Evaluation of the ITOC Information System Design Recovery Tool

Anthony Berglas, John Harrison*  
Centre for Software Maintenance  
The University of Queensland  
Brisbane 4072, Australia  
{Berglas, Harrison}@it.uq.edu.au

Abstract

Design recovery from legacy information systems is an industrially relevant problem which poses significant research challenges. However, although many approaches have been proposed, few of these have been implemented beyond a prototype stage which makes it difficult to evaluate their effectiveness.

The Ingres to Oracle Conversion (ITOC) project is a large collaborative research initiative between The University of Queensland and Oracle Corporation to design and implement a tool that automatically recovers both the application structure and the static schema definition of Ingres 4GL applications. The results of this analysis are then loaded into Oracle’s CASE tool.

This paper describes the basic techniques used by the ITOC tool which includes data flow analysis, query analysis and simple data mining. The effectiveness of these techniques when applied to real commercial applications is then evaluated in detail

Keywords: Reengineering, Design Recovery, Information Systems, 4GL Applications

2 Introduction

A large proportion of all program code in use today implements information systems created using fourth-generation programming languages. There have been considerable advances in the design of these languages and the tools that support development using them. Unfortunately, 4GLs are usually very vendor specific languages which often prevents organizations from realizing the benefits of newer technologies without incurring a very large redevelopment cost.

The Ingres to Oracle Conversion (ITOC) tool has been built collaboratively by The University of Queensland and Oracle Corporation. ITOC automatically recovers both the application structure and the static schema definition of Ingres 4GL applications, the results of which are then loaded into Oracle’s CASE tool.

Unlike conventional conversion tools, ITOC performs design recovery which involves a deep analysis of the queries and structures within source 4GL programs so that a succinct description of potentially large amounts of 4GL code can be produced. This approach has been chosen because the alternative direct conversion approaches are either technically infeasible or produce code that is difficult to maintain (Harrison et al., 1995). Design recovery of information systems is feasible because a large proportion of their code provides similar types of functionality such as retrieving data and providing effective user interfaces.

To the best of our knowledge this is the first 4GL design recovery system that has ever been implemented beyond a prototype stage. This is important because as (Blaha and Premerlani, 1995) point out, most existing techniques fail to be effective when confronted with the idiosyncrasies that occur in real applications. Many unexpected results were obtained which required significant enhancements to be made to the tool. The results of the application of the ITOC tool have been assessed by experienced Ingres and Oracle developers. Although researchers that develop reengineering tools may interpret the output of the tools as commercially useful, our experience indicates that the most accurate assessment is provided by the practitioner that actually uses the tool to convert applications. In particular, the practitioners pointed out that recovering an application’s structure was at least as important as just recovering the schema because both the source and target schemas were already specified in reasonably standard SQL.

The paper describes each step in the analysis performed by ITOC in some detail. This includes data flow analysis, query analysis, form analysis, and simple data mining. Many of these techniques could be used to build other information system design recovery tools. It then evaluates the effectiveness of these approaches when applied to real applications. The ITOC tool is also compared with related reengineering projects.

*This work has been sponsored by Oracle Corporation
3 The Design Recovery of Information Systems

4GL-based information systems are comprised of a user interface, application logic and a relational database management system (RDBMS). 4GLs model user interfaces as forms such as the one shown in Figure 1 which contains a number of fields that relate directly to columns in the database. The 4GL can then perform the administrative processing that is required to track the end user as they move from field to field. Fragments of procedural 4GL code are then invoked when significant events occur such as changing the value of a field that requires complex validation.

** Company XYZ's Order Entry **

| OrderNr:    | __________ |
| Billing Customer: | ____________________|
| Shipping Customer: | ____________________|
| Item Nr | Item Name | Qty | Unit Price | Total |
| ___ | _________ | ___ | ________ | ___ |
| ___ | _________ | ___ | ________ | ___ |
| ___ | _________ | ___ | ________ | ___ |
| ___ | _________ | ___ | ________ | ___ |

** Company XYZ's Order Entry **

Figure 1: Sample Order Entry Form

CASE tools can further optimize development by also automating the code that is required to move data between the form and the underlying database. Figure 2 is an Oracle CASE “module structure” diagram that describes most of the functionality required for the order entry form.

The boxes represent “Module Detail Table Usages” (MDTUs), that provide a subschema that describes how various tables are being utilized by the interface. Oracle CASE uses the layout of the boxes to indicate how the tables are used. Thus the fact that the Order box is above the Item box indicates that they form a “master/detail” relationship, while the fact that Product is to the right of Item indicates that the Product Name is a “look up” (retrieved) and is not directly enterable in this form. This example contains two MDTUs that manipulate the Customer table. Recursive structures can also be defined if necessary.

This simple description is all that need be specified to enable the CASE tool to automatically generated the code required to select Order and Item rows from the database, look up and validate their Customer and Product Numbers, and update the database in response to user input.

The ITOC tool examines the queries within the source application in order to determine the relationship between tables both in the schema and in the way that they are used within the source language forms. This needs to be performed within the context of the procedural environment in which they are enmeshed. Data flow analysis is used to determine how fields in the form are related to columns in the table, and then module analysis determines how several source forms may be used to build one target module. The next section describes the architecture of the tool together with an overview of the analysis that is performed at each stage.

4 Architecture of the ITOC Tool

The ITOC tool recovers the design of Ingres 4GL programs in a number of distinct steps as indicated in Figure 3.

Each step produces a number of different types of objects that are stored in a subschema and then used as inputs to the next step, with very little control flow occurring between each step. The processing continues until a file is produced that can be loaded into the CASE tool to define the schema and modules that are to be used to implement the finished application. This section will now provide an overview of each step which is followed by an evaluation of their effectiveness. Space limitations will require many details to be elided.

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 **Refine** programming language and **Dialect** compiler compiler. Unlike many other compiler compilers, Dialect automates the production of an abstract syntax tree as well as simply parsing the source file given a formal grammar of the source language. The Abstract Syntax Tree (**AST**) consists of a number 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.

The AST is a large and complex tree structure that reflects the sequence in which the code was written. This is flattened into a high level abstraction which is more suitable for further analysis. Subschemas of this abstraction have been shown in Figure 3.

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 it is a common Ingres coding practice to write 3GL programs that performed 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 as a subtype of the Query-Table class, and then treated exactly the same as if they were SQL queries for subsequent analysis despite the fact that they have a radically different syntactic structure.

Data flow analysis is used to link different query tables together even though they might be in different queries and to link fields in each frame to query columns. This is done by assuming that if a there is a data flow between two variables, then they will have essentially the same value. This assumption appears to be largely valid in information systems.

Direct data flows arise from assignment statements, parameters to procedure calls, and queries that assign values to variables or update the database based on the values of variables. The transitive closure of these is taken in order to model more complex forms, although transitive data flows are given less weight than direct ones when conflicts arise.

The joins between query tables are used to infer the relationships between MDTUs in a subschema such the one shown in Figure 2. For example, the order entry program could easily contain the following two queries that fetch the Item details and Product Names into the scrolling region:

1. Select :Product-Nr = Product-Nr
   From Items
   Where Order-Nr = :Order-Nr
...

1. Select :Product-Name = Name
   From Product
   Where Product-Nr = :Product-Nr

The second query would occur in event activated code that is invoked once for each row in the scrolling region.
The ITOC tool can infer the implied equijoin between the two query tables because of the data flow between the Items.Product-Nr and Product.Product-Nr query columns via the Product-Nr field. The direction of the relationship can be inferred from knowledge of the keys of the tables. In the example, Product.Product-Nr is a key, but Items.Product-Nr is not. MDTUs actually correspond to sets of query tables because applications often have several query tables that perform essentially the same function.

Foreign keys are used to define the relationship between tables in a RDBMS. The Ingres RDBMS does not model these explicitly in the DDL, so ITOC recovers their implicit definitions by assuming that equijoins between query tables have an underlying foreign key. However, due to the heuristic nature of the analysis, it was found that it was important to validate these foreign keys against a database extension. If a significant number of records violated the foreign key, then it was considered to be invalid and not processed further. Each relationship between MDTUs must correspond to a validated foreign key, so pruning invalid ones also improved the accuracy of the user interface definitions.

Other structural issues can also be best resolved by examination of an application’s data. ITOC performed queries to determine the following additional information:-

- Whether columns are mandatory or optional. Although Ingres contains Null and Not Null DDL definitions, their meaning is subtly different from Target’s and so the Ingres DDL cannot be used to determine this directly.
- Specific ranges for column values, for example an Order’s status might be restricted to “QUOTE”, “ORDERED”, or “PAID”.
- Whether foreign keys participate in a one to one relationship.
- Whether numeric fields may be negative.

Having performed this simple data mining, ITOC can convert source DDL into target DDL. The target DDL will include all important foreign keys together with simple domain constraints that were not present in the source DDL.

While producing an accurate schema is of major benefit, ITOC’s most important product is the mapping of source forms into MDTUs. This requires fields to be related to the sets of query tables which is performed by analysing the data flows between each field and the columns that appear in the individual queries.

Each field may only relate to one column usage in one MDTU, but in practice it was observed that many fields have data flows that relate them to more than one column in the database. This required a conflict resolution strategy to be developed that determines which competing set of data flows is the strongest and thus uniquely relates a field to a query table set. The conflict resolution strategy calculates the total strength of all the data flows within each competing set, and then allocates the field to the query table set with the largest total. The strength of each data flow is dependent upon the type of flow and the number of transitive links that are involved. However, it was also found to be necessary to include several additional factors into this resolution strategy so that it performs a holistic analysis of how each field interacts with other fields in each form.

Due to limitations in the source 4GL, it is a common code practice to produce several source forms that should be combined into one target module. For example, there might be one source form to select a master record, a second to view a master record with numerous details, and a third that is “popped up” to enable detail records to be changed. Alternatively, there might be one form that allows new records to be created, but a different one that enables them to be updated. The final stage of the analysis removes many of these redundant forms by examining the data flows in the parameters that passed between them.

5 Evaluation of the approach

Building practical reengineering tools is expensive so the availability of good tools that reduce this expense is important. It was found that almost a half of the total time required to build ITOC was consumed in producing an accurate language model of the source 4GL, linking references to definitions, and performing the AST abstraction. Dialect and the LXWB provided significant automation in this phase although the source language contained several unusual syntactic constructs which Dialect’s LALR(1) compiler compiler could not process without significant manual intervention.

The use of an explicit AST abstraction schema was found to be very effective in producing this design recovery tool because it enabled the subsequent complex analysis to be defined in terms of relatively simple objects with well defined relationships to each other that are independent of any complex surface syntax.

It is interesting to note that a tool that performed direct conversion rather than design recovery would not need to perform this step because the conversion would be performed by examining each statement in isolation which could be performed directly on the AST. Thus the presence of an AST abstraction is one of the distinguishing features of a design recovery tool.
The data flow analysis was performed in a very naive manner, and this proved to be largely adequate for the applications that were processed. This is because the structure of 4GL programs is relatively simple, so complex flows were rarely encountered.

However, the ITOC tool treats flows between actual and formal parameters as if they were simple assignment statements between variables, and does not take into account the fact that a procedure may be invoked multiple times. This means that invalid data flows can be created between the actual parameters of each invocation of the procedure. While there are several ways that this problem could be overcome, it did not arise in carefully constructed test cases and so was not discovered until after the tool was applied to real applications.

It has been discovered that when this heuristic analysis does produce errors they tend not to be serious when allocating field to MDTU columns because the use of weighted strengths tends almost always enable valid data flows to overpower the invalid ones. Data flows are also used to determine the implicit joins between query tables, but again invalid ones are usually eliminated by the need to make foreign keys reference keys, and by the data mining that is performed before any join is accepted as being valid.

The query analysis is similar to independent work by (Andersson 1994). However, no attempt has been made to infer the keys of tables by examination of the queries because keys are almost always indicated by the presence of unique indexes in the source application. Further, part of Anderson's analysis involved inferring which table in a join is not referenced by a key and then assuming that the other table is a key. It would seem that this would be very susceptible to invalid data flows and other noise resulting from the heuristic nature of the analysis.

While the data mining definitely enhances the results of the code analysis, algorithms exist that enable foreign keys to be determined from the data alone without any reference to the code. However, there are a number of benefits of synergistically combining analysis of code and data.

- While modern data mining algorithms can find foreign keys in polynomial time, this is still a very expensive operation on large databases. Using the queries in an application enables foreign key mining to be performed much more efficiently.
- More importantly, results found by data mining alone may be either coincidental or temporary rather than business rules. For example, it would not be surprising to find that the combination of a Product's Price, Weight and Quantity-On-Hand uniquely identifies each row in a large Product table and thus forms a “key”, even though this is nonsense semantically. Similar effects occur for other business rules. For example, if a column contained a range of values up to but not exceeding 96, it would be tempting to assume that this was a maximum value for the column. However, it column could simply reflect the current year, and any implied business rule would be invalidated the following year. If the application contained a query that involved just the columns a potential key or a code fragment such as “If col > 96...” then one could have much more confidence in the inferred business rules.
- More fundamentally, code analysis is required to determine the structure of the user interfaces, which is a primary objective of ITOC.

The form analysis proved to be effective for most fields with the notable exception of fields that represent the keys to tables. This is because there are often multiple sets of query tables that use the same fields for a key. In the example in Figure 1, the Order Number field is the key to the Order table, but all the Item records will also have that same Order Number. Further, non-key fields are usually selected from the database and then updated which produces many strong flows to their corresponding query table set columns. However, key fields are often input directly from the user and the keys of existing records may usually not be changed. This means that key fields are usually only used in the where clauses of select statements which only produces relatively weak data flows. An analysis of numerous forms was required to tune the form analysis procedures so that they were sufficiently discriminating.

The combination of multiple similar forms into a single module was effective in preventing numerous redundant target modules from being created. However, several unexpected cases were found in practice and future work could enhance this aspect of the tool.

7 Related Work

Existing work on migrating legacy information systems can be 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. (Andersson, 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) conducted 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), (Edwards and Munro, 1995), and (Lim and Harrison, 1996), have described similar approaches for extracting conceptual models from either relational, hierarchical or network data models.

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

(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 reengineering” 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.

(Verreer 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.

8 Conclusion

A design recovery tool such as ITOC must necessarily utilize heuristics in order to produce an abstract target representation of complex source code. However, it is very difficult to predict the effectiveness these heuristics in a complex tool without actually applying them to real applications.

While some of the analysis is very specific to the Ingres 4GL, much of the core technology of data flow analysis and query analysis could be used for other 4GL and 3GL languages. For example, much of this technology could be reused to build a Cobol design recovery system because many information system programs utilize a transaction processing environment such as IBM’s CICS. These environments impose similar constraints to the way programs are written which could be exploited by a design recovery tool.

The following considerations need to be taken into account before attempting to build design recovery tool:-

- Substantial reengineering tools are expensive. Over 200 person days have already been expended on ITOC, and a significant amount of refinement is still required. Thus it appears that construction of such tools would only be commercially viable if there is a significant body of source code that requires conversion.

- A good model is required to be able to provide high level descriptions of the code. ITOC is successful because information systems can be described very succinctly in terms of MDTUs.

- A design recovery approach will always be incomplete, so a significant amount of manual work may still be required. However, experience with applications that have been recovered using ITOC
suggests that significant automation can be obtained.

The next phase of development will involve a full scale deployment of ITOC to many more Ingres 4GL applications world wide. This will provide substantial data upon how effective the tool is in practice. Other future work includes the development of better language technology support for parser generation. Building a tool with a different source language could also be investigated to determine the extent to which the analysis techniques presented in this paper could be reused.

This paper has provided a step by step overview of how the ITOC tool performs design recovery and the effectiveness of each step. Although the ITOC tool was developed by people familiar with both the source and target languages, many insights were only gained once the tool was applied to real applications. This has enabled basic research to be performed that develops advanced techniques to solving practical problems. It has also enabled the effects of those techniques to be properly evaluated, as well as evaluating the design recovery process in general.

9 Acknowledgments

The authors would like to thank Blair Layton and Linda Campbell for their contribution to this work.

References


