Chapter 1

Introduction

1.1 Introduction

One of the biggest challenges in database migration based on a technology change is to provide new data structures that translate all the semantics of the legacy database and nothing else. These structures should not inherit any technology-dependent feature from the legacy data structures. Failing to meet this requirement would provide a database that is flawed from its very start and that will lead to increasing semantic, integrity and performance problems. The most popular migration approach that could be called one-to-one strategy ensures the structural equivalence between both legacy and new data structures. In a file to relational database migration, it consists of converting each record type into a table and each top-level field into a column. Its popularity comes from its extreme simplicity and its low cost both databases have the same schema and converting the programs is particularly easy since each I/O statement is replaced with a functionally equivalent DML sequence. No understanding of the data structures nor of the processing logic is required, so that the translation process can be automated to a large extent. Such an approach naturally yields, but in some exceptional situations, poor data structures that are difficult to maintain and to evolve.

The other approach that could be called semantics-based strategy [1]. It consists of converting the legacy database into new, normalized and data structures that ensure semantic equivalence only. For example, a record type can be translated into several normalized tables, while several record types could be merged into a single table. The new database is strictly independent from the legacy technology and no longer suffers from the flaws and idiosyncrasies of the legacy database. Unfortunately, this strategy can be much more costly than the former one. It requires a deep understanding of the legacy data structures before translating them into the target data model. Secondly, since both data structures generally are quite different, data conversion involves complex data transformations that go well beyond the straightforward store-each-record-into-a-row technique of the former approach. Thirdly, the conversion of the programs is more complex since a legacy I/O statement could have to be developed into a complex procedure the writing of which may require an in depth

Understanding of the application logic. The goal of this thesis is to explore and develop solutions to the problems raised by the n-tire application architecture approach. Current trend is faster and globally sometimes faster and everywhere. It induces that services that were available locally or indirectly only should be available permanently online by web interface or as web service also for automatic use. The development speed is very significant – who is too slow cannot be successful just like on the market too late to get interesting share, except investing large amount of money into very strong advertisement or illegal pushing of the product and therefore it is no time to redevelop the service. It is a strong reason why to use legacy systems. There is also other reason why to use a legacy system there can be some know-how hidden in legacy system LS that can be lost if LS must be redeveloped.

The idea to use legacy system has more advantages that are visible at the first sight. The legacy system is usually proven that it matches some area of user needs and does there what user wants and expects. There are no costs related to the requirement specification – it is only needed to state whose services will be made available through a new interface of a legacy system. The new interface of given legacy system should be specified in a problem-oriented or user-oriented way – it should hide the implementation details of the legacy system. Such interface is usually very stable as the problems that people have to solve tend to be stable for longer time than the methods used to solve it. The legacy systems are, however, usually not written to enable future addition of new interfaces. Even if the source code of an application A is available and it is not always the case, it can be quite difficult to provide a new gate of A. The source code can be lost or was not delivered at all especially if the application has been developed by some other company. If want to transform a system having no available source code into a web service, must apply special technical turns discussed below. Legacy systems must often keep functions and interfaces specified in their original requirements specification, their local users should use them as before – it is the users should not to mention that their application has been converted into a web service.
Generally a service in the sense of web services [20]. Attempt to give an overview of the techniques for the development of new legacy systems interfaces and the practical experience with some of them. It is also discuss some techniques of overcoming some obstacles of the

development of new interfaces of the legacy systems having no available or maintainable source code.

The migration strategy describe in this thesis is sketched in Figure.Figure.1.1.The left part shows the main parts of the legacy system, comprising programs that interact with the legacy data through its legacy schema.

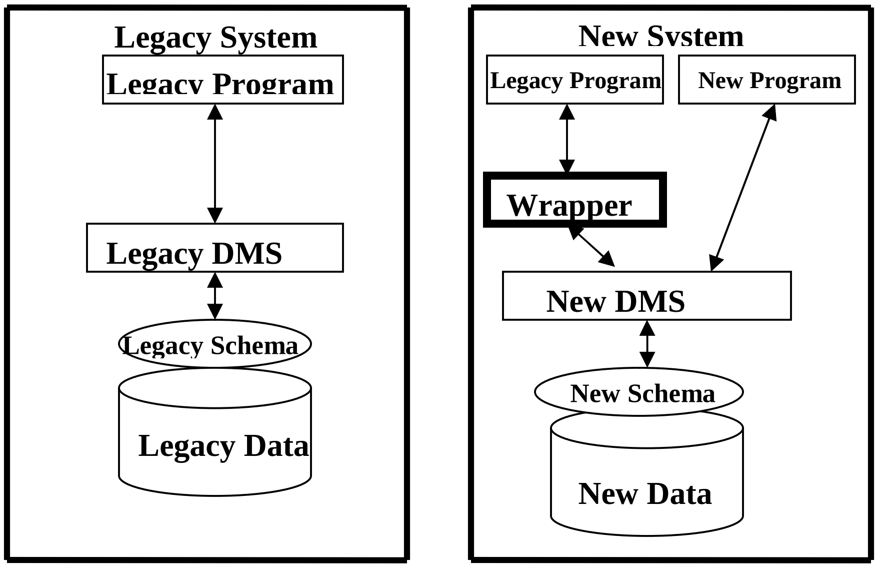


Figure 1.1 System conversion

1.2 Updating legacy data through wrappers

Data models in general and legacy models in particular, cannot express all the structures and properties of the real world. Limitations of the modeling concepts and information hiding programming practices lead to the incompleteness of the database schema that only contains the constructs structures and constraints explicitly expressed in the DDL code, while the other constructs are implicitly implemented in the program codes. For examples, while foreign keys can be explicitly declared in modern relational schemas, they are implicit hidden in FOXPRO file structures and in old relational schemas. This leads to distinguishing the physical schema which includes explicit constructs only and the logical schema which includes the implicit constructs, for instance a logical schema of FOXPRO files generally includes foreign keys, while the latter cannot appear in a FOXPRO file physical schema. Since these foreign keys must be recovered through in-depth system analysis, wrapper development and database reverse engineering, one of the goals of which is to elicit hidden structures and constraints are closely linked. More precisely, database reverse engineering will be the first step of the wrapper development methodology.

The legacy DBMS only manages the explicit constraints while the implicit constraints are implemented by validation code section scattered throughout the local application programs. In order to preserve the legacy data consistency in new distributed contexts both types of constraints must be considered. Therefore, it is the responsibility of the wrapper to guarantee legacy data consistency by rejecting updates that violate implicit constraints. In this way, both legacy applications and new applications that update the data through the wrapper can coexist without threatening data integrity (Figure 1.2).

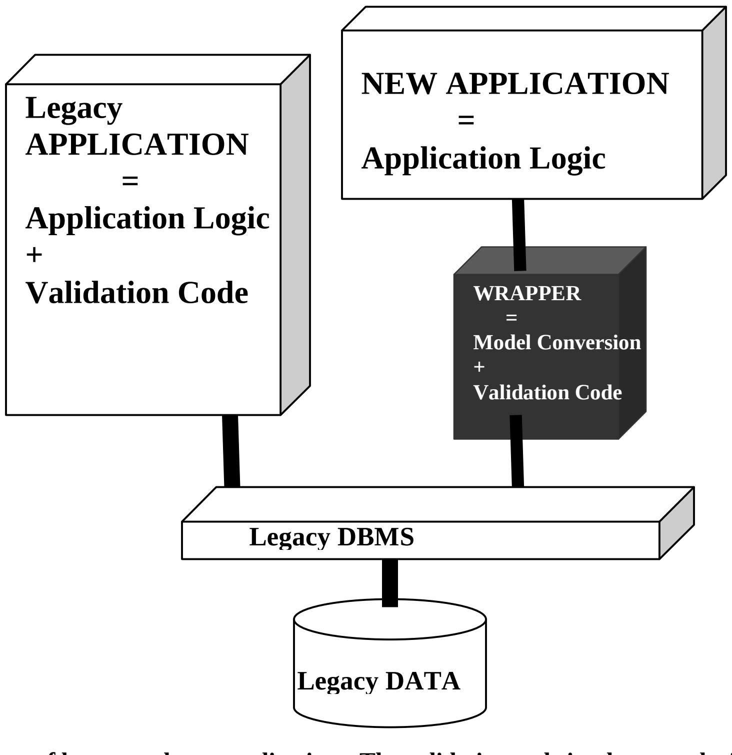

Legacy APPLICATION = Application Logic + Validation Code
NEW APPLICATION = Application Logic
WRAPPER = Model Conversion + Validation Code
Legacy DBMS
Legacy DATA

Figure 1.2 Coexistence of legacy and new applications. The validation code implements the implicit constraints.

1.3 Current approaches

Several prototype wrapper systems for legacy databases have been developed. They share two common characteristics against which can compare our approach ,namely the level of transparency and the update facilities they provide:

- **Wrapper transparency.** Several researchers ([9], [10], [1], [3], [4] and [11]) consider a wrapper as a pure model converter i.e. a software component that translates data and queries from the legacy DBMS model to another, more abstract and DBMS independent model. That is, the wrapper is only used to overcome the data model and query heterogeneity in database systems, leaving aside the added value that can be expected from wrapping. In such approaches, the semantic contents of both database and wrapper schemas are identical the wrapper schema just translates explicit constructs and ignores implicit ones.

- **Update facilities.** From the application point of view, it is natural that an update request expressed on the wrapper schema be automatically mapped onto the underlying legacy database. However, very few of the current wrappers supply this kind of support [3] or [4]. Wrappers that support updates generally do not consider implicit constraints they only provide update query mappings without any data consistency verification.

This work had to be extending on wrappers for legacy databases by exploring the updating function of such wrappers [14]. Consider wrappers that export a wrapper schema augmented with integrity constraints and structures that are not defined in the database schema. Updating data through such a wrapper poses the problem of guaranteeing legacy data consistency by rejecting updates that violate constraints be they implicit or explicit. This thesis describes a general architecture that addresses the problem of providing users and programmers with a wrapper able to emulate implicit structures and constraints. It also states that the main steps of a methodology for developing these wrappers and describes briefly a CASE tool that supports it. Both are based on the transformational paradigm that provides a rigorous framework for automatically translating queries and to automatically generating the data consistency management functions.

1.4 Objective of Study

The research will be mainly concentrated to develop Wrapper Database Schema and how to access the wrapper schema in application development .In general the research will have following objective:

i. To develop a wrapper Schema of legacy database that allows programmers with a wrapper. That able to catch the implicit structure and constraints in new application through the concept of N-tire database Architecture.
ii. To address the problem of guaranteeing legacy data consistency during development of new application.
iii. Develop a toolkit-A programming technique that applies the concept of wrapper schema of legacy database during Technology change.

1.5 Significance and Limitations of the Study

This study focuses on the updating legacy Database through wrapping Technology- A programming Technique. This also introduces the basic concepts associated with the wrapping technology and its architecture. Every research work has to face some limitations. This study was completed within a confined time with limited resources. However, the researcher had tried to make every possibility to carry out the study works more accurately as far as possible rather than just being perfunctory.

1.6 Thesis Structure

Chapter 1 “Introduction” this section introduced some idea about the wrapping Technology, Legacy database and motivation behind this. This section also presented some aspects of this study.

Chapter 2 “Schema Reengineering” includes an introduction to the Schema Reengineering, a discussion on the distinguishing features of the reverse engineering process, schema definition, tool support, Schema Reengineering Process and describes different components and schema architecture including Component transformation and data Conversion.

Chapter 3 “Wrapper Schema” describes different mapping Technique, Generic Transformation framework of schema, Inverse Transformation, schema transformation framework for specifying query and update mappings as well as implicit constraints and how wrapper schema derivation takes place in database management system. This chapter also cover small case study that allows us to identify some of the main problems to be solved.

Chapter 4 “Wrapper Architecture” describes the major wrapper architecture and the problem of wrapper query/update, implicit constraint, and update problem, main aspects of the architecture of wrappers that guarantee legacy data consistency with respect to both explicit and implicit constraints and phantom problem. These problems are generally arises in database management system due to the lack of proper programming technique (N-tire). Moreover, this chapter describes architecture of wrapper and it’s utilization in N-tire programming concept.

Chapter 5 “Wrapper Technology in microsoft.Net” describes N-tire programming technique and its layer with high level of implicit constraint management using wrapping technology .Theory in detail, describes presentation, data access and business logic layer. DAL (wrapper) is a major correctness criterion of implicit constraint control problems.

Chapter 6 “Wrapper Development Algorithm” describes the wrapper develop algorithm with programming technique using wrapper technology. This chapter also deals with developing wrappers for legacy databases in a semi-automated way.

Chapter 7 “Implementation and Testing” has described a complete organization of the program to implement the model presented in the sixth chapter. Sample testing and output analysis has been presented in this chapter.

Chapter 8 “Conclusion and Further Recommendations” has presented the concluding remarks of this study and future work on the legacy database updating using wrapper technology.

1.7 Problem Definition

A community of database researchers has proposed, implemented and measured a variety legacy database updating through code level programming. While these directly updating of legacy database doesn't consider the problem of guaranteeing data consistency by rejecting updates that violates constraints implicitly or explicitly.

My studies propose a different strategy how to keep original database unchanged but to build a wrapper that will allow new and external clients to retrieve and updates the data. Specially, in web application how to use the legacy database as N-tire (i.e. the Database server, web server, business logic layer, data access layer and presentation layer) database architecture with a design aspect of n-tire application architecture- A programming technique. The central approach in my thesis is to address the problem of providing users and programmers with a wrapper able to emulate implicit structure and constrains in new application through database wrapper schema rather than through application code.

1.8 Literature Survey

The literature survey is essential for keeping speed with the development of related fields. Several related works have been done in the related areas. So, several related books, papers and information are collected from respected supervisor and Internet and have been studied sincerely which helps us to complete my research work.

Chapter 2

Schema Reengineering

2.1 Introduction

The schema reengineering or conversion process analyzes the legacy applications to extract the logical and conceptual schemas of the legacy database through a database reverse engineering phase. Then this conceptual schema is transformed into the logical schema of the new database through a classical database design process and then is coded into the DDL of the new DMS.

2.2 Methodology

A complete presentation of this methodology can be found in [20]. The DDL analysis parses the legacy DDL code to retrieve the raw physical schema. In the schema refinement process, the schema is refined through an in-depth inspection of the way the program uses and manages the data. Through this process, additional structures and constraints are identified, which were not explicitly declared but expressed in the procedural code.

The existing data can also be analyzed either to detect constraints or to confirm or discard hypotheses on the existence of such constraints. The final DBRE step is the data structure conceptualization interpreting the legacy logical schema into the conceptual schema. Both schemas have the same semantics but the latter is platform independent and includes no technical constructs.

The logical schema of the new database is derived from the conceptual schema through standard database engineering techniques [24]. This schema is then enriched with technical constructs specific to the new platform and is then used to generate the DDL code of the new database.

2.3 Tool Support

Extracting the raw physical schema and storing it in the CASE tool repository is done through a DDL extractor (SQL, FOXPRO, CODASYL and RPG) from the parser library. Schema refinement requires powerful and customizable schema and program analyzers such as program slicing and pattern matching. Experience showed us that there is no such thing as two similar reengineering projects [27]. Hence the need for programmable extensible and customizable tools. DB-MAIN, it is more specifically its Meta functions includes features to extend its repository and its functions. It includes a 4GL that allows analysts to develop their own customized processors [20].

Data structure conceptualization and database design are based on schema transformations that will be discussed below. Code generators produce the DDL code of the new database according to the specifications of logical schema.

2.4 Mapping Method

Deriving a schema from another is performed through techniques such as renaming, translating, conceptualizing, which basically are schema transformations. Most database engineering processes can be formalized as chains of schema transformations as demonstrated in [21].

2.5 Schema Transformation

A schema transformation consists of deriving a target schema S' from a source schema S by replacing construct C possibly empty in S with a new construct C' it is also possibly empty. Adding an attribute to an entity type and replacing a relationship type with an equivalent entity type or with a foreign key are three examples of schema transformations.

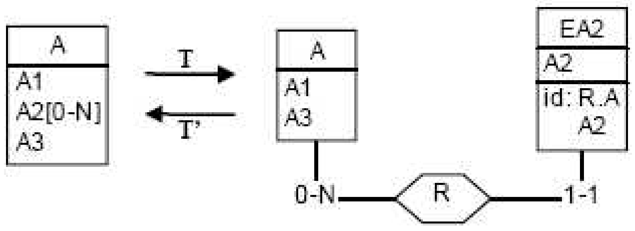


Figure 2.1 Representation of the structural mapping of a transformation that replaces multivalued attribute A2 with entity type EA2 and relationship type R[2]

More formally, a transformation is defined as a couple of mappings <T,t> such as: C' = T(C) and c' = t(c), where c is any instance of C and c' the corresponding instance of C'.
Structural mapping T explains how to modify the schema while instance mapping t states how to compute the instance set of C' from the instances of C. Any transformation can be given an inverse transformation T' = <T',t'> such that T'(T(C))=C. If this also have t'(t(c)) =c, then and T' are said semantics preserving.

2.6 Compound Transformation

A compound transformation T = T2 to T1 is obtained by applying T2 on the schema that results from the application of T1 [1]. An important conclusion of the transformation-based analysis of database engineering processes is that most of them including reverse engineering and database design can be modeled through semantics-preserving transformations. Transforming a conceptual schema CS into a logical schema LS can be modeled as a compound semantics-preserving transformation C-to-L = <CS-to-LS, cd-to-ld> in such a way that LS = CS-to-LS (CS). This transformation has an inverse: L-to-C = <LS-to-CS, ld-to-cd> such as CS = L-to-C (LS).

2.7 Source and Target Logical Mappings

The mappings between the source and target logical schemas are modeled through a transformation history. The history is defined by the trace of the complex compound transformation LegLS-to-NewLS = LS-to-CS to CS-to-LS in such a way that NewLS =CS-to-LS(LS-to-CS(LegLS)), where NewLS is the new logical schema and LegLS is the legacy logical schema.

2.8 Support

DB-MAIN includes a rich toolkit of transformations, most of them being semantics preserving. In addition, it can record the history of the transformations applied to convert the legacy logical schema into a conceptual schema and to transform the latter into the new logical schema.

2.9 Data Conversion

Schema conversion is concerned with the conversion of the data format and not of its content [1]. Data conversion is taken in charge by a software component often called ETL processor, which transforms the data from the data source to the format defined by the target schema. A converter has three main functions. Firstly, it performs the extraction of the data source. Then, it converts these data in such a way that their structures match the target new format. Finally, it writes legacy data in the target format. A converter relies on the mappings between the source and target physical schemas. The analysis of the Schema reengineering process has shown that, through schema refinement, implicit constraints can be discovered, that will be translated into the new schema. Quite often, data may violate these constraints, so that the legacy data often have to be cleaned before being migrated. Though data cleaning can be a complex task, it can be partly automated through specific functions of the migratory[26].

2.9.1 Methodology

Data conversion involves three main tasks. Firstly, the legacy logical schema is converted into the new logical schema. Secondly, the mapping between the source and target physical schemas is extracted. Finally, this mapping is implemented in the migrator for translating the legacy data according to the format defined in new logical schema.

2.9.2 Tool Support

Writing data migrators manually is an expensive and complex task particularly for mapping them in semantics-based migration. DBMAIN [23] offers the mechanisms to record transformation histories. It also provides a translator of histories into mapping between the legacy and new schemas. Several migratory generators have been developed for various technology conversions. Such a generator automatically derives the migrator code from the mapping between both logical schemas.

It must be noted that full automation generally is not achieved for cost reason. Some mappings are problem-specific and have not been coded in the generator. In such cases, the analyst has to add the specific code in the migrator. For example, if some specific data conversion is needed convert measuring units, capitalize a string and add a prefix to a field value the user must write the conversion code and insert it in the migrator.

In order to design a more general solution and to minimize hand-written migratory generator these works have divided the data migratory in two modules. The first one reads the source data and produces an XML file. The second reads the XML file and stores the data into the target database. The structure of the XML file is intermediate between the source schema and the target schema, in order to ease the migration. The only purpose of the XML document is to facilitate the transfer of data. Thus this thesis did not try to have an XML file that express the semantics of the data, as if it had to be used to store and manipulate the data.

For example, if the data contains information about customers and orders at that time this is not necessarily produce a file where the orders are encapsulated into the data of its customer. Usually the structure of the XML file is very close to the structure of the legacy schema.

Chapter 3

Wrapper Schema

3.1 Wrapper schema definition

The database physical schema of the legacy database (Figure 3.1).The physical schema comprises the tables Customer and Order. Optional it may be nullable columns are represented by the [0-1] cardinality constraint and indexes by the access key constructs. This process is fairly straightforward since it is based on the analysis of declarative code fragments or data dictionary contents. It recovers explicit constraints only ignoring all the implicit constraints that may be buried into the procedural code of the programs. Hence the need for a refinement process that cleans the physical schema and enriches it with implicit constraints providing the logical schema. Their elicitation is carried out through reverse engineering techniques such as program analysis and data analysis [5].

1) **Program analysis**: Before inserting a new order the local applications check whether the customer number is already recorded in the database as an implicit foreign key.

2) **Data analysis**: If Reference is a primary key of Customer, then its values must be unique, a property that will be checked through a query such as:

```
Select * from Customer
Group by Reference having count (Reference)>1
```

Those to be obtain the logical schema shown in Figure 3.1. New constraints now appear such as primary keys (id constructs), foreign keys (ref constructs) and redundancies expressed by a functional dependency FD: Customer $\rightarrow$ Account.

The next phase consists in interpreting and exporting the logical schema therefore providing the wrapper schema through which the new client applications will view the legacy data. The logical schema expressed in an abstract data model must be expressed in the operational data model of the wrapper. This schema still includes undesirable constructs such as redundancies and other idiosyncrasies that must be managed but also hidden from the client applications and therefore discarded from the wrapper schema. The logical schema of Figure 3.1 includes a property stating that Address is often split into three pertinent fragments.

It depicts a structural redundancy the attribute Account of Order is a copy of the attribute Account of Customer. To hide this redundancy the attribute Account of Order does not appear anymore in the wrapper schema.

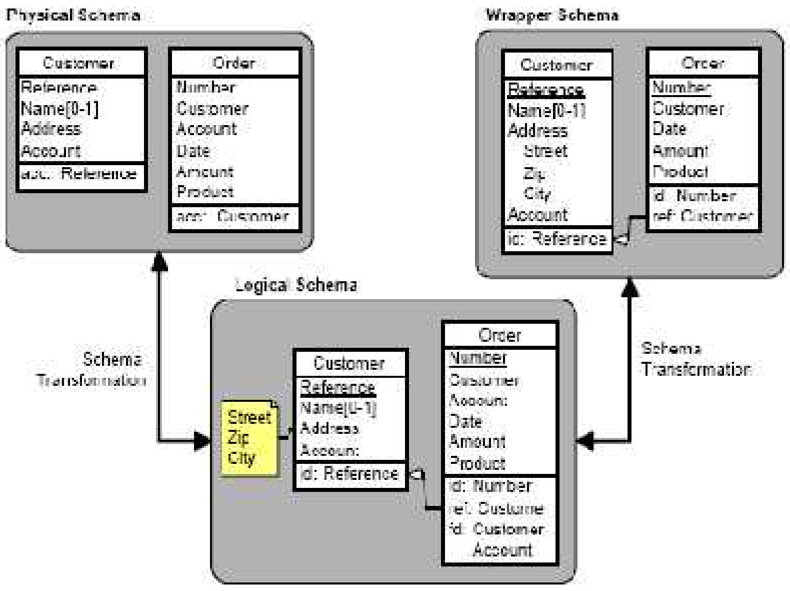


Figure 3.1 the physical database, logical and wrapper schemas[2].

3.2. Mapping definition

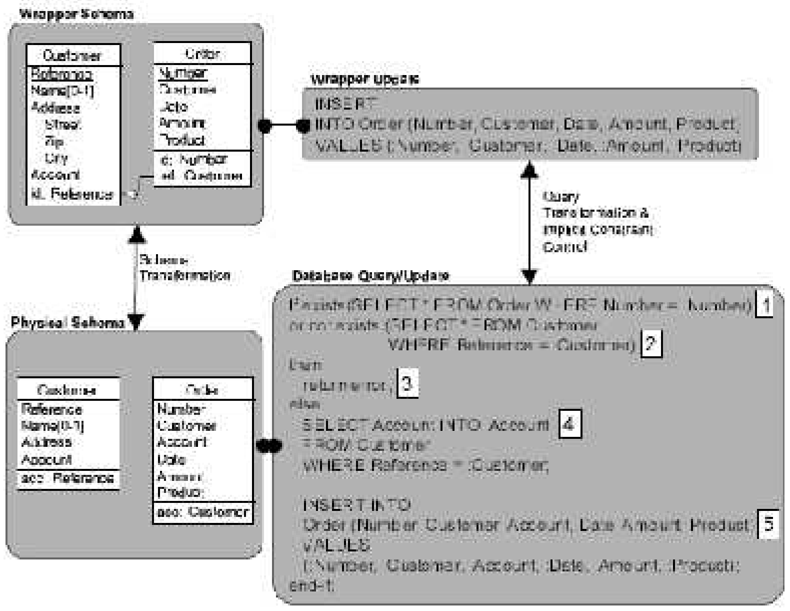
A transformation consists in deriving a target schema S′ from a source schema S by replacing construct C possibly empty in S with a new construct C′ possibly empty. Considering instance c of C and instance c′ of C′, a transformation ∑ can be completely defined by a pair of mappings <T, t> such that C′ = T(C) and c′ = t(c). T is the structural mapping that explains how to replace construct C with construct C′ while t, the instance mapping states how to compute instance c′ of C′ from any instance c of C [2].

Once the wrapper schema has been built that have to state how wrapper retrieval and update queries can be mapped onto legacy data. In the schema hierarchy mentioned above the transitions between physical and wrapper schemas can be expressed as formal transformations on structures such as discarding, renaming or aggregating and on constraints such as adding primary keys, foreign keys and functional dependency. The complexity of the transformation depends on the distance between the logical and wrapper schemas. For instance, an XML-based wrapper schema would require more sophisticated mapping rules than those mentioned above [13].

The database wrapper mappings can then be built by interpreting the transformations on structures as two-way data conversion functions whereas the implicit constraint management can be emulated by transformations on constraints. Let us assume the wrapper update shown in Figure 3.2. By analyzing the update statement the wrapper dynamically generates a sequence of operations that emulate and manage the implicit structures and constraints. Clearly, the main operations carried out by the wrapper are the following:

- **Implicit constraint management:** The wrapper checks the satisfaction of the constraints implied by implicit identifier 1 and the implicit foreign key 2.
- **Data error management:** The wrapper reports possible implicit constraint violation 3.
- **Redundancy management:** The wrapper controls the redundant attributes by assigning the value it gets from the source data 4.

) **Query translation:** Translation of the wrapper update against the wrapper schema into updates on the physical database schema 5.



```
INSERT
INTO Order (Number, Customer, Date, Amount, Product)
VALUES (:Number, :Customer, :Date, :Amount, :Product)
```

```
if exists(SELECT * FROM Order WHERE Number = :Number)        1
or not exists (SELECT * FROM Customer
               WHERE Reference = :Customer)                  2
then
  return error;                                               3
else
  SELECT Account INTO :Account                                4
  FROM Customer
  WHERE Reference = :Customer;

  INSERT INTO
  Order (Number, Customer, Account, Date, Amount, Product)    5
  VALUES
  (:Number, :Customer, :Account, :Date, :Amount, :Product);
end-if;
```

Figure 3.2 Example of update translation and implicit constraint management[1].

3.3. Generic transformational framework

Query translation is a process that relies on mappings between schemas that are built within different paradigms. Database and wrapper schemas are expressed in a unique wide spectrum specification model so-called Generic Entity-Relationship model (GER), from which the operational data models can be derived by specialization that is, by selecting a subset of concepts and by defining restrictive assembly rules. As a result, it provides an ideal support for our query translation approach based on schema transformations. Any transformation can be used whatever their underlying data model. The same schema transformation can be used in a relational schema and in an ER schema [14].

3.3.1. Inverse transformation.

Each Transformation $\sum_1 = <T_1, t1>$ can be given an inverse transformation $\sum_2 = <T2, t2>$, usually denoted $\sum^{-1}$ such that, for any structure C, T2 (T1(C)) = C. $\sum_2$ being the inverse of $\sum_1$ does not imply that $\sum_1$ is the inverse of $\sum_2$. $\sum_2$ is not necessarily reversible. These properties can be guaranteed only for a special variety of transformations called symmetrically reversible [2].

$\sum_1$ is said to be a symmetrically reversible transformation or more simply semantics-preserving if it is reversible and if its inverse is reversible too. From now on, unless mentioned otherwise, that would be worked on the structural part of transformations, so that that will denote a transformation through its T part.

3.3.2 Structural analysis of schema transformations.

A transformation is known to replace construct C with construct C′ in schema S, The new schema S′. The effect of a transformation T in schema S can be specified as follows. Define a schema S as a set of constructs. Therefore, set-theoretic relations and operators apply on schemas. Let us consider the structural functions C_-, C_+ and C_0.

- C_- returns the constructs of S that have disappeared in S′.
- C_+ returns the new constructs that appear in S′.
- C_0 returns the constructs of S that are concerned by T, but that are preserved by transformation the catalytic constructs of T.

3.3.3. Transformation Sequence.

A transformation sequences a list of n primitive transformations S1-to-S2 = (T1 T2 ... Tn). The application of S1-to-S2 = (T1 T2) on a schema S1 consists of the application of T2 on the schema that results from the application of T1 so that would be obtained S2. As for schema transformation, a transformation can be inverted. The inverse sequence S2-to-S1 can be derived from the sequence S1-to-S1 and can be defined as follows:

If S1-to-S2 = (T1 T2 ... Tn) then S2-to-S1 = ($Tn-^{1}$ ... T_2-^{1} T_1-^{1}) where T_i-^{1} is the inverse of T_i and hence S1 = S2-to-S1 (S2).S2-to-S1 is obtained by replacing each origin schema transformation by its inverse and by reversing the operation order. The concepts of sequence and its inverse are used for defining the mappings between two schemas. The transformational approach then consists in defining a reversible transformation sequence which applied to the source schema and produces the target schema.

3.4. Transformational approach of query mapping

Query translation is the core function of a wrapper. It refers to operations that translate queries between two schemas the database and wrapper schemas and two languages the database and wrapper query languages. Considering the issue of translating queries from one language to another one, the idea is to use an intermediate level, independent of all the possible operational query languages. That has been used by an internal abstract query language as the bridge for the translation rather than directly translating wrapper queries into database queries. Considering the issue of schema mapping this is to use schema transformations that provide mechanisms for formally defining the schema correspondence between the database and wrapper schemas and then on using that equivalence to automatically perform the query mappings.

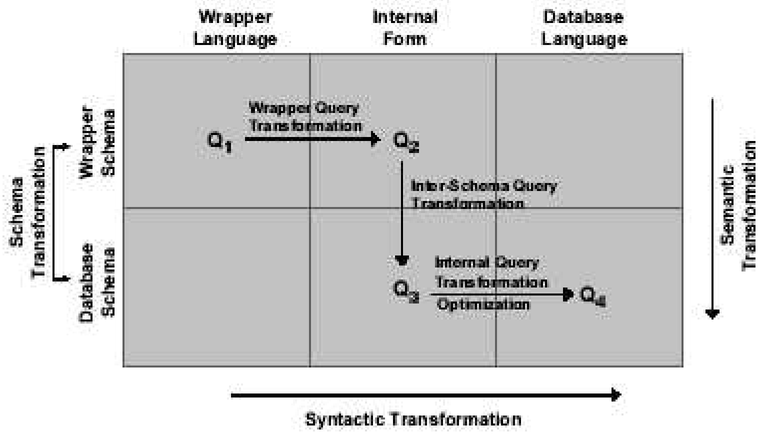


Figure 3.3 Language and schema mappings of a wrapper query Q1 into a DML query Q4[1].

Figure 3.3 shows the translation process. The wrapper query Q1 is first stripped of creating an internal form Q2 that captures purely the semantics of the Query. Next Q3 is derived by application of the schema transformations on the constructs of Q2. Finally, Q3 is translated into a query Q4 that complies with both the database chema and the database DML. That could be now state the three main successive steps of query translation.

- **Language mappings:** Syntactic translation of the wrapper query into an internal form.

- **Inter-schema mappings:** Semantics translation of the query using the schema transformation approach for defining the mappings between the database and wrapper schemas.

- **Language mappings and optimization:** Syntactic translation of an internal form into a query based on the DBMS query language. Producing an efficient execution strategy depends on the syntax and expressiveness of both the wrapper or internal and DBMS query processing capabilities.

Dealing with such issues is out of the scope of this thesis however. This thesis present some strategies for implementing query processors and optimizers in wrappers dedicated to Oracle database systems. That has been described the formal framework of reversible transformations based on a generic data model and internal query language based on the same data model. As will see reversible transformations allow internal queries to be automatically translated in either direction between two Schemas.

3.5. Generic Entity-Relationship Model

For the need of this thesis, the GER can be perceived as an enriched variant of the standard entity relationship model. It includes the concepts of entity type, attribute and value domain and relationship type. Attributes can be atomic or compound, mandatory or optional, single-valued or multivalued. The roles of a relationship type can be labeled it has a cardinality constraint a pair of integers stating the range of the number of relationships in which any entity can appear. An attribute has a cardinality constraint too that states how many values can be associated with each parent instance default is 1-1 and does not appear in graphical schemas. Several properties hold and must be declared among the components of an entity type uniqueness, referential and existence constraints are just some of them. Due the wide variety of such properties the GER includes the generic concept of property group or group for short. A group is any subset of components attributes and roles of an entity type on which one or several properties are defined. The label of the group specifies its properties (id for identifier, ref for referential, excl for exclusive, and so on). For example, a group of attributes of entity type E can be declared identifier and referential.

This group models such relational pattern as a primary key that simultaneously is a foreign key. This generic data model can be specialized into any operational model. A specialized model is built by selecting generic constructs and structural constraints and by renaming constructs to make them comply with the concept taxonomy of the specialized model. As an illustration, the relational model considered as an operational database model can be precisely defined as:

- **Selecting constructs.** Select the following constructs: entity types, domains, attributes, identifiers and reference attributes.

- **Structural constraints.** An entity type has at least one attribute. The valid attribute cardinalities are [0-1] and [1-1]. An attribute must be atomic.

- **Renaming constructs.** An entity type is called a table, an attribute is called a column, an identifier, a key and a group of reference attributes a foreign key.

3.6. Schema and query mapping

A schema transformation sequence can be used to automatically translate queries between a pair of schemas. More precisely, for a schema transformation sequence between two schemas S1 and S2, show how this sequence can be used to automatically translate queries posed on S2 to queries posed on S1.

3.7. Model and query language

A binary model defined as a sub-model of the generic data model described above. This model is compliant with standard files, SQL and ER models. It is expressive and generic enough to describe all the main structures and constraints that are explicitly offered by these data models:

- Atomic or compound attributes like single-valued or multivalued attributes.

- Reference, identifier and access groups.

- Entity types with at least one attribute and one identifier.

- Binary, non cyclic relationship types and without attribute.

It was also provided a simple query language based on this binary model a query is a conjunction of schema constructs. A query answer is a set instance of schema constructs. Any query over a schema S is an expression whose variables are constructs of S. The syntax of a query is:

Query: = Construct | Predicate | [and, Query, Query | [or, Query, Query] | [not, Query]

Predicate: = [eq, Atom,] | [less, Atom, Atom]

Construct identifies a schema construct being added or deleted by a transformation. This is one of the constructs that take part in the definition of a schema transformation signature. The underscore character is an anonymous variable. Atom represents a variable declared in a schema construct. When eq refers two variables of the same query, it can simplify the query and omit this predicate.

3.8. Schema transformation and query substitution

Let us assume that a schema S1 is transformed into a schema S2 and the queries posed on S1 have to be translated to queries posed on S2. Consider first the case where S1 is transformed into S2 by a single primitive transformation T. The only cases that need to consider in order to translating a query Q1 posed on S1 to an equivalent query Q2 on S2 are to apply renaming and to substitute occurrences of constructs of C (T) .For transformation sequences the substitutions are successively applied in order to obtain the final query Q2.

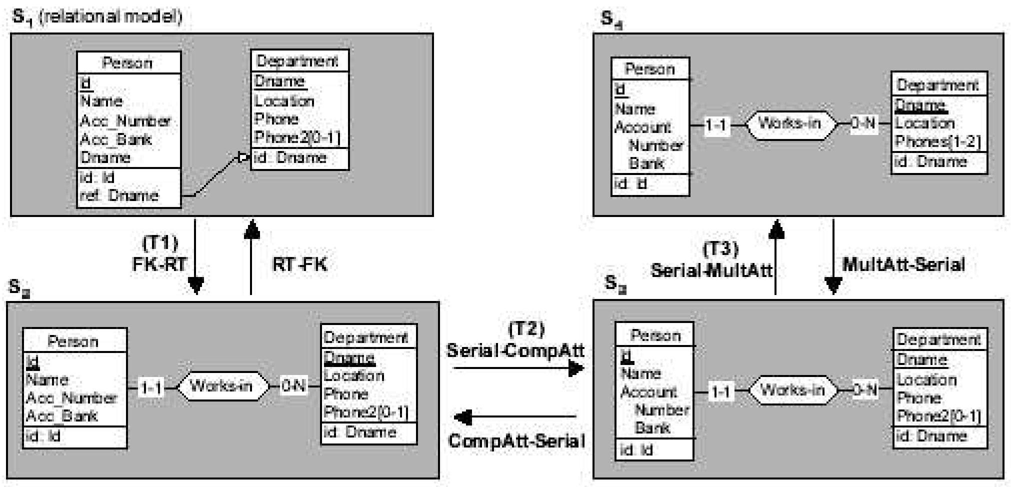


Figure 3.4 Sequence of schema transformations: a foreign key transformation followed by an aggregation transformation and a transformation of serial attributes into a multi-valued one [1].

3.9. Wrapper schema derivation

This process of semantic interpretation consists in exporting and interpreting the logical schema from which one tries to extract the wrapper schema WS and the schema transformation sequence LS-to-WS. Two main different problems have to be solved through specific techniques and reasoning's.

- **Model translation:** The logical schema expressed in the GER must be expressed in the operational data model of the wrapper. This process can be fairly straightforward if the logical and wrapper models are similar like DB-to-ODBC but it can be quite complex if they are different such as Oracle-to-XML or FOXPRO-to-relational. This model translation basically is a schema transformation. It consists in translating a schema expressed in a source data model M_s into a schema expressed in a target data model M_t where M_s and M_t are defined as two sub-models i.e. subsets of the generic data model. Model transformation is defined as a model-driven transformation within the GER. A model-driven transformation consists in applying the relevant transformations on the relevant constructs of the schema expressed in M_s in such a way that the final result complies with M_t [6].

- **De-optimization:** Most developers introduce consciously or not optimization constructs and transformations in their physical schemas. These practices can be classified in three families, namely structural redundancies, non normalization merging data units linked through many-to-one relations and restructuring such as splitting and merging tables. The de-optimization process consists in identifying such patterns and discarding them either by removing or by transforming them [28].

Chapter 4

Wrapper Architecture

4.1 Motivation

A data wrapper is a data model conversion component that is called by the application program to carry out operations on the database. Its goal is generally to provide application programs with a modern interface to a legacy database allowing Java programs to access FOXPRO files. In the present context, the wrapper is to transform the renovated data into the legacy format e.g. to allow FOXPRO programs to read, write records that are built from rows extracted from a relational database.

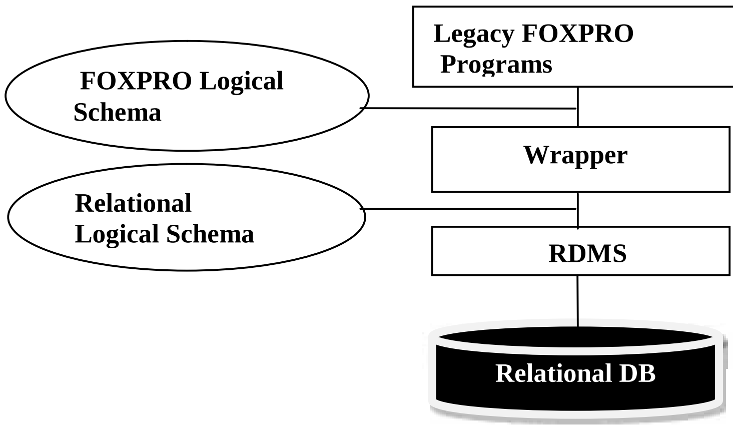


Figure.4.1 a wrapper allows the new database to be accessed by the legacy application.

The wrapper converts all legacy DMS requests from legacy applications into requests against the new DMS that now manages the data. Conversely, it captures results from the new DMS possibly converts them to the appropriate legacy format [22] and delivers them to the application program. Figure 4.1 depicts a frequent pattern in which the legacy logical schema relying on the FOXPRO file interface is rebuilt from the new relational logical schema.

Generic architecture for wrappers that provides both extract and update facilities and that control the implicit constructs of the source databases. This leads these wrappers to emulate advanced services such as integrity control and transaction and failure management if the underlying DBMS does not support them.

The functionalities of such a wrapper are classified into functional services (Figure 4.2) among which have been mentioned those which are relevant to the update aspects.

- **Wrapper query/update analysis**. The wrapper query is first analyzed so that incorrect updates are detected and rejected as early as possible.
- **Error reporting**. Wrappers report errors back to the client application.
- **Query/update and data translation.** This refers to operations that convert data and queries/updates from one model to the other.
- **Semantic integrity control.** Wrappers emulate implicit integrity constraints.
- **Security.** Data security is another important function of a wrapper that protects data against unauthorized access through its interface. Data security includes two aspects: data protection and authorization control.

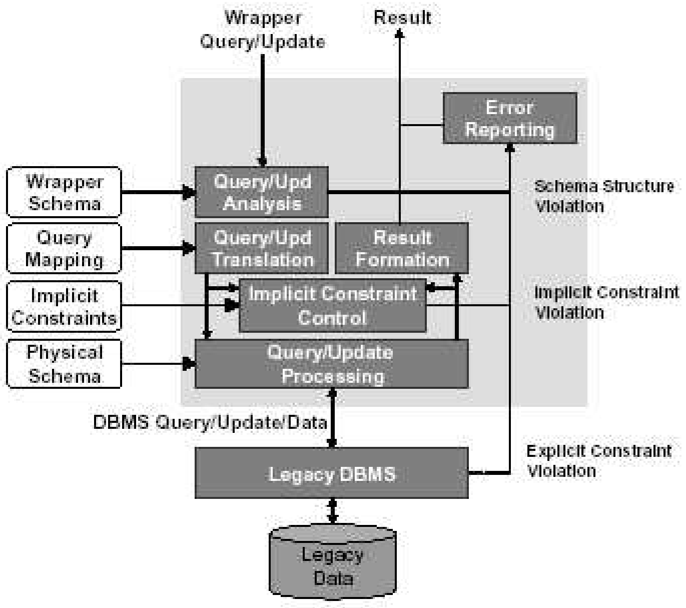


Figure 4.2 Wrapper architecture[2].

- **Concurrency and recovery control.** This function is to permit concurrent updates of the underlying legacy databases. This includes transaction and failure management. The latter two services will be ignored in which that only explain and discuss the first four classes of functionalities.

4.2. Wrapper query/update analysis

Query analysis enables rejection of queries for which further processing is either impossible or unnecessary. The main reasons for rejection are that the query is syntactically or semantically incorrect. When one of these cases is detected, the query is simply returned to the user with an explanation .Otherwise, query processing goes on. A query is incorrect if any of its attribute or entity type names are undefined in the wrapper schema or if operations are being applied to attributes of the wrong type. Concurrency, reliability and security are the major factor in query/update [8][12].

4.3. Error reporting

A wrapper returns a value that indicates the success or the failure of a wrapper query. An error can occur at two levels:

- **At the legacy DBMS level:** Legacy DBMS return some indicators on completion of query execution.

- **At the wrapper level:** The wrapper catches internal errors. For example, it detects query syntax errors or violation of implicit constraints [15].

Although DBMS return similar kinds of errors, each does it in a different way so that the wrapper must provide a unified error handling interface for both DBMS and proper wrapper conditions [2].

4.4. Query/update and data translation

Query translation is the core function of a wrapper. It refers to operations that translate queries between two schemas the database and wrapper schemas and two languages, the database and wrapper query languages.

It relies on the use of schema transformations for formally defining schema correspondence between the database and wrapper schemas. By replacing the schema constructs names in the wrapper query with their database equivalent that produce a database query that can be executed on the actual data.

4.5. Implicit constraint control

While the DBMS manages the explicit constraints defined in the physical schema the wrapper emulates the implicit constraints by rejecting updates that violate them. To prevent inconsistencies, pre-tests services are implemented in the wrapper.

This method is based on the production at wrapper development time of implicit constraint check components which are used subsequently to prevent the introduction of inconsistencies in the database.

An implicit constraint checking is defined by a triple <ET, T, and C> in which

- ET is an entity type of the physical schema.
- T is an update type (INSERT or DELETE).
- C is an implicit constraint assertion ranging over the entity type ET in an update of type T.

Example: The implicit primary key of Order is associated with the triple <Order, INSERT, C> where C, defined in SQL-like expression

EXISTS (SELECT * FROM Order WHERE Number =: Number).

Implicit constraint check assertions are obtained by applying transformation rules to the wrapper schema [15]. These rules are based on the update and implicit constraint types.

Assertion enforcement efficiency: Checking consistency assertions has a cost that depends on the physical constructs implemented in the legacy database. Checking uniqueness such as primary or candidate key or inclusion foreign key assertions can be very costly if such a construct is not supported by an index. While legacy applications easily cope with missing indexes, through sort or merge batch operations, new generally transactional, applications cannot afford relying on such outdated procedures. Besides complex architectures in which additional data structures are implemented to support efficient implicit constraint enforcement the order in which the assertions are evaluated often is relevant.

Chapter 5

Wrapper Technology in Microsoft .Net

5.1 Introduction

When Developing an Application various issues have to taken while designing the architecture of the application such as Performance, Scalability, Enhancements of the application, Security. So while deciding the architecture for an application that has to keep all the said issues in mind. That is the importance that needs to give for the Architecture of an application[18].

Firstly what is n-Tier architecture? It refers to the architecture of an application that has at least 3 "logical" layers or parts that are separate. Each layer interacts with only the layer directly below and has specific function that it is responsible for.

The main advantage of n-Tier architecture is each layer can be located on physically different servers with only minor code changes; hence they scale out and handle more server load. what each layer does internally is completely hidden to other layers and this makes it possible to change or update one layer without recompiling or modifying other layers. The another advantage of this architecture is if any change or add a layer, it can be done without redeploying the whole application For example, by separating data access code from the business logic code, when the database servers changes only needs to change the data access code. Because business logic code stays the same, the business logic code does not need to be modified or recompiled.

Generally N-Tire Application has 3 Layers they are:

1) **Presentation Layer**
2) **Business Logic Layer**
3) **Data Access Layer**

5.2 Presentation Layer

Presentation Layer is nothing but a piece of software that deals with User interface of the application. Displaying Data to the user and allowing him to interface with it is the main functionality. "Driving" that interface using business tier classes and objects. In ASP.NET it includes ASPX pages, user controls, server controls and sometimes security related classes and object [18].

There is possibility of creating Presentation layer by Asp.net Web Forms .Windows Forms are basically used to create traditional Desktop application. They can offer rich UI design. The main draw back of windows application is they need to install on each and every machine. On the other hand ASP.net web forms also offer rich UI. They are mainly used for web applications.

The **Presentation Layer** uses XML and XSLT based technology to present a graphical user interface to users, regardless of the device being used. This includes:

- **Themes —** customize the look, feel and layout of an application using pre-set theme packs. CSS is under this category.
- **Layout Templates —** change the look, feel and layout of any folder including the home page in the system using a layout manager
- **Application Interfaces** — pre-defined interfaces by application type
- **Page Editor** —Online editor for customizing the look and feel of specific community pages
- **Personal Area —** each application offers private member areas to store and organize private and personal information

5.3 Business Logic Layer

Business Logic Layer is responsible for processing the data retrieved and sent to the presentation layer. The main task of Business layer is business validation and business workflow

In ASP.NET it might be using the DataSet and DataReader objects to fill up a custom collection or process it to come up with a value, and then sending it to Presentation Layer. BLL sometimes works as just transparent layer.

For example, if a Dataset or DataReader is passed as an object directly to the presentation layer.Net Components forms these layers. Asp.net web service can also serve as Business Logic Layer. But they should be use only if the business validation happens at some external place other than our network .By using .Net Remoting can distribute the logic to other achiness also [18].

The **Business Logic Layer** determines access to various applications offered by the platform to users through a variety of standard devices including web browsers, desktop applications, email clients e.g.MS Outlook and mobile devices .This layer manages:

- **Authentication —** users must have a unique username and password authentication
- **Permissions —** every application has a pre-set of system wide permissions including none, read, write and full which can changed depending on the application and security requirements
- **Security —** all transactions can be encrypted using the SSL protocol for secure transmission

The **Business Logic Layer** also manages the interaction and access controls between a rich set of corporate social networking applications and tools:

- Control Panel
- Broadcasts
- Blogs

- Country Profiles
- Document Management
- Forums
- Notifications
- Photo Galleries
- Web Search
- Videos

5.4 Data Access Layer

Data Access Layer deals with data manipulation actions such as Insert, edit, and delete, select in .NET Database can be from SQL Server or Access database; however it's not limited to just those. It could also be Oracle, mySQL or even XML .Designed a data access layer in such a way that other layers need not have any knowledge about underlying data store.

ADO.NET is the data access technology under .NET. Though ADO.NET allows connected data access via DataReader classes more focus is on disconnected data access [18]. DataSet plays a key in this mode. Use ADO for data access but its use should have valid reasons. Do not use ADO just because just like Recordset Again .NET components form this layer. Web services can also form data access layer. This is especially true if database do not have a data provider. In such cases write some custom code to connect with the data and populate DataSet with it and then return DataSet to the caller. In addition to ADO.NET can also make use of built-in RDBMS capabilities such as stored procedures and functions. The Data Layer securely stores and manages all user and community data [21]. This data is only accessible by the Business Logic Layer. It enables the system to harvest, index, and catalog, publish and report on all network data. It has advantage like Reduces tight coupling between User interface, business process and database, Change in database or any data access methods do not have effect on the presentation Layer and It becomes easier to modify or extend application, without breaking or recompiling the client-side code.

Chapter 6

Wrapper Development Algorithm

The mapping between wrapper and database schemas is formally defined, can expect them to be a sound basis to build the wrapper in a systematic way. While the structural mapping T of a transformation defines a rewriting rule that can be used to transform the input query, its instance mapping t states how the instance of the target construct can be derived from that of the source construct. Therefore, these mappings can be used to define the query translation logic and the data transformation rules of the wrapper that implements this transformation. This analysis is still valid for transformation sequences so that complete wrappers can be formally specified by such sequences [18].

Each wrapper is developed as a program component dedicated to a specific database model and to a specific database. It comprises two parts, namely a model layer, in which the aspects specific to a given data model are coped with, and a database layer that is dedicated to the specific database schema. While the model layer is common to all the databases built in this model, the wrapper/database schemas mapping is hard coded rather than interpreted from mapping tables as it is the case in other approaches.

Algorithm 6.1 (Update Legacy Database through Wrapper**)**

```
declare-var
 msg : Message
Op : Operation
 x : DataItem
Ca: Code Access
ws :wrapper Schema
Bll :Busniess Logic
Dal: Data Access Layer
Er: Error
```

```
begin
      repeat
          WAIT(msg)
          case of msg
            WS:
            begin
                Case of Ca
            If(Da!=NULL) then
                    Check_Constraints ;
                    Begin_transaction:
            Use WS:
                        Update Legacy Database;
                    begin
                        Send Request to Server and Response by the client;
                   end
            update or delete data through Wrapper:
                else
                  Err;
               end-if
            end
    end-case
begin
  If  BLL=Change  or Op= failure  then
  begin
      for each BLL unit code  do
      begin
change the Business_Rule;
            if there are no more  Err on WS updation and
                there are operations to update then
            begin
            begin_Transaction;
            update_wrapper(X);
            end;
                  end-if
              end-if
             end
       until forever
   end. {Wrapper Schema Change}
```

Chapter 7

Implementation and Testing

7.1 Implementation

As the implementation part of this study, a simulation environment has been developed. This environment simulates the updating legacy database through Wrapping Technology. The program has been developed using of Microsoft .NET framework 2.0 with C# as a sever site script in web based application. There has been a legacy Database of KSK consisting Fox Pro Base Database.

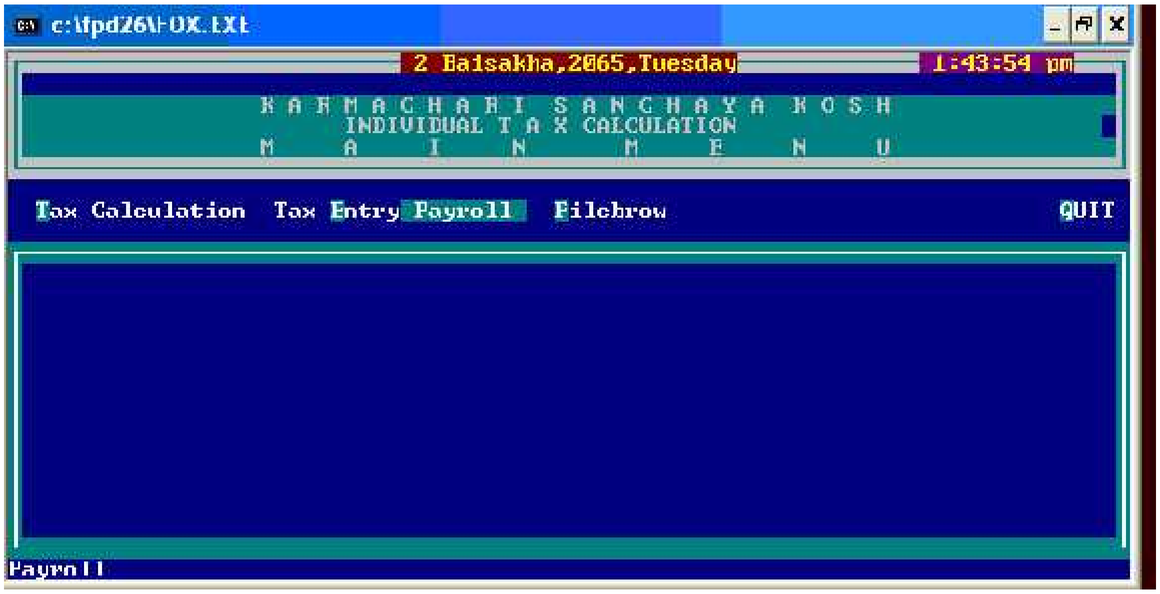
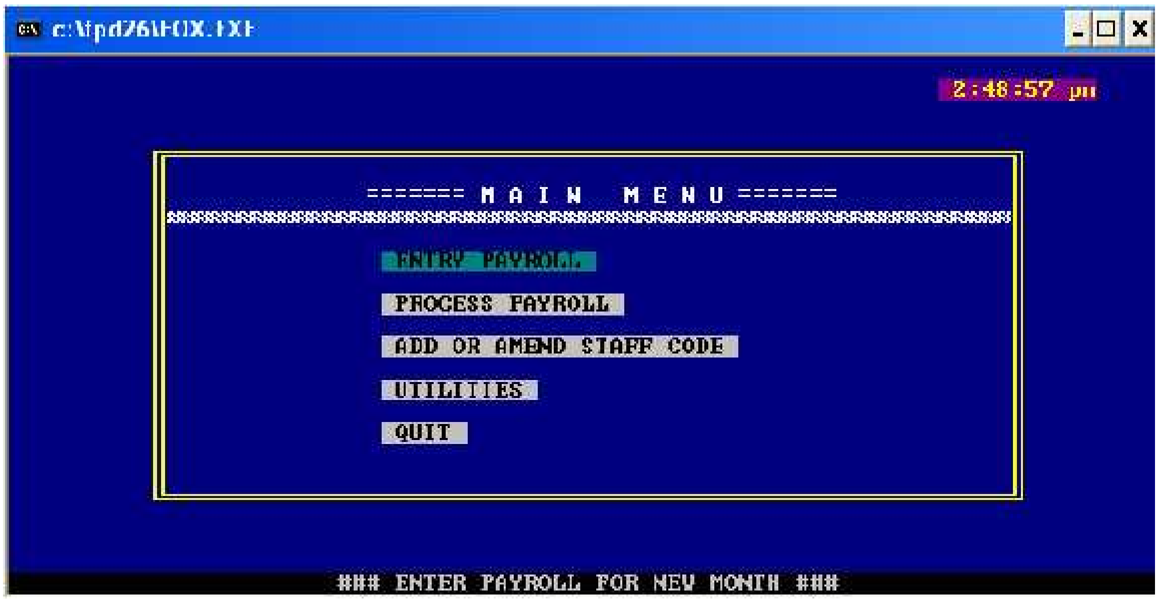


Figure 7.1 Main Form of Payroll System in Karmachari Sanchaya Kosh


```
c:\fpd26\FOX.EXE
                                                        2:48:57 pm

              ======= M A I N   M E N U =======

                    ENTRY PAYROLL
                    PROCESS PAYROLL
                    ADD OR AMEND STAFF CODE
                    UTILITIES
                    QUIT

             ### ENTER PAYROLL FOR NEW MONTH ###
```

Figure7.2 Payroll System

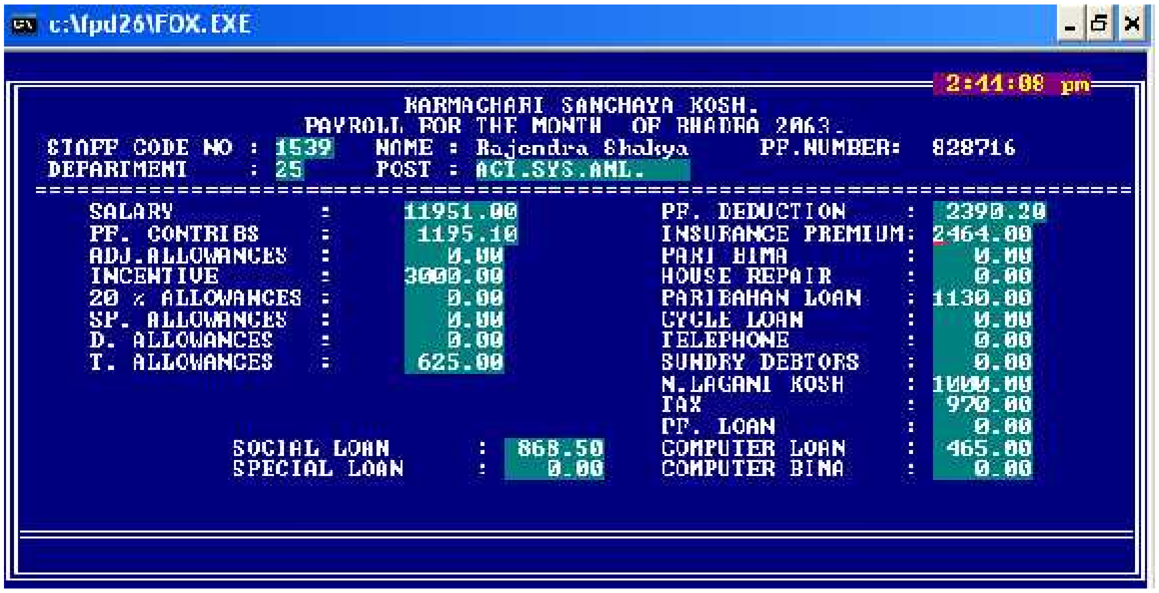
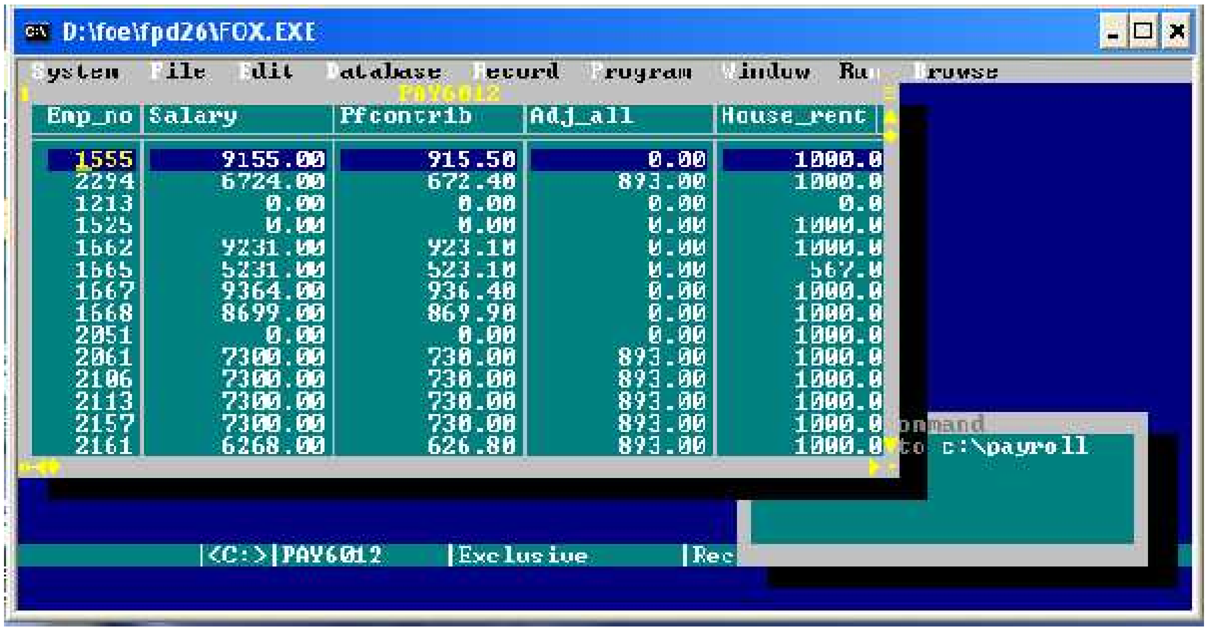

```
c:\fpd26\FOX.EXE
                                                        2:41:08 pm
                    KARMACHARI SANCHAYA KOSH.
               PAYROLL FOR THE MONTH  OF BHADRA 2063.
STAFF CODE NO : 1539   NAME : Rajendra Shakya     PF.NUMBER:  828716
DEPARTMENT    : 25     POST : ACT.SYS.ANL.
==========================================================================
   SALARY            :     11951.00      PF. DEDUCTION     :  2390.20
   PF. CONTRIBS      :      1195.10      INSURANCE PREMIUM: 2464.00
   ADJ.ALLOWANCES    :         0.00      PAKI BIMA         :     0.00
   INCENTIVE         :     3000.00       HOUSE REPAIR      :     0.00
   20 % ALLOWANCES   :         0.00      PARIBAHAN LOAN    : 1130.00
   SP. ALLOWANCES    :         0.00      CYCLE LOAN        :     0.00
   D. ALLOWANCES     :         0.00      TELEPHONE         :     0.00
   T. ALLOWANCES     :       625.00      SUNDRY DEBTORS    :     0.00
                                         N.LAGANI KOSH     : 1000.00
                                         TAX               :   970.00
                                         PF. LOAN          :     0.00
             SOCIAL LOAN     :   868.50  COMPUTER LOAN     :   465.00
             SPECIAL LOAN    :     0.00  COMPUTER BIMA     :     0.00
```

Figure 7.3 Program Structure in current scenario



Emp_no	Salary	Pfcontrib	Adj_all	House_rent
1555	9155.00	915.50	0.00	1000.0
2294	6724.00	672.40	893.00	1000.0
1213	0.00	0.00	0.00	0.0
1525	0.00	0.00	0.00	1000.0
1662	9231.00	923.10	0.00	1000.0
1665	5231.00	523.10	0.00	567.0
1667	9364.00	936.40	0.00	1000.0
1668	8699.00	869.90	0.00	1000.0
2051	0.00	0.00	0.00	1000.0
2061	7300.00	730.00	893.00	1000.0
2106	7300.00	730.00	893.00	1000.0
2113	7300.00	730.00	893.00	1000.0
2157	7300.00	730.00	893.00	1000.0
2161	6268.00	626.80	893.00	1000.0

```
ommand
to c:\payroll
```

```
|<C:>|PAY6012        |Exclusive        |Rec
```

Figure7.4 Legacy Database of KSK

7.2 N-Tire Programming   Technique Components

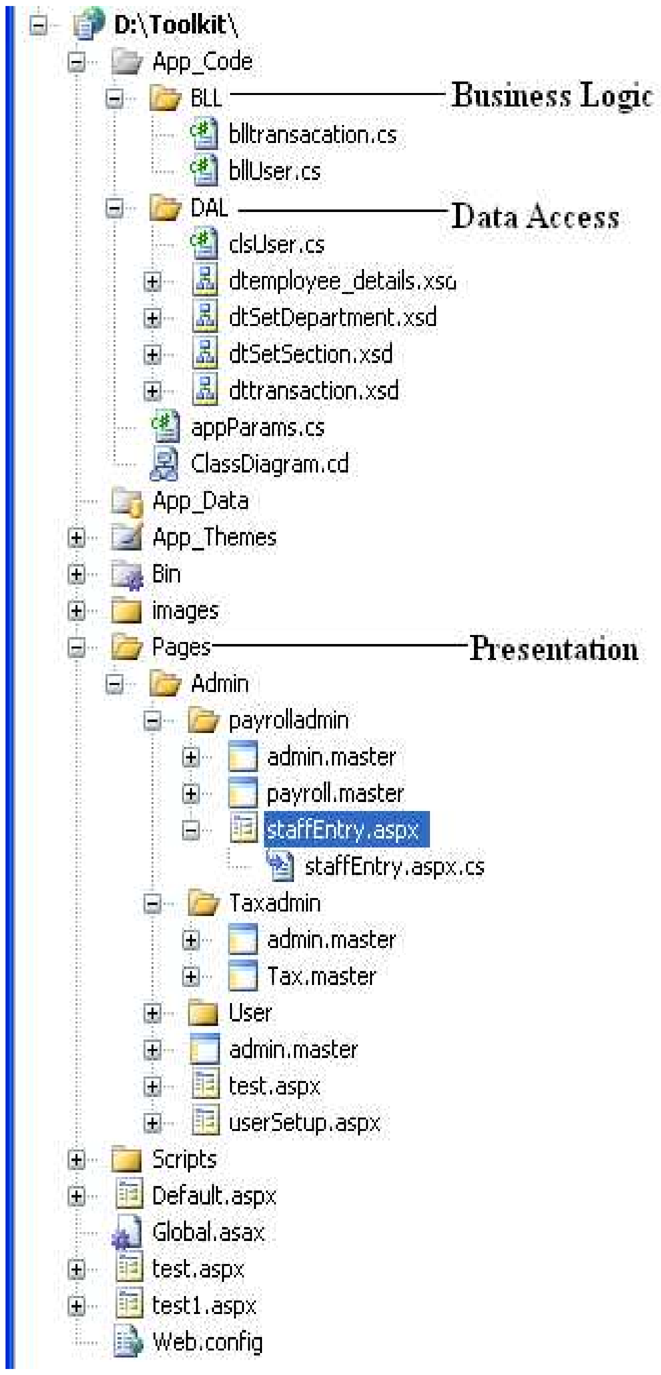


Figure 7.5 N-tire Programming Components

7.3 Wrapper Schema of Legacy Database

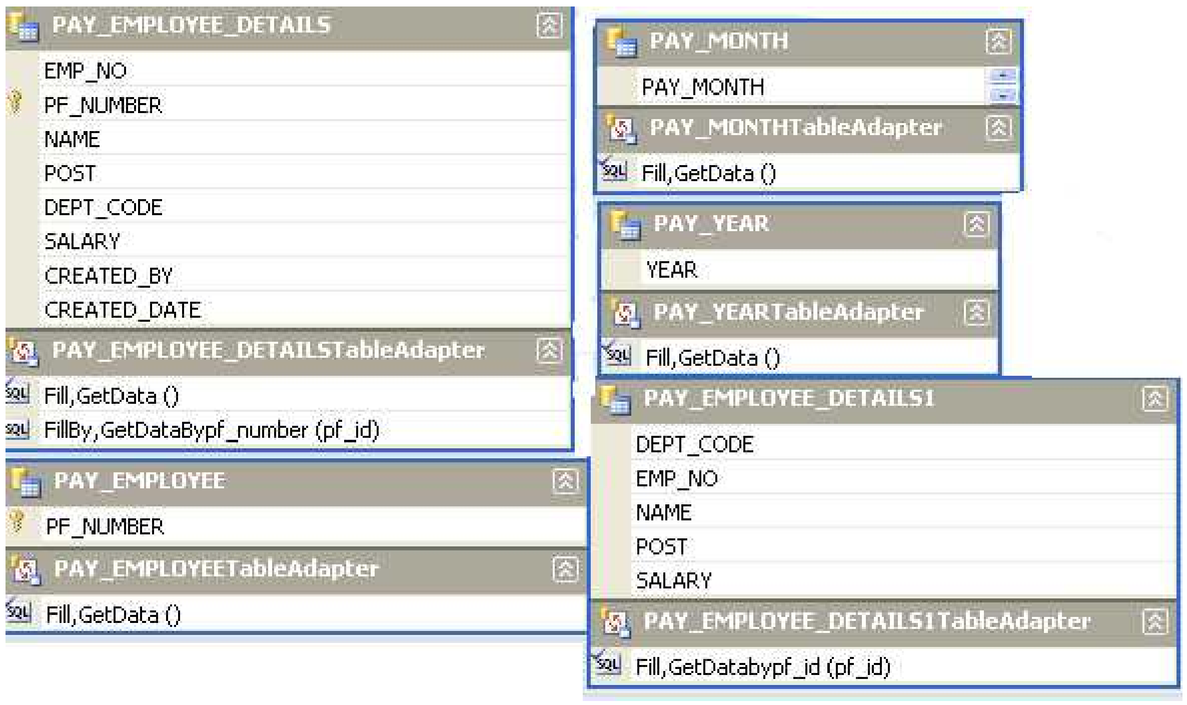
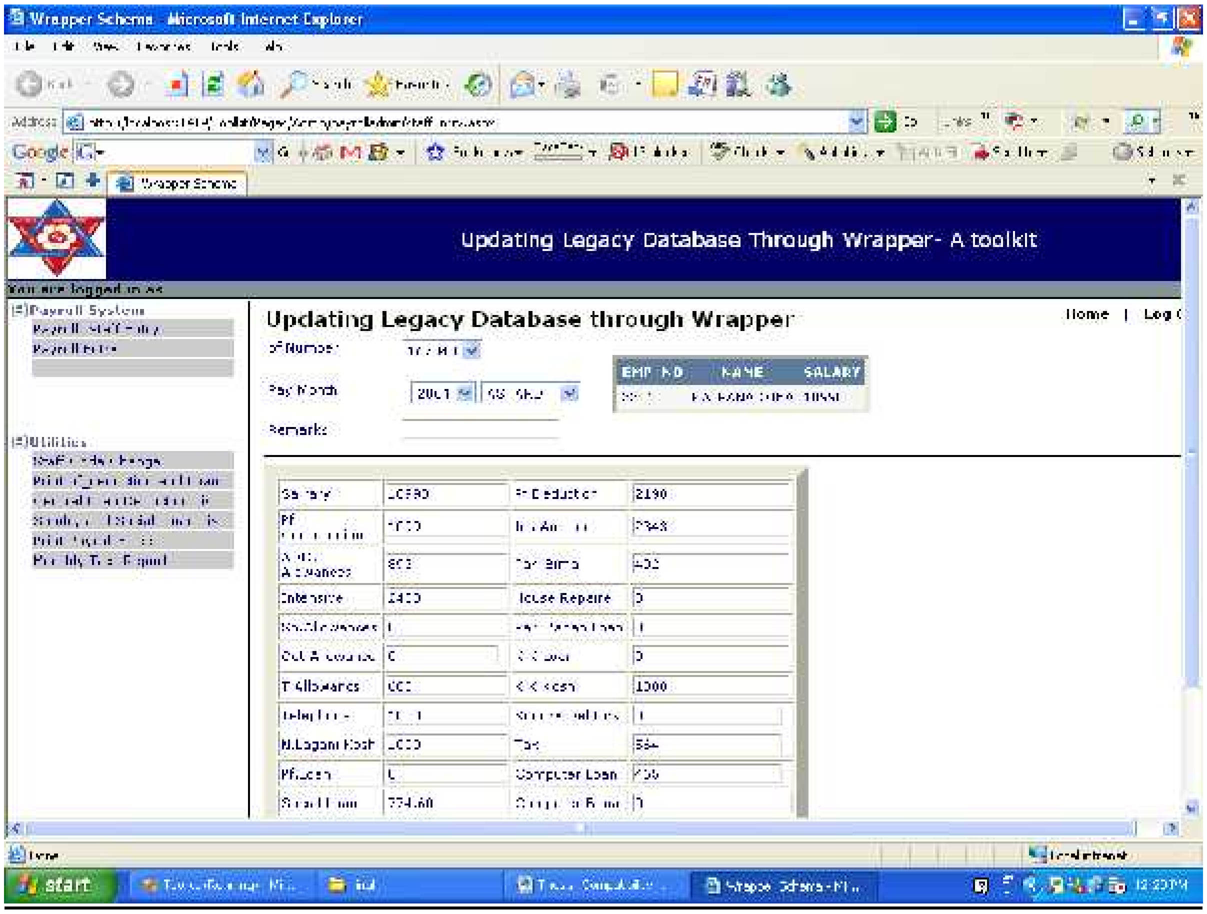


Figure 7.6 Wrapper Schema of Legacy System

7.4 Testing

In this section, a sample testing and the result have been described. The testing was carried out on the legacy Database of KSK Payroll Management System. The sample program for testing is given below.

7.4.1. Main Screen



Wrapper Scheme - Microsoft Internet Explorer

Updating Legacy Database Through Wrapper- A toolkit

You are logged in as

Payroll System

Utilities

Updating Legacy Database through Wrapper

Home | Log

PF Number:	[illegible]		
Pay Month	2061	[illegible]	
Remarks			

EMP NO	NAME	SALARY
[illegible]	[illegible]	10550

Salary	10580	PF Deduction	2190
PF Contribution	[illegible]	[illegible]	2348
[illegible] Allowances	800	[illegible] Bima	400
Intensive	2400	House Repaire	0
[illegible] Allowances	0	[illegible] Loan	0
Out Allowance	0	[illegible]	0
T Allowance	000	[illegible]	1000
Telephone	[illegible]	[illegible]	0
N.Lagani Kosh	1000	Tax	564
PfLoan	0	Computer Loan	756
[illegible]	774.60	[illegible]	0

Figure 7.7 Main screen of toolkit

7.4.2. Programming technique to update legacy database through wrapping Technique

7.4.2.1 BLL

```
public string addtransaction()
    {
        try
        {
            dttransactionTableAdapters.PAY_TRANSACTION_DETAILSTableAdapter
adapter = new
dttransactionTableAdapters.PAY_TRANSACTION_DETAILSTableAdapter();
            dttransaction.PAY_TRANSACTION_DETAILSDataTable datatable =
adapter.GetDatabypf_number(this.Bpf_number);
            dttransaction.PAY_TRANSACTION_DETAILSRow datarow =
datatable[0];
            datarow.EMP_NO = this.Bemp_no;
            datarow.SALARY = this.Bsalary;
            datarow.PF_CONTRIB = this.Bpf_contrib;
            datarow.ADJ_ALL = this.Badj_all;
            datarow.HOUSE_RENT = this.Bhouse_rent;
            datarow.HOUSE_RENTAL = this.Bhouse_rental;
            datarow.SPECIAL_ALL = this.Bspecial_all;
            datarow.OTHER_ADD = this.Bother_add;
            datarow.TIFF_ALL = this.Btiff_all;
            datarow.TOTAL_SAL = this.Btotal_sal;
            datarow.PF_DED = this.Bpf_ded;
            datarow.BEEMA = this.Bbeema;
            datarow.HOUSE_LOAN = this.Bhouse_loan;
            datarow.H_REPAIR = this.Bh_repaire;
            datarow.SOC_LOAN = this.Bsoc_laon;
            datarow.MED_LOAN = this.Bmed_loan;
            datarow.CYCLE_LOAN = this.Bcycle_loan;
            datarow.TELEPHONE = this.Btelephone;
            datarow.SUNDRY_DR = this.Bsundry_dr;
            datarow.TAX = this.Btax;
            datarow.TOT_DED = this.Btot_ded;
            datarow.NET_PAY = this.Bnet_pay;
            datarow.OTHER_SUBS = this.Bother_subs;
            datarow.PF_LOAN = this.Bpf_loan;
            datarow.PAMONTH = this.Bpay_month;
            datarow.SPECIAL_LOAN = this.Bspecial_loan;
            datarow.PF_NUMBER = this.Bpf_number;
            datarow.CO_LOAN = this.Bco_loan;
            datarow.CO_BIMA = this.Bco_beema;
            datarow.KK_LOAN = this.Bkk_loan;
            datarow.NL_KOSH = this.Bnl_kosh;
            datarow.KK_KOSH = this.Bkk_kosh;
            datarow.INCENTIVE = this.Bincentive;
            datarow.TEN_ALLOW = this.Bten_allow;
            datarow.OUT_ALLOW = this.Bout_allow;
            datarow.PAR_PREM = this.Bpar_prem;
            datarow.PAR_LOAN = this.Bpar_loan;
            datarow.CREATED_BY = this.Bcreated_by;
            datarow.CREATED_DATE = this.Bcreated_date;
            datarow.PAYMENT_FLAG = this.Bpayment_flag;
            adapter.Update(datarow);
             return "Information update in your Account";
        }
        catch (System .Exception err)
        {
            return (err.Message.ToString());
        }
```

7.4.2.2. DAL(Wrapper Schema)

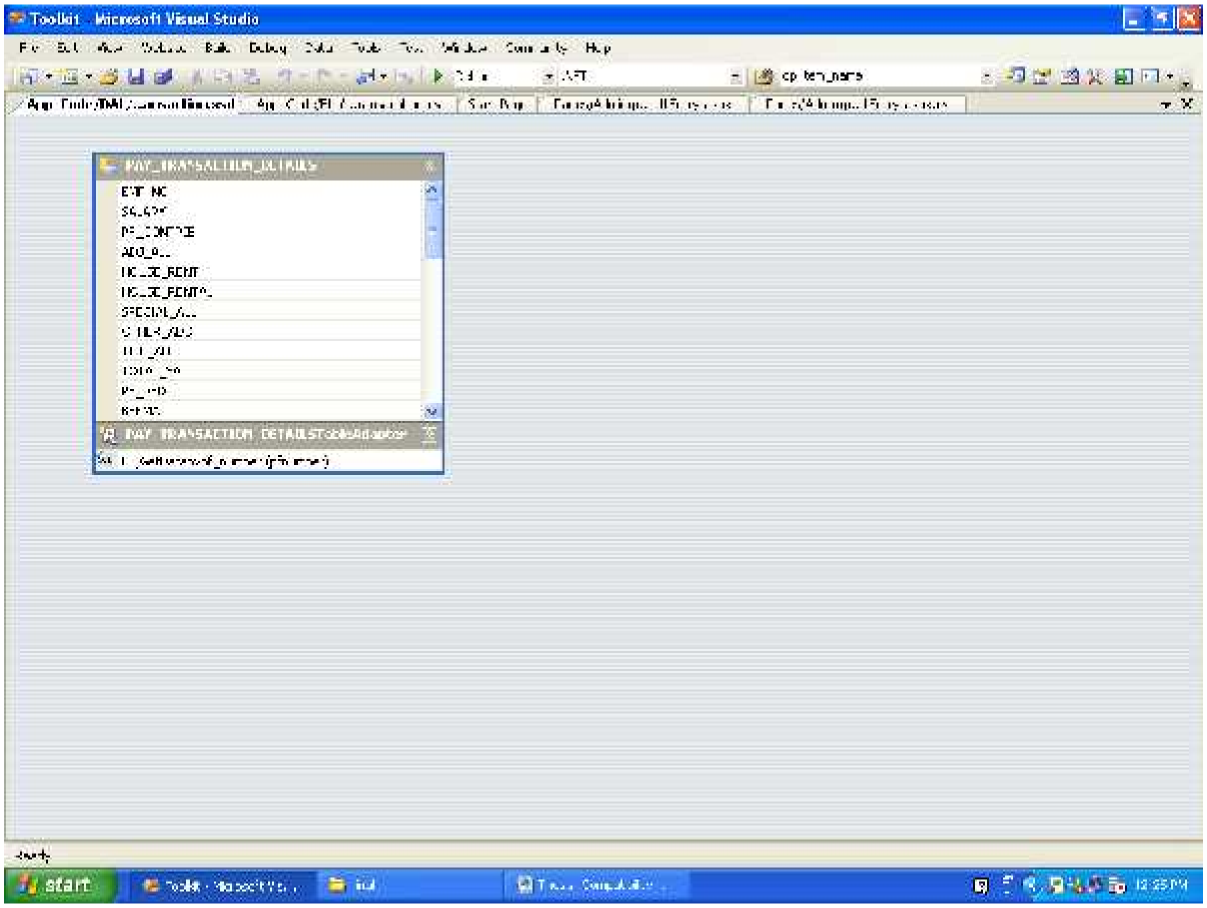

Toolkit - Microsoft Visual Studio
PAY_TRANSACTION_DETAILS
PAY_TRANSACTION_DETAILSTableAdapter
start

Figure 7.8 Data Access Layer of N-tire Programming Technique

7.4.2.3. Presentation Layer

```
protected void cmdSave_Click(object sender, EventArgs e)
    {
       Int64  total = System.Convert .ToInt64 (string .Concat
(DropDownList2 .Text .Trim (),DropDownList3.SelectedValue.Trim ()));
       TextBox2.Text = total.ToString();
       blltransacation tran = new blltransacation();
       tran.Bpf_number  =
System.Convert.ToInt64(DropDownList1.SelectedItem.Text.ToString());
       tran.Bsalary = System.Convert.ToInt64(txtsalary.Text.ToString());
       tran.Bpf_contrib =
System.Convert.ToInt64(txtcontribution.Text.ToString());
       tran.Badj_all =
System.Convert.ToInt64(txtalowances.Text.ToString());
       tran.Bincentive =System.Convert
.ToInt64(txtIntensive.Text.ToString());
       tran.Bspecial_all =
System.Convert.ToInt64(txtspallowances.Text.ToString());
```

```
        tran.Bout_allow = 
System.Convert.ToInt64(txtoutallowances.Text.ToString());
        tran.Btiff_all = 
System.Convert.ToInt64(txtTallowance.Text.ToString());
        tran.Bpay_month = System.Convert.ToInt64(total.ToString());

        tran.Btelephone = 
System.Convert.ToInt64(txttelephone.Text.ToString());
        tran .Bnl_kosh = System.Convert .ToInt64(txtLagani_Kosh.Text 
.ToString ());
        tran.Bpf_loan = System.Convert.ToInt64(txtpf_loan.Text.ToString());
        tran.Bsoc_laon = 
System.Convert.ToInt64(txtsocial_loan.Text.ToString());
        tran.Bspecial_loan = 
System.Convert.ToInt64(txtspcial_loan.Text.ToString());
        tran.Bpf_ded = 
System.Convert.ToInt64(txtdeduction.Text.ToString());
        tran.Bbeema = 
System.Convert.ToInt64(txtInsurance_amount.Text.ToString());
        tran.Bpar_prem = 
System.Convert.ToInt64(txtpari_bina.Text.ToString());
        tran.Bh_repaire = 
System.Convert.ToInt64(txthouse_Repaire.Text.ToString());
        tran.Bpar_loan = 
System.Convert.ToInt64(txtpari_bahan_loan.Text.ToString());
        tran.Bkk_loan = System.Convert.ToUInt32 
(txtkk_loan.Text.ToString());
        tran.Bkk_kosh = System.Convert.ToInt64(txtkk_kosh.Text.ToString());
        tran.Bsundry_dr = 
System.Convert.ToInt64(txtsundry_debtors.Text.ToString());
        tran.Btax = System.Convert.ToInt64(txttax.Text.ToString());
        tran.Bco_loan = 
System.Convert.ToInt64(txtcomputer_loan.Text.ToString());
        tran.Bco_beema = 
System.Convert.ToInt64(txtcomputer_bima.Text.ToString());
        Label1.Text = tran.addtransaction();



        MultiView1.Visible = false;
        cmdSave.Enabled = false;


    }
```

Chapter 8

Conclusion and Further Recommendations

8.1 Conclusion

Wrapping databases has been studied for years leading to prototypes that insure a data model and query independence in database systems. The contribution of this thesis is on the data consistency aspects of database wrapping- A programming Technique.

This thesis has explored the principles of a wrapper architecture that provides update through a wrapper schema that integrates not only explicit but also implicit structures and constraints. Since the elicited constraints have been coded in the wrapper newly developed programs can profit from automatic constraint management that the underlying DBMS cannot ensure.
One of the most challenging issues was building the inter-schema mappings. Thanks to a formal transformational approach to schema engineering. It was possible to build the mappings that derive the query decomposition and the data recomposition as well as the implicit constraint checking emulation.

The wrapper development is supported by a CASE tool [23] that gives the developer an integrated toolset for schema definition, inter-schema mappings definition and processing and code generation.

8.2 Further Recommendations

This Work has focused on the N-tire application Programming technique in database wrappers that maintain the database Consistency. This process relies on a special kind of inter-schema mapping, namely sequences of transformations. By replacing the schemas constructs names in the wrapper query with their database equivalent produce a database query that can be executed on the actual data. This systematic approach can be automated in such a way that wrappers can be generated based on the schema mappings. Considering two schemas and their mappings expressed by a sequence of transformations. It generates wrappers for FOXPRO files and relational databases. The approach and the tool have been applied among others for building federated databases mixing both legacy and modern technologies.

One problem encountered when building wrappers is coping with non standard constructs. The generic model despite its power, cannot express all integrity con-straints. When a specific constraint is found in the reverse engineering process. It is expressed as a generic constraint described by a free text annotation. The wrapper generator includes in the wrapper code a skeleton that documents the constraint. It is up to the programmer to write the specific code for this constraint. Therefore, the technology have developed is being integrated into a development environment for business-to-customer applications that are built on top of legacy databases this is just a n-tire application architecture as Service oriented architecture.

REFERENCES

[1] Philippe Thiran, G.-J. Houben, J.-L. Hainaut. Databases wrappers Developement:Towards Automatic Generation. In I. C. Press, editor, WCRE Proceedings,2006 , Vol. 10, pp. 529-564..

[2] P. Thiran, G.-J. Houben, J.-L. Hainaut, and D. Benslimane. Updating legacy databases through wrappers: Data consistency management. In I. C. Press, editor, WCRE Proceedings,2004, Vol. 7, pp. 300-340..

[3] M. Fernandez, W. Tan, and D. Suciu. Silkroute: Trading between relations and XML. In Proceedings of the Ninth International World Wide Web Conference, 2000.

[4] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. Journal of Intelligent Information Systems, 8(2):117–132, 1997.

[5] G. Graefe. Query evaluation techniques for large databases. ACM Computing Surveys, 25(2):73–170, 1993.

[6] J.-L. Hainaut. Introduction to database reverse engineering. Technical report, University of Namur, 2002.

[7] J.-L. Hainaut. Transformation of Knowledge, Information and Data: Theory and Applications, chapter Transformationbased Database Engineering. IDEA Group, 2005 Vol. 13, pp 512-548.

[8] J.-M. Hick, V. Englebert, J. Henrard, D. Roland, and J.-L. Hainaut. The DB-MAIN database engineering CASE tool (version 6.5) - functions overview. Db-main technical manual, Institut d'informatique, University of Namur, 2002.

[9] D. Lee, M. Mani, F. Chiu, and W. W. Chu. NeT and CoT: Translating relational schemas to XML schemas using semantic constraints. In ACM International Conference on Information and Knowledge Management, 2002.

[10] I. Manolescu, D. Florescu, and D. K. Kossmann. Answering fXMLg queries over heterogeneous data sources. In VLDB, pages 241–250, 2001.

[11] P. McBrien and A. Poulovassilis. Automatic migration and wrapping of database applications - a schema transformation approach. In International Conference on Conceptual Modeling / the Entity Relationship Approach, pages 96–113, 1999.

[12] P. McBrien and A. Poulovassilis. Schema evolution in heterogeneous database architectures. In CAiSE Proceedings. Springer-Verlag, 2002.

[13] J. Shanmugasundaram, J. Kiernan, E. J. Shekita, C. Fan, and J. Funderburk. Querying XML views of relational data. In Proceedings of the 27th VLDB Conference, 2001 ,pp. 185-222.

[14] P. Thiran. Legacy Database Federation - A Combined Reverse-Forward Approach. Phd thesis, University of Namur, October 2003.

[15] P. Thiran, F. Estievenart, J.-L. Hainaut, and G.-J. Houben. Exporting databases in XML a conceptual and generic approach. Proc. of CAiSE Workshops (WISM04), 2004.

[16] P. Thiran and J.-L. Hainaut. Wrapper development for legacy data reuse. In I. C. Press, editor, WCRE Proceedings, 2001.

[17] P. Thiran, G.-J. Houben, J.-L. Hainaut, and D. Benslimane. Updating legacy databases through wrappers: Data consistency management. In I. C. Press, editor, WCRE Proceedings,2004.

[18]http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/aina/2003/1906/00/1906toc.xml&DOI=10.1109/AINA.2003.1192945.

[19] Hongzhi Wang, Jianzhong Li, Zhenying He, "An Effective Wrapper Architecture to Heterogeneous Data Source," aina, p. 565, 17 th International Conference on Advanced Information Networking and Applications (AINA'03), 2003

[20] Hainaut, J.-L., Roland, D., Hick, J.-M., Henrard, J., Englebert, V.: "Database Reverse Engineering:from Requirements to CARE Tools", Journal of Automated Software Engineering, 3(1), 1996.

[21] Hainaut, J.-L.: "Specification preservation in schema transformations - Application to semanticsand statistics", Data & Knowledge Engineering, 16(1), Elsevier Science Publish, 1996.

[22] Papakonstantinou, Y., Gupta, A., Garcia-Molina, H., Ullman, J.: "A Query Translation Scheme for Rapid Implementation of Wrappers", in Proceedings of the international Conference on Declarative and Object-oriented Databases, 1995.

[23] http://www.db-main.be/doc.php?id=ro12&tid=2&docid=61&lang=2

[24] Aiken, P., Muntz, A., Richards, R.: "DOD Legacy Systems - Reverse-Engineering Data Requirements", Communications of the ACM, May 1994.

[25] Bull: DSE I-D-S/II (FOXPRO) Reference Manual, Bull, 1993.

[26] Brodie, M. L., Stonebraker, M.: Migrating Legacy Systems: Gateway, Interfaces & the Incremental Approach, Morgan Kaufmann, 1995.

[27] Henrard, J., Hick, J-M., Thiran, Ph., Hainaut, J-L.: "Strategies for Data Reengineering", in Proeedings. of WCRE'02, IEEE Computer Society Press, 2002. 42

[28] Thiran, Ph., Hainaut, J.-L.: "Wrapper Development for Legacy Data Reuse", in Proceedings of WCRE'2001, IEEE Computer Society Press, 2001.

[29] Thiran, Ph., Hainaut, J-L., Integration of Legacy and Heterogeneous Databases, LIBD Publish., Namur, 2002.

[30] van den Brand, M.G.J., Klint, P.: ASF+SDF Meta-Environment User Manual, 2003.

[31] Cleve, A.: Data-centered Applications Conversion using Program Transformations, Master's Thesis, University of Namur, 2004.

[32] http://www.info.fundp.ac.be/libd/rubrique.php3?id_rubrique=30

[33] S. Bergamaschi, S. Castano, D. Beneventano, and M. Vinci. Retrieving and integrating data for multiple sources: the momis approach. Data and Knowledge Engineering, 36, 2001 Vol. 10, pp. 529-564.

[34] M. J. Carey, D. Florescu, Z. G. Ives, Y. Lu, J. Shanmugasundaram, E. J. Shekita, and S. N. Subramanian. XPERANTO: Publishing object-relational data as XML. In WebDB (Informal Proceedings), pages 105–110, 2000.