroneous mapping expression; when I corrected the mapping error and recompiled the query, the correct data was returned. Mapping errors may be subtle and remain undetected in the absence of rigorous testing. This fact points to the need for extensive test suites of standardized data and queries to increase the likelihood of detecting errors.

2. Another query failed because one of the subject groups populated its relational schema in a manner inconsistent with the semantics of the original clinical data set; after we modified the relational data, the same query produced correct results.

3. The third failure occurred because one relational schema contained no data regarding patient allergies, although this data was included in the clinical data set. When I mapped the FER relationship type representing each patient's allergies to this schema, I assumed that patients had no allergies because no data existed from which to derive an appropriate set of allergies. This assumption was incorrect and caused incorrect data to be retrieved by a query. Missing clinical data elements pose a significant problem for the sharing of clinical decision-support applications, because incorrect assumptions regarding the values of such elements may affect the validity of inferences in ways that are not apparent to users. The alternative to making assumptions regarding the values of missing data at the time that a relational schema

is mapped is to assign a special value to those data that notifies the application or the user that the requested information is not available. This alternative allows an application to incorporate the knowledge that certain data is missing into its logic and to respond in an application-specific and appropriate manner (for example, by querying the user).

5.5.4 Efficiency

To assess the efficiency of the relational queries produced by the TransFER query-translation methodology, I compared the 39 TransFER-generated database programs to a corresponding set of 39 database programs that I *manually encoded* for the same ReFER queries. The manually encoded database programs constitute a *gold standard* of efficiency in the sense that I formulated these queries with the intent of specifying the minimal set of SQL operations required to retrieve the requested data. Recall from Section 4.4 that the TransFER query-translation methodology may introduce redundant relational operations into the translations of ReFER queries, which result in relational expressions that are significantly more complex than necessary; the query optimizer subsequently detects and eliminates certain of these operations in an attempt to arrive at a minimal relational algebra expression that has the same meaning as the original expression. The purpose of directly comparing the TransFER-generated database programs to the gold-standard database programs is to assess the effectiveness of the query optimizer in removing unnecessary operations and in automatically generating SQL queries that are as efficient as manually generated queries.

I characterize the complexity of each database program generated by TransFER in terms of two parameters: (1) the number of distinct SQL queries in each database program ("SQL queries/ReFER query") and (2) the *Join Index* of each database program ("Join Index/ReFER query"). The Join Index (JI) of a database program is defined as the total number of joins across all of the SQL queries in the program *plus* two times the number of SQL queries in the program. The JI is an approximate measure of the complexity of database programs that consist of multiple SQL queries. The assumption underlying this metric is that the overhead associated with each distinct SQL query in a database program is roughly equivalent to two relational joins. I use relational joins as a measure of complexity because they typically constitute the bulk of the run-time processing of SQL queries. This definition of the JI implies, for example, that a database program consisting of five SQL queries that each contain one join (JI = 15) is more costly to execute than a database program consisting of one SQL query containing five joins (JI = 7).

Parameter	Manual translation	TransFER translation (no optimization)	TransFER translation (partial optimization)
Average # SQL queries/ReFER query	1.1 (1–2)	5.5 (3–11)	3.5 (1–9)
Average JI/ReFER query	4.4 (2–9)	22.9 (9–54)	17.1 (2–44)

Table 5.3: The complexity of TransFER-generated queries Numbers in parentheses indicate the range of values of each parameter.

Table 5.3 shows the average number of SQL queries and the average join index for the entire set of 39 database programs generated by the TransFER compiler. The parameters were calculated once before any optimizations were implemented (column 3 of Table 5.3) and once after a subset of the proposed optimizations was implemented (column 4 of Table 5.3). The results in column 4 do not represent the truly optimal complexity of TransFER-generated queries because not all of the potential optimizations described in Chapter 4 were implemented at the time I obtained the results. Column 2 of Table 5.3 shows the average number of SQL queries and the average join index for the gold-standard database programs that I generated manually.

It is useful to consider the complexity of the original ReFER queries that formed the basis of the translations characterized in Table 5.3. The 39 ReFER queries that were translated to database programs contained, on average, one term, two declarations, and four predicates. The following ReFER query, therefore, is representative of the average complexity of the 39 translated queries:

```
[ value-of(na) | (pt : Patient), (na : Serum_Na) AND
                 Patient-Test(pt,na) AND
                 last-name-of(pt) = "Smythe" AND
                 time-of(na) > "1995/09/20 0000" AND
                 time-of(na) < "1995/09/21 0000" ]
```

The data in Table 5.3 indicate that the automatically generated database programs (before optimization) are more complex than the ideal (gold-standard) programs by a factor of five to one. Direct comparison of corresponding database programs confirms that superfluous JOIN and UNION operations in the TransFER-generated programs are, in fact, responsible for the disparity in complexity. However, the data also indicate that implementation of just a subset of the optimization rules described in Section 4.4 results in a 20 percent reduction in complexity. This result is promising because it indicates that optimization techniques based on the principles of relational algebra can significantly improve the efficiency of TransFER-generated queries. The development and implementation of additional optimization rules will likely lead to further performance improvements, bringing the efficiency of TransFER-generated database programs closer to that of the gold standard. I will continue to use the join-index metric to gauge progress as the development of optimizations continues.

The definition of formal evaluation criteria, the implementation of a working prototype, and the execution of a carefully planned experiment allowed me to rigorously assess the merits of the TransFER methodology with respect to a non-trivial query-translation problem. In the next chapter, I discuss the value as well as the limitations of the experimental results that I obtained. I also look ahead to future research based on the TransFER model for query translation, and I summarize the contributions of the TransFER methodology to sharing decision-support applications.