Automated Reverse Engineering of Legacy 4GL Information System Applications Using the ITOC Workbench

John V. Harrison and Wie Ming Lim

Centre for Software Maintenance
Department of Computer Science and Electrical Engineering
The University of Queensland, Brisbane, QLD 4072, Australia
E-mail: {harrison|wieming}@csee.uq.edu.au

Abstract. Most contemporary fourth-generation languages (4GLs) are tightly coupled with the relational database and other subsystems provided by the vendor. As a result, organisations wishing to change database vendors are typically forced to rewrite their applications using the new vendor's 4GL. The anticipated cost of this redevelopment can deter an organisation from changing vendors, hence denying it the benefits that would otherwise result, for example, the exploitation of more sophisticated database technology. If tools existed that could reduce the rewriting effort, the option of changing database vendors would become more economically feasible.

The ITOC project is a large collaborative research initiative between the Centre for Software Maintenance at the University of Queensland and Oracle Corporation. The primary goal of the project is to develop tools to assist in the migration of 4GL information system applications. A tool resulting from the project has been utilised to recover design information from several deployed commercial applications. This paper describes the tool, evaluates its performance when applied to these applications and provides insight into the development of "industrial strength" re-engineering tools.

1. Introduction

There has been a significant investment in the development of information systems built using fourth-generation programming languages (4GLs). Although there have been considerable advances in the design of both 4GL languages and their associated development environments, most are still proprietary. There are no open industry standards, as there are with third-generation programming languages (3GLs) such as COBOL. The dependency of the customer on a particular vendor often prevents the customer from realising the benefits of newer technologies without incurring a very large redevelopment cost. Thus, in order to make application migration economically feasible, it is important to develop tools which will assist in the migration process.

Most work in information system reengineering addresses either the data repository alone, or different aspects of migrating 3GL applications that access a network, hierarchical or some other legacy data repository. While there remains significant demand for tools that assist in this domain, the pace of technological advance in database systems, as well as the high level of competition amongst

database vendors, has resulted in 4GL-based information system applications now being considered as "legacy" by their organisations.

The ITOC Design Recovery Tool (Harrison, et al., 1995, Berglas and Harrison, 1997, Harrison and Berglas, 1997, Harrison, et al., 1997) was developed collaboratively by the Centre for Software Maintenance at The University of Queensland, and Oracle Corporation, and has been deployed to assist with the recovery of both the application semantics and the static schema definition from Ingres ABF ™ 4GL applications. The recovered design components are loaded into the Oracle Designer 2000™ CASE repository, and can then be used to forward engineer an application in several representations, for example, HTML, Visual Basic and Oracle Developer 2000™ (a "form-based" implementation environment).

To the best of our knowledge, this is the first 4GL design recovery technology that has been implemented beyond a prototype stage. Existing techniques that we believed would address part of the design recovery task were determined to be insufficient when applied to deployed commercial applications. Consequently, we concur with the observations and conclusions made by (Blaha and Premerlani, 1995), which state that many techniques fail to be effective when confronted with the idiosyncrasies that occur in real applications.

The next section provides a general introduction to the characteristics of 4GL information system applications using as examples the source and target development environments selected for the project. Following that, we describe the basic structure and functionality of the tool. Section four presents our results, experience, and our evaluation of the applying the ITOC tool to legacy applications. We then describe related work and conclude with a summary and future research direction.

2. Information System Application Characterics

4GL-based information systems are comprised of a user interface, application logic and a relational database management system (RDBMS). Much of the power of 4GLs is derived from the recognition that most user interfaces can be modelled as *forms*, which can be viewed as electronic representations of their paper-based counterparts. Forms typically contain a number of fields that correspond directly to columns in the database. The 4GL development enviroment environment is utilised to provide the (considerable) amount of user interface functionality necessary to faciliate cursor movement, e.g., to monitor cursor movement from field to field and perform basic validation. Fragments of procedural 4GL code are invoked when complex validation is necessary.

For example, consider the sample Ingres ABF "order entry" form illustrated in Figure 1. In order to implement this form the programmer would declaratively specify the name, data type and position of each field, and also the fixed "boiler plate" text such as the string "Invoice Num". Most 4GLs will then automatically implement logic that permits the user to move the cursor from field to field, ensure that alphabetic characters are not entered into numeric fields, and enable multiple records such as the products purchased on the order to be scrolled in the scrolling region. Automating this functionality is an advantage of using 4GLs because implementation using a 3GL is both tedious and error prone. However, 4GL code is then required to

perform higher level validation and processing such as ensuring that both the Billing and Shipping Customer Numbers are valid, retrieving the Customers' Name, and calculating the "Total Delivered Cost".

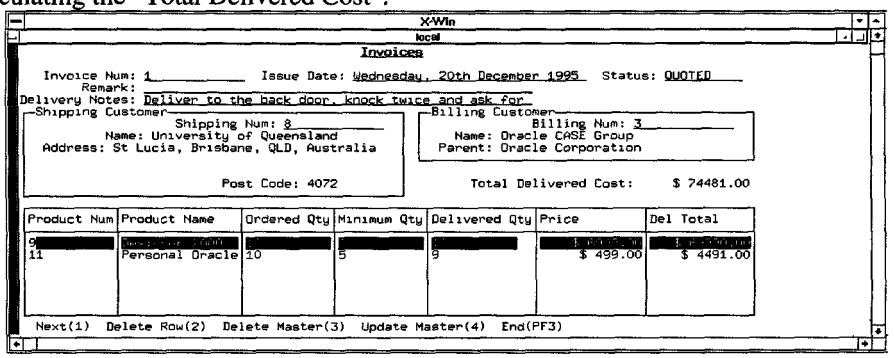

Figure 1: A Sample Ingres ABF Form

The Ingres 4GL development environment requires a considerable amount of 4GL code to be written to retrieve records from the database and display them on the form, and then to update changed information. An Ingres ABF application is comprised of ABF *frames*. Each frame is comprised of user interface items that appear on a particular screen, termed an ABF *form*[1], and code which implements the application logic associated with that particular screen. The code contains embedded SQL, which is typically used to retrieve values from the database into fields on the screen, and vice versa.

While 4GLs reduce the amount of code required to build user interfaces, CASE tools can further improve productivity by also automating the code that is required to move data between the form and the underlying database. A developer using a CASE tool can avoid the necessity of writing code explicity to implement certain application logic and creating the user interface of each module. These components are generated automatically from a very high level description of the application, which is input into the CASE repository.

For example, the Oracle CASE tool allows forms to be generated automatically based on a "Module Data Diagram", which is made up of "Module Detail Table Usages" (MDTUs). A MDTU shows how a database table is to be used to construct a specific module. Within a MDTU, "Module Detail Column Usages" (MDCUs) show the usage of individual database columns within a particular module.

Figure 2 shows a Module Data Diagram corresponding to the application illustrated in Figure 1. Each rectangle on the diagram represents a usage of a table by this module, namely a MDTU. The layout of the MDTUs in the Module Data Diagram is important as it represents the relationship between the underlying database tables. The relative placement of the MDTUs in a module ultimately determines the appearance and functionality of the generated target application.

[1] Note that the term "form" is overloaded here. Ingres ABF "frames" correspond to the form construct described earlier. This is a simple example of the lack of standards in this domain.

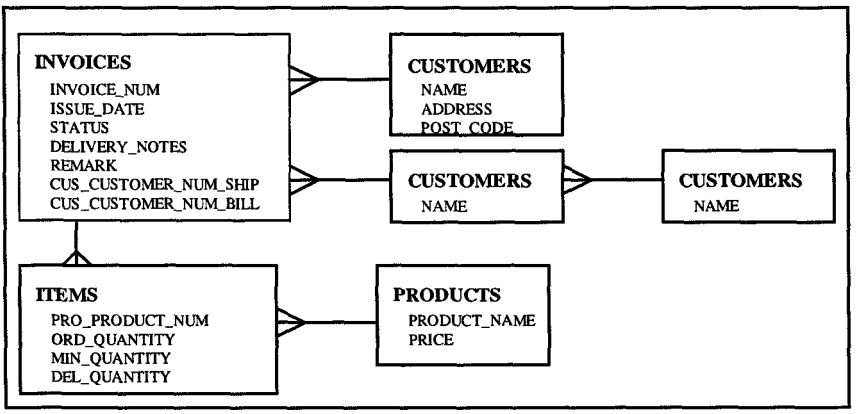

Figure 2: Module Data Diagram for the Invoices Frame

The CASE environment supports two types of relationships, or *links*, between MDTUs. These relationships are termed *master-detail* and *look-up,* which are terms that are commonly used by practitioners using different 4GL environments. Both master-detail and look-up relationships are only specified if a foreign key exists between the database tables on which the MDTUs are based. For example, a *"master-detail"* relationship exists between the INVOICES and ITEMS MDTUs in Figure 2 (note the placement of the former MDTU being *above* the latter). This relationship implies that a foreign key exists between the ITEMS and INVOICES database tables.

Similarly, a *"look-up"* relationship, such as the one between the CUSTOMERS and INVOICE MDTUs in Figure 2, can only be defined if a foreign key exists between the INVOICES and CUSTOMERS database tables. *Look-ups* are positioned to the right of the referenced MDTU on the diagram.

On the basis of the above Module Data Diagram (MDD), the CASE tool can generate an application in several representations, such as the Oracle Developer 2000™, HTML and Visual Basic. These representations would offer equivalent functionally to the Ingres implementation illustrated in Figure 1. The MDD graphical description is all that need be specified to enable the CASE tool to automatically generate the code required to select Invoice and Items rows from the database, look up and validate their Customer and Product Numbers, and update the database in response to user input.

3. Design Recovery From Information Systems

The ITOC design recovery process involves the extraction of information from an Ingres relational database application, and the conversion of this information into Oracle CASE repository *elements*. An overview of the process appears in Figure 3. The ITOC tool uses information from the Ingres relational database schema as obtained from the database catalogue, the user interface (screen) definitions as extracted using the Ingres "copyapp" utility and 4GL procedural code that is contained within the source, i.e., ".osq", files. After processing, the tool loads the

recovered design information into the CASE repository. Executable applications, in various representations, can then be created automatically using code generators.

The processing steps implemented by the ITOC tool are shown in detail in Figure 4. Each step produces objects that are defined by the ITOC analysis schema, and then used as inputs to the next step. The output of the tool are CASE elements that define the schema and modules needed to construct the Module Data Diagrams, and other design information, that corresponds to the source application. The following sections provide an overview of the processing performed at each step. Many details have been omitted due to space limitations. A more comprehensive description can be found in (Harrison and Berglas, 1997).

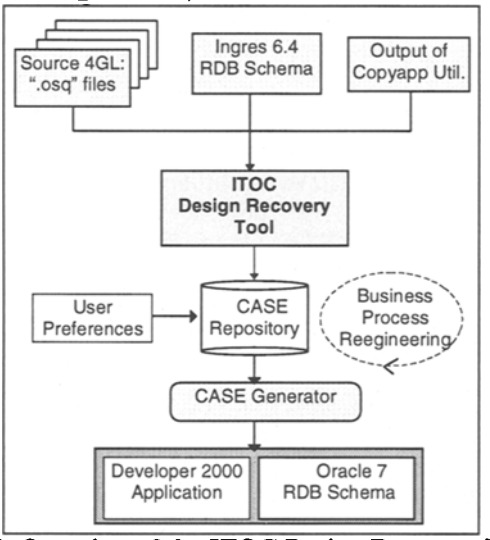

Figure 3: Overview of the ITOC Design Recovery Process

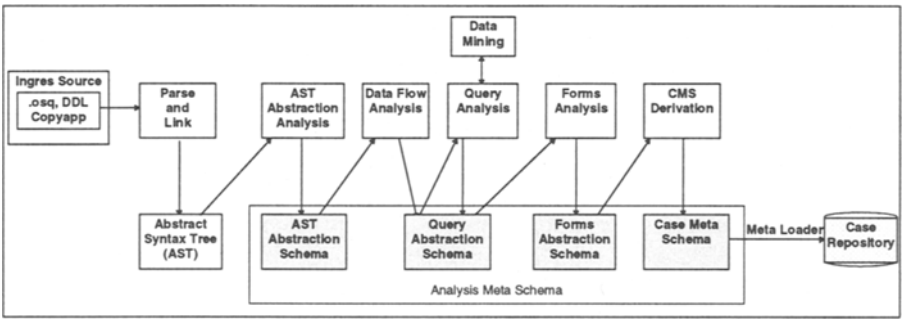

Figure 4: Steps of the ITOC Design Recovery Process

3.1 Parse and Link

The first step is to parse the source code and link uses of each identifier to its definition. This was performed using Reasoning System's Refinery™ environment, including the Refine™ programming language and the Dialect™ compiler compiler. We found that, unlike many other compiler compilers, in addition to parsing the

source file given a formal grammar of the source language, Dialect automated the production of an abstract syntax tree.

The Abstract Syntax Tree (AST) output consists of objects which represent instances of terminal and non-terminal nodes in the grammar, together with derived information such as the links to identifier definitions. The *Language eXtension WorkBench* (LXWB) (Peake, 1996) was used to extend Dialect to unify the definitions of the formal grammar and the objects that form the AST.

3.2 AST Abstraction

Once the AST is created, it is compressed into a high level abstraction which is more suitable for further analysis. Defining an explicit abstraction of the source code enabled different syntactic structures to be processed in a consistent manner. For example, while most queries are represented by explicit SQL statements within the 4GL code, it was discovered that a common Ingres coding practice is to write 3GL programs which perform implicit queries by scanning plain text tables of static data on the client side of a client/server environment. AST abstraction enabled these procedure calls to be abstracted, and subsequently treated exactly the same as if they were SQL queries despite the fact that they have a radically different syntactic structure. The results of this processing phase are recorded in the AST Abstraction Schema, which is used as input into the following phase.

3.3 Data Flow Analysis

The ITOC design recovery process relies on analysis of data flow in the Ingres source code. The data flow analysis is heuristic-based and differs from the conventional use of data flow analysis (Callahan 1988, Marlowe and Ryder 1990, Ryder et. al. 1990). The heuristic we adopted assumes that if a data flow exists from a source object **A** to a target object **B**, then the value of object **A** is coupled, and hence considered equal, to object **B**. Based on our experience applying the tool, which is described below, we found this heuristic to be effective.

The data flow analysis phase consists of two sub-phases. Initially, all the direct data flows between variables and database columns are recorded. For example, the first line of the first SQL query variant in the code fragment below contains a data flow from `I.Invoice_Num` to the variable `:Invoice_Num` in the first query statement. Note that the data flows are recorded between each reference in the code to a database column, which we term *query columns*.

```
Select    :Invoice_Num = I.Invoice_Num
From      INVOICES  I
Where     :Billing_Cust_Num = I.Billing_Cust_Num
...
Select    :Billing_Cust_Name = C.Cust_Name
From      CUSTOMERS  C
Where     C.Cust_Num = :Billing_Cust_Num
...
```

On the basis of the direct data flows, the transitive links that exist between two query columns, as opposed to between a query column and a variable, are computed. These transitive data flows are needed to capture the implicit enforcement of foreign

keys, such the one between the "I.Billing_Cust_Num" and "C.Cust_Num" query columns via the ":Billing_Cust_Num" variable appearing in the code fragment. Note that this phase only computes the transitive data flows. The derivation of foreign keys is done in the following phase.

3.4 Query Analysis and Data Mining

A fundamental part of the ITOC design recovery process addresses the recovery of foreign keys using information obtained from the Ingres 4GL code. This recovery is necessary as the Ingres Data Definition Language (DDL) does not support the explicit declaration of foreign keys, which form an integral part of the construction of Module Data Diagrams.

Based on the data flow, both direct and indirect, recorded in the previous phase, the ITOC tool computes references between *query tables,* which are references to a database table appearing in a query. A *reference* object is created for each data flow that terminates at, or originates from, a reference to a database column that forms part of a key of a relational database table. The data flow must be from a query column, as opposed to a variable. If every part of a key is *covered* by a reference object, meaning there exists a data flow from some query column to each part of the given key, then a *candidate foreign key* is generated.

The generation of the candidate foreign keys is based on heuristics about data flows. As the heuristics are not completely accurate, invalid foreign keys will be proposed. Pruning these invalid candidate foreign keys is necessary to reduces the possibility of errors in subsequent phases. The candidate foreign keys are tested using rudimentary data mining techniques applied to the original Ingres database instance.

In addition to validation of candidate foreign keys, data mining is used to derive some additional information, namely:

- Whether columns were deemed *mandatory* or *optional* by the source designer. Although Ingres contains *null* and *not-null* DDL definitions, their semantics are inconsistent with the target database environment. As a result the Ingres DDL cannot be used to determine this information directly.
- Specific ranges for column values. For example, an Order's status might have been restricted to "QUOTE", "ORDERED", or "PAID".
- Whether foreign keys participate in a one-to-one cardinality relationship.
- Whether numeric fields may be contain negative values.

The results of this phase are represented in the Query Analysis Schema, which would now contains objects which define the foreign keys, and other domain constraints, that were implemented in the source application.

3.5 Form Analysis

Form analysis identifies which database column is used to populate each field in an Ingres frame and which fields are populated with derived values. In the CASE tool, a field can be associated with at most one database column. Ambiguities related to uniquely associating a database column, via a MDCU, to a field arise from the fact that data flows can exist between more than one query column and a given field, which we were informed often occurs in practice. For example, if a billing customer

can be the same as a shipping customer, then the billing customer's details will be displayed in both the billing and shipping fields on our example *Invoices* frame. As a result, a data flow will have been created from both the shipping and billing customer names to the shipping customer field. The goal of this phase is to determine which database column should populate the shipping customer field.

The solution to resolving the above problem is based on a strategy involving numerical weighting of the relative data flows. Using weighting heuristics, which are described in detail in (Tech 1996), each field can be uniquely associated with a database column.

Note however, that not every field will necessarily be associated with a database column, for example, fields which get their value from a complex calculation perhaps involving several database columns. At present, the tool only highlights the existence of such fields, and provides links to their occurrence in the source code via a hypertext report viewable using a conventional web browser. The results of this phase are recorded in the Forms Abstraction subschema, and again used as input into the subsequent, and final, processing phase.

3.6 Module Analysis

Having uniquely associated fields and database columns, the Module Data Diagrams can be created. Module Analysis maps each Ingres frame to a corresponding Module Data Diagram, and determines the relationships / usages between the MDTUs for the particular module.

In addition to producing a mapping between each Ingres frame and a corresponding Module Data Diagram, the tool also provides the software re-engineer with alternate design options. Through analysis it was discovered that an Ingres application will often contain several distinct forms which all perform a semantically related task. For example, one Ingres form may allow the user to read Invoice details, another may allow the editing of the Invoice and a third may allow the creation of a new Invoice.

In the target 4GL, the same functionality can be implemented using one form, which allows updates, queries and insertion of new records. By examining the data flows between forms, the tool is able to suggest an alternative, more succinct representation of the source application. This phase is described in (Harrison and Berglas, 1997).

The remainder of this paper evaluates the effectiveness of the ITOC design recovery process described in the previous sections. The evaluation is based on the commercial and custom built applications that have been re-engineered using the ITOC tool.

4. Evaluation of the Tool's Performance

This section describes the results of applying the ITOC tool to four Ingres ABF applications. Three are commercial applications that have been deployed. The fourth was constructed by the ITOC research group to test the tool on an application that contained constructs permissible by the language but are rarely used in practice

according to the practitioners who participated on the project. Each application is described briefly below.

The Customer Invoicing Application (CUST-INV) is the experimental Ingres ABF application, which contains various 4GL constructs that our analysis, and discussions with practitioners, indicated were obscure and only occasionally appear in a deployed commercial application. As the tool was designed to be robust in anticipation of worldwide deployment, any permissible construct had to be addressed and tested. The Curriculum Management System (CMS) is a large Ingres application (containing 120 frames) used by the Australian Technical And Further Education Institute of Australia (TAFE) to manage their curriculum. The Curriculum Monitoring Application (CMA) is a related application (10 frames) belonging to the TAFE Institute. The Contractor Monitoring System (DME) is a contractor registering and monitoring system of the Australian Department of Mines and Energy. DME contains 41 frames which record and maintain information about contractors and their licensing details.

The tool's effectiveness was measured primarily on its ability to recover a number of major design components, namely *Tables, Module-Detail-Table-Usages (MDTU), Module-Detail-Column-Usages (MDCU), Foreign-Key Usages, Master-Detail Relationships (MDR)*, and *Look-Up Relationships (LUR)*. *Tables* refer to the database tables that are referenced in a particular Ingres form. *Foreign-Key Usages* represent the usage of a foreign key in a given module. The remaining components were introduced in Section 2.

Table 1 presents a summary of the results of the evaluation. For each application, the number of design components found in the source Ingres ABF application, by a manual line-by-line review of the code by a domain expert, appear in the columns labelled "In Source". The quantity of each design component recovered by the ITOC tool, stored in the CASE repository and also properly displayed to the developer using the CASE environment appears in the columns labelled "Recovered". The last column of Table 1 summarises the performance of the tool across all four applications.

Although these figures give a reasonable indication of the performance of the tool, other more informative criteria may exist. This topic is currently under investigation. In addition to the analysis results based on the above criteria, we describe some additional features of the tool which facilitate the overall re-engineering process.

4.1 Evaluation Results

The numerical results of the application of the ITOC tool on the four applications are shown in Table 1. Analysis indicated that omitted design information was due to one of only four reasons, namely missing foreign keys, missing MDTUs, errors in distinguishing integrity constraints, and errors displaying foreign keys that were recovered.

	CUST-INV		CMA		DME		CMS		
	In Source	Re-covered	In Source	Re-covered	In Source	Re-covered	In Source	Re-covered	Percentage Recovered
Tables Referred	10	10	14	14	83	83	219	219	100%
Foreign Keys	10	10	6	5	30	23	205	79	46.6%
MDCUs	43	43	38	38	296	296	640	640	100%
MDTUs	15	15	14	14	78	78	295	265	92.5%
MDR	4	4	3	3	13	13	48	40	88%
LUR	6	5	3	2	17	10	154	39	31%

Table 1: Results of the Analysis of the Four Applications

4.1.1 Missing Foreign Keys

As mentioned in section 3.2, we discovered that a common Ingres programming construct involved calls to 3GL procedures to retrieve code descriptions for a given code. In general, the 3GL code will contain a *look-up* to what is referred to as a "code table" such as the partial one illustrated in Table 2. In this example, the primary key of the code table is the columns *Code_Type* and *Code*.

Code_Type	Code	Columns
SUBJ_CODE	Subj_Dtls_1	Subj_Abbr_Name
SUBJ_CODE	Subj_Dtls_2	Subj_Description

Table 2: Example Code Table

In the Ingres frame, the call to one of these 3GL look-up procedures will contain a hardcoded reference to the Code_Type, for example:

```
callproc CODE_TABLE_LOOKUP('SUBJ_CODE', subj_code,
BYREF(Subj_Description));
```

Here the calling frame passes a literal ('SUBJ_CODE') and a variable (subj_code) to the 3GL procedure. This results in two data flows, one to each component of the primary key of the code table.

Although each component of the primary key of the code table is represented as query column appearing as an endpoint of a data flow, a reference object is not created for this data flow. Recall from section 3.3 that the source and destination of each relevant data flow must be from a query column (either directly or indirectly). As one of the data flows originates from a literal, a reference object will not be created, which results in a missing foreign key.

An unrecovered foreign key subsequently results in one of the *master-detail* or *look-up* relationships being missed. With reference to the results in Table 1, this case accounts for all the missing foreign keys, and then subsequently missing *look-ups*, in the CMA and DME applications, and also similar missing components in the CMS application.

4.1.2 Missing MDTUs

Analysis of the CMS application revealed that all frames constructed for querying the database were of similar structure. These frames consisted of a *master-detail* relationship between MDTUs based on the same underlying database table. The user enters search criteria into the master frame, and the results of the query are displayed in the detail frame. The ITOC tool incorrectly corrupts the design by combining the two MDTUs into one.

The error is due to incorrect formation of *query table sets*, which are abstractions used to represents the consolidation of similar queries and updates appearing in the 4GL code. For example, a *select*, *update*, and *delete* operation on the same database table, involving the same columns are represented as one query table set. This more consise representation was found to be more maintainable and would result in a better implementation after generation, as described in section 3.6.

Although the query table set principle is sound in most cases, there are instances, such as the case described above, where the tool incorrectly unifies MDTUs and hence misses foreign keys and *master-detail* relationships appearing in the CSM application. This problem of forming correct query table sets is described in (Berglas and Harrison, 1997) and is currently under investigation.

Another characteristic of the CMS application involves the population of a detail MDTU using tuples derived from a relational union of the results of several SQL queries. The version of the target CASE tool did not support the concept of relational union directly through its graphical module creation utility. As a result, in order to represent this construct in the CASE repository, the designer can either define a view definition that includes a union operation, or write a stored database (PL/SQL) procedure. At present, the ITOC tool only recovers components that are directly representable in the CASE repository, hence this construct is not recovered, which accounts for the missing MDTUs, *master-detail* relationships, and foreign keys that should have been recovered from the application.

4.1.3 Referential Integrity Constraints

The Oracle CASE tool supports relationships between MDTUs based on foreign key constraints between database tables but does not support the more general referential integrity constraint (Elmasri and Navathe, 1994). In the DME application, there are several frames that, after design recovery, would result in MDTUs connected using referential integrity constraints, but not foreign keys constraints. An example is a *look-up* to a database table that references specific tuples in the target table using only part of its key. For example, consider the following ABF-variant of an SQL *Select* statement:

```
Select    :Endorsee_Name = E.Endorsee_Name
From      ENDORSEES  E
Where     E.Licence_No = :Licence_No
```

Assuming that the primary key of the ENDORSEES table is the combination of *Endorsee_Name* and *License_No*, and that the variable *:License_No* contains a value from the LICENCES table. The relationship between the ENDORSEES and

LICENCES tables is based on a referential integrity constraint, but not a foreign key constraint.

As a result of the restriction in the particular version of the target CASE environment used in our experimentation, a link between MDTUs based on these two database tables cannot directly enforced using the declarative mechanism provided. Consequently, the ITOC tool does not attempt to recover this information, which accounts for the missing links we expected to encounter after manual analysis of the DME application.

4.1.4 Foreign Keys Recovered But Not Displayed

The ITOC tool utilises information from the user interface definitions to create MDTUs. One such component of the definition is the *field sequence*. The field sequence specifies the order in which the fields were created by the application developer. As a result of manual analysis, which was verified using practioners, we observed that the fields of an ABF screen that corresponds to a column of a *look-up* MDTU, which we term *look-up fields,* occur later in the sequence than fields in ABF screen that corresponds to a *master* MDTU, which we term *master fields.* Consequently, the ITOC tool employs a heuristic based on this observation that assists in distinguishing master and look-up MDTUs.

However, in the case of the CUST-INV application, an anomaly occurs. A *look-up* field occurs before the *master* field. This results in the failure of the tool to create the MDTU constraint, and also accounts for the missing *look-up* relationship. Note however, that the associated foreign key is recovered. It is only its usage in this module that was not recovered.

4.2 Other Recovered Design Information

In addition to the above, the ITOC tool recovers other design information from the source application. This recovery, and some informal observations, are described in this section.

4.2.1 User Interface Specification

When the user interface definition is declaratively specified in a 4GL environment, as is most often the case, it can be almost completely recovered. Re-use of the recovered specification can be of value to an organisation in certain circumstances. If the re-engineered system retains a similar, or identical, appearance and behaviour as the source system, fostering user acceptance of the re-engineered system may be easier. In addition, the cost to retrain end-users to use the re-engineered system may be reduced.

The declarative interface specification includes the position of the fields on the source application's form, the ordering sequence of the fields, the length of the fields, and the "boilerplate" text, ie., field labels, associated with each field. Text enhancement characteristics such as bold, underline, blinking, etc., are also recovered. As this information is both explicit and declaratively represented, only rudamentary information system semantic analysis is required to allow the tool to completely recover this information.

The recovered specification is loaded into the CASE repository, and is used to create templates, which are used for generating the user interface of the target application. Even when the source user interface is text-based and the target user interface is graphical (GUI) it can be easily determined that the design of the user interface originated from the source application. The templates can also be used to create new applications, unrelated to either the source or target, that possess the same "look and feel" as the source, hence retaining consistency.

4.2.2 Business Rule Recovery

Certain expressions appearing in the application logic or declarative user interface specification are both identified, and recovered, from the source application. These expressions, termed *derivations* and *validations*, can derive the value to be stored in a field and can restrict the value that will appear in a field. These expressions represent a subset of a larger set of logical expressions that complete the representation of the semantics of the application, and are termed "business rules" by practitioners.

The ITOC tool analyses assignment statements such as "*A := B + C*". It also analyses SQL "Select" statements such as "*Select A + B, from T where C > D and (I = J or K = L)*" and validation expressions such as "*Minimum_Amt > Ordered_Amt*". After analysis, the tool produces a *business rule report*,

The report contains a hyper-text (HTML) listing of each frame's source code, displaying business rules (in italics) and references to variables that are connected to either fields or query columns (in bold), which we observed indicated the existence of a business rule. The use of the report reduces the likelihood of these rules being overlooked during the manual re-implemention phase of the re-engineering process. The practitioners who completed the development of the target application after performing design recovery using the ITOC tool found the report useful but had hoped for more extensive support for business rule recovery than provided by the tool.

4.2.3 Create, Read, Update and Delete Operations

Operations for creating, reading, updating and deleting database tuples, which are collectively referred to as *CRUD* by practioners, are the four possible types of operations that can be defined in a CASE Module Data Diagram. This information is recovered from the source application by analysing SQL statements in the application, and is stored in the CASE repository. This allows the CASE generators to produce target applications which support the same table operation modes as the source application. This information is both explicit and declaratively represented in the 4GL code, hence only rudamentary information system semantic analysis is required for full recovery.

4.2.4 Metric Report

The ITOC tool contains a utility for estimating the effort required to complete re-engineering of an application after design recovery using the ITOC tool is complete. The report is based on heuristics provided by the practioners that reason with the

number of *activations* appearing in a frame. Activations, which are also referred to as *triggers*, are event-activated procedural code segments appearing in a 4GL application. The heuristics also reason with the number of activations associated with each field and the number of occurances of a program variable in non-trivial expression. The practioners indicated that the report proved useful. A more detailed description of this report appears in (Tech 1996).

5. Related Work

This section provides a summary of research that relates closely to the work described here. It is loosely classified into that which only attempts to analyse data, and that which also attempts to migrate application code.

(Petit et al., 1994) proposed a design recovery method to extract an entity-relationship (EER) schema from a relational database extension and SQL queries. Like ITOC, the method does not require that the same attribute in different relations have the same name nor that the relations are in 3NF. (Andresson, 1994) proposed a similar approach. However, like much of the related work, no indication is provided as to whether these methods were actually implemented.

(Chiang et al., 1994) proposed a method to extract an EER model from the database extension. It assumes that the database is in 3NF, that there is consistent naming of attributes, that there are no erroneous data values and that inclusion dependencies are available. A prototype implementation is described but no results of utilising the prototype are provided.

(Signore et al., 1994) describe an expert system prototype for rebuilding a conceptual schema from a logical schema. Unlike ITOC, the approach does not utilise the database extension, so it does not verify the results on the inferencing process.

(Rugaber and Doddapaneni, 1993) attempted to automate the extraction of and SQL schema from a COBOL program that used flat ISAM files. Like ITOC, the recovered database design information was loaded into a CASE tool prior to forward engineering into SQL. Unlike ITOC, however, the structure of the *application programs* are not recovered.

Other researchers, such as (Navathe and Awong, 1988), (Markowitz and Makowski, 1990), (Winans and Davis, 1990), (Fong and Ho, 1993), (Premerlani and Blaha, 1994), (Campbell and Halpin, 1996), and (Lim and Harrison, 1996), have described similar approaches for extracting conceptual models from either relational, hierarchical or network data models.

The next class of related work we review attempts to migrate the application programs as well as just the static schema.

(Zoufaly et al, 1995) describe an emulation approach for migrating RPG to Oracle PL/SQL and Forms 3. Low-level transformation, implemented using their own high-level language, Logistica are used to directly translate RPG statements to Forms 3. Although an implementation is reported, the target system's representation is at the same low level of abstraction as the RPG source which does not facilitate understanding and maintenance.

COBOL/SRE (Engberts, et al., 1993) is described as a "renovation and re-engineering" environment. The environment produces reports and support

sophisticated browsing and enhancement to an existing COBOL system. Unlike ITOC, this tool only supports forward engineering back into COBOL and so does not raise the level of abstraction. No results were presented as to the effectiveness of the environment. (Karakostas, 1992) describes a similar system.

(Vermeer and Apers, 1995) described an approach for recovering design information from 3GL application programs based on the observation that "C" data structures that store the result of SQL queries can serve as object schemata. No indication is given that a prototype has been constructed to evaluate the feasibility of the approach.

(Hainaut et al., 1995), proposed a design recovery methodology and generic architecture for performing the extraction of an enhanced ER schema from a database followed by the conceptualisation of the resulting data structures.

The ITOC tool described in this paper is novel in that it recovers the structure of 4GL applications at a high level so that new application programs can be generated using a CASE tool. It is also unusual because it has been applied to commercial legacy information systems and the results evaluated by experienced information system developers.

6. Conclusion

In this paper, we have outlined the functionality of the ITOC tool and described the results obtained from applying the tool to commercial deployed legacy information systems. The experiment conducted indicates that the approach can be successfully used to recover certain fundamental design components, such as module data diagrams that capture the semantics of the information system application, and also database constraints that are not explicit in the data definition language. The experimentation revealed that "industrial-strength" tools can be constructed using the approach as both a contemporary commercial CASE repository product and (deployed) commercial information systems implemented using a contemporary, commercial 4GL were utilised.

As we expected, there were some design constructs that the tool should have recovered from the source application but failed to do so. However, it is likely that these deficiences can be corrected and do not represent a fundamental flaw in the approach. Consequently, we encourage other researchers to utilise it.

Our experience indicates that it is essential to apply software re-engineering tools on deployed, commercial applications to demonstrate their true effectiveness. It is also beneficial to have the output of the tools reviewed by experienced practitioners to gain useful feedback for tool improvement. From this feedback, we learned that the ITOC approach must be improved in the area of business rule recovery.

Ongoing research involves the monitoring of practitioners who are utilising the ITOC tool to determine what proportion of the total re-engineering effort is eliminated due to its use. We are also investigating metrics that can be used to estimate this effort, which will be extracted automatically from a source application via a utility that includes the source 4GL language model. Finally, we also plan to study the end-user's acceptance of the re-engineered systems produced using the ITOC tool and the costs of retraining these users.

Acknowledgements: The authors wish to recognize the contributions of Professor Paul Bailes, Dr. Anthony Berglas and Mr. Ian Peake (CSM), and also Mr. Blair Layton and Mr. Ronald Bradford (Oracle Corporation).

References

Andersson, M., Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering. *Proc. of the 13th Entity-Relationship Conference,* Manchester, UK, Dec. 1994, pp. 403-419.

Berglas A. and Harrison, J.V., Evaluation of the ITOC Information System Design Recovery Tool, *Proc. of Fifth International Workshop on Program Comprehension,* Michigan, March 1997, pp 176-182.

Blaha, M.R. and Premerlani, W. J., Observed Idiosyncracies of Relational Database Designs, *Proc. of Second Working Conference on Reverse Engineering,* Toronto, Ontario, July 1995, pp. 116-125.

Callahan, D., The Program Summary Graph and Flow-sensitive Interprocedural Data Flow Analysis, *Proc. of the SIGPLAN'88, Conference on Programming Language Design and Implementation*, Atlanta, Georgia, June 1998, pp. 22-24.

Campbell, L.J. and Halpin, T.A., The Reverse Engineering of Relational Databases. *Proceedings of the 5th Workshop on the Next Generation of CASE Tools*, Utrecht, June 1994, pp 50-66.

Chiang, H.L., Barron, Terrence M., and Storey, V.C., Reverse engineering of relational databases: Extraction of an EER model from a relational database. *Data & Knowledge Engineering* 12 (1994), pp. 107-142.

Elmasri R. and Navathe S.B., Fundamentals of Database Systems. 2nd Ed. The Benjamin/Cummings Publishing Comp, Inc. 1994.

Engberts, Andre, Kozaczynski, Liongosari, Edy and Ning, J.Q., COBOL/SRE: A COBOL System Renovation Environment, *Proc. of the 6th Intl. Workshop for Computer-Aided Software Engineering, Ed. H. Lee and Thonas Reid*, Singapore, July 1993, pp. 199-210.

Fong, J. and Ho, M., Knowledge-based approach for abstracting hierarchical and network schema semantics. *Proc. of the 12th Entity Relational Approach*, Arlington, Texas, Dec. 1993, pp. 508-519.

Harrison, J.V., Bailes P.A., Berglas A., Peak I., Re-engineering 4GL-based Information System Applications. *Proc. of the Asia Pacific Software Engineering Conference*, Brisbane, Dec. 1995, pp. 448-457.

Harrison, J.V. and Berglas A., Data Flow Analysis with the ITOC Information System Design Recovery Tool, *Proc. of Automated Software Engineering Conference*, Incline Village, Nevada, USA, Nov. 1997, pp. .

Harrison, J.V., Bailes P.A., Berglas A., Peak I., Legacy 4GL Application Migration Via Knowledge-Based Software Engineering Technology: A Case Study , *Proc. of Australian Software Engineering Conference*, Sydney, Oct. 1997, pp. 70-78.

Hainaut, J.L., Englebert, V., Henrard, J., Hick, J.M., Roland, D., Requirements for Information System Reverse Engineering Support, *Proc. of the 2nd Working Conference on Reverse Engineering*, Toronto, Ontario, Canada, July 1995, pp 136-145.

Lim, W.M. and Harrison, J.V., An Integrated Database Re-engineering Architecture - A Generic Approach, *Proc. of Australian Software Engineering Conference*, Melbourne, Aug. 1996, pp. 146-154.

Karakostas, V., Intelligent Search and Acquisition of Business Knowledge from Program. *Software Maintenance: Research and Practice*, Vol. 4, (1992), pp. 1-17.

Markowitz, V.M. and Makowsky, J. A., Identifying Extended Entity-Relationship Object Structures in Relational Schemas. *IEEE Transactions on Software Engineering.* Vol. 16, No. 8. Aug. 1990, pp. 777-790.

Marlowe, T.J., and Ryder, B.G., 1990. An efficient hybrid algorithm for incremental data flow analysis. In *Conf. Recored of the 17th Annual ACM Symp. On Principles of Programming Languages* (San Francisco, Jan.), ACM Press, pp. 184-196.

Navathe, S.B. and Awong, A.M., Abstracting Relational And Hierarchical Data With A Semantic Data Model. *Proceedings of 8th Entity Relational Approach: A bridge to the future*, New York, Nov. 1988, pp.305-333.

Peak, I.., User's Guide to the Language Extension Work Bench Version 1, Technical Report 387, Centre for Software Maintenance, Department of Computer Science, University of Queensland, Australia, March 1996.

Petit, J.M., Toumani, F., Boulicaut, J.F., Kouloumdjian, J., Using Queries to Improve Database Reverse Engineering. *Proc. of the 13th Intl. Conf. on Entity Relational Approach,* Manchester, Springer-Verlag, 1994, pp. 369-386.

Premerlani, W.J. and Blaha, M.R., An Approach for Reverse Engineering of Relational Databases, *Communication of the ACM*, Vol.37, No.5 May 1994, pp. 42-49.

Rugaber, S. and Doddapaneni, S., The Transition of Application Programs from COBOL to a Fourth Generation Language, In *Proc. of Intl. Conference on Software Maintenance*, 1993, pp. 61-70.

Ryder, B.G., Landi, W., and Pande, H. Profiling an incremental data flow analysis algorithm. IEEE Trans. Software Eng., 16, 2: 1990, pp. 129-140.

Signore, Oreste, Loffredo, Mario, Gregori, M., and Cima Marco, Reconstruction of ER Schema from Database Applications: a Cognitive Approach, *Proc. of the 13th Intl. Conf. on Entity Relational Approach,* Manchaster, Springer-Verlag, 1994, pp. 387-402.

Technical Report on the Ingres to Oracle Design Recovery Workbench, Department of Computer Science and Electrical Engineering, University of Queensland, 1996.

Vermeer, M.W.W. and Apers, P.M.G., Reverse engineering of relational database applications. *Proc. of the 14th International Conference on Object-Oriented Entity Relationship Modelling,* Gold Coast, Australia, Dec. 1995, pp. 89-100.

Winans, J., and Davis K.H., Software Reverse Engineering from a Currently Existing IMS Database to an Entity-Relationship Model, *Proceedings of 9th Entity Relationship Approach,* Lausanne, Switzerland, Oct. 1990, pp. 333-348.

Zoufaly, Federico, Araya, Carlos, Sanabria I. and Bendeck, F., RESCUE: Legacy System Translator, In *Proc. of the Second Working Conference on Reverse Engineering*, Toronto, Ontario, July 1995, pp 39-50.