Abstract

August-II is a data reverse engineering tool whose input can be from a variety of sources from COBOL record layouts to DB2 data definitions. The translation that is done is designed to produce a conceptual data model to that allows the user to understand the current environment and to move to new data technology. August-II produces output that can be used as input to many different software packages such as CASE tools and commercial database management systems. The power behind August-II is two-fold: 1) the "layered data architecture" of its information dictionary system; and 2) the step-by-step reverse engineering process. The layered data architecture allows August-II the capability of representing the wide range of data environments in use today. The step by step takes the database designer through the reverse engineering process in such a way that the user can concentrate on one aspect of the process at a time and not to get bogged down in the process itself.

Introduction

Data reverse-engineering is a relatively new area of research that is deals with the problem of understanding existing systems and recovering the detailed information requirements embodied within them. The reverse-engineered data model is a representation of the current data environment. These models are pictures of "where the system is now". They offer a general understanding of current data structures and applications. They can assist in standardizing the definitions of entities and attributes and can be used to evaluate how well new technology will fit into the environment.

In [DAVI94] we described the way that August-II a data reverse engineering tool implements the flexible conceptual data model and the how it uses data model to conceptual data model translations in the reverse engineering process. In this paper, we will describe the layered data architecture of the August-II data dictionary and its use in the step-by-step reverse engineering process of August-II. First, we give a short history of August-II. Second, an overview of the flexible conceptual data model of August-II is given along with how it differs from current research since this is needed to understand August-II. We do not cover the exact reverse engineering algorithm of the relational data model here as that is not the focus of this paper. The next section describes the main aspects of this paper--the layered data architecture of August-II's dictionary and the individual components of August-II's step-by-step process. Lastly, we give a brief overview of the tests that have been performed using August-II.
**A Short History**

The work on August-II began in 1980 by studying the reverse translation of conventional file systems, specifically COBOL File Descriptions. This tool was called August, was published in [DAVI85] and differed from earlier research in that the COBOL File Descriptions were not assumed to be normalized. In fact the user was queried about any references between files so no assumptions at all were made about naming conventions, or even that a reference from one file to a second was made through the primary (or candidate) key of the second. The August software tool was written in Prolog on a PC and was extremely slow in processing the input data.

The next step in developing August-II was to see if we could improve the process. We rewrote the August software using Texas Instruments expert system tool called M1 (supported by a grant from the Department of Commerce and Community Affairs of the State of Illinois). We abandoned M1 when we learned that it would not allow us to do certain things--specifically was to "save-the-state". One of our main objectives for August was to allow the user to be able to turn off the machine anytime during the reverse engineering process and then pick up from this point later at their discretion. At that time, M1 did not allow us to save-the-state.

During this same time period we were studying the reverse engineering algorithms for various input data environments other than COBOL. Specifically, we looked at relational database to entity-relationship (ER) [DAVI87] and at IMS to ER [WINA90]. In the process of studying the various algorithms we fully developed August-II's flexible conceptual data model [DAVI90] and designed how we were going to implement this concept in the tool [DAVI94]. Since we knew that we wanted heterogeneous data environments as input to our reverse engineering tool, we created the "layered data architecture".

Currently, the newest implementation of August-II's user interface is written in Borland C++ 4.0, the translations, etc., are being written in Borland C++ 3.1, and the data dictionary is in Paradox for Windows.

**Flexible Conceptual Data Models in August-II**

Generally, data models have been defined as 1) the structure of the data and 2) the operators allowed to perform on the data. For example, it is the specification of the allowable operators that would change the data structure of a 'list of objects' into either a stack, COBOL record, or queue. Data models also include constraints placed upon the allowable operations. These constraints are restrictions on the permitted operations that prohibit the existence of certain data structures. The insertion and deletion processing constraints are called "behavior" of the data model. In [DAVI90] we proposed an enrichment of these basic definitions that is used in August-II in the Information Dictionary and within the reverse translation algorithm.

1: Data Object= [Data Structure, Operators]
2: Data Model= [Data Object, Inherent Behavior]
3: Conceptual Model= [Data Model, Explicit Behavior]

First, the structure of the data is combined with allowable operators into a concept called a 'Data Object'. Data Objects and their 'Inherent Behavior' make up a Data Model. Inherent Behavior is the collection of all inherent constraints. This differs from the definition of data models in [TSIC82], only in that we have made the inherent constraints part of the Data Model rather than the idea that a data model can have inherent constraints associated with it. Conceptual models are defined as representations of the information content of a database without regard to the physical storage of the data. This definition is enhanced with the addition of 'Explicit Behavior' to a Data Model to produce the Conceptual Model. Therefore, a Flexible Conceptual Model not only includes the information content of a database but also the restrictions placed upon the information during insertion and deletion operations.

**Translation Algorithms.**

Since the main purpose of this paper is to show the August-II tool and not the reverse engineering algorithms it uses, we only present an overview of the reverse engineering data algorithm from a relational database as an example. A more complete description of these algorithms can be found in other publications cited. Currently, many algorithms to translate existing data environments into "conceptual models" have been presented in [CA5AB3], [DAVI83], [DAVI87], [WINA90], [NAVIA87], [NILS85], and [ANDE94].

In [FONK92] and [PREM94] a comparison is made of several reverse engineering algorithms that use the relational database as input along with descriptions of their own algorithm. They both suggest the need to consider hierarchical (generalization/specialization) relationships in the data and the concept of synonyms and homonyms of attributes. Synonyms and homonyms have always been used in August-II with the user being asked to supply the needed information. The determination of the subtype/supertype semantics has
always been considered in the translation of COBOL file definitions and is being added for relational database.

Essentially, there still are two types of translation algorithms. The first type of translation algorithm translates a Source Data Model (SDM) into a Destination Data Model (DDM) obtaining only the Data Objects and the Inherent Behavior in the DDM. This is called "Data Model-to-Data Model" translation (DM-to-DM).

A second translation algorithm shown in [DAVI90] translates a SDM into a Destination Conceptual Model (DCM) using the DCM’s Data Objects, Inherent Behavior, and Explicit Behavior. This type is called the "Data Model-to-Conceptual Model" translation (DM-to-CM). The algorithm used in the DM-to-CM translation example chooses the Explicit Behavior of the destination Entity-Relationship Conceptual Model (ERCM) to represent only the Inherent Behavior of the source Relational Data Model (RDM) that, when imposed upon a Schema in the ERCM, would cause a Schema restructuring simply for a behavioral change. The inherent constraints chosen are ones that have a high probability of needing modification by the database designer. Figure 1 shows the ERCM that would result from a translation in August-II for the following relational database:

Employee(E#, Name, Salary, Building#, Room#)
Employee-Project(E#, P#, Hours-spent)
Department-Project(D#, P#, Budget)
Job(J#, Description, Salary-Range)
Employee-Job(E#,J#)
Location(Building#, Room#, Description, Capacity)

To summarize, the DM-to-CM translation algorithm is used in August-II. The ERM, along with the Explicit Behavior making up the ERCM, represent all the properties of the current data environment needed by the database designer in order to fully understand the current relational DM. And the flexibility of the ERCM achieved is shown by the ability to change the role of either the DEPARTMENT or PROJECT entities by simply changing the Explicit Behavior.

Overview of August-II

August II is an intelligent tool to assist users in the data reverse engineering process of their current heterogeneous data environments. August II, contains
several components as shown in the architecture in Figure 2.

Current Data Environments

August-II receives input from any current environment and produces a conceptual model and information dictionary representing that environment. It is modularized to allow for easy adaptation to the various current data environments.

Layered Data Architecture

The Information Dictionary

The Information Dictionary in August-II is a repository about the current data environment as created from the Extractors as well as the conceptual data model that is the output from the Translator. Different data models are needed within the Information Dictionary in order to accurately describe each kind of input data environments as well as the flexible conceptual data model created by the Translator. Layered data architecture is used to facilitate the need for the different data models.

Layered data architecture as proposed by the Reference Model Task Group of the ANSI X3H4 committee, is used in the Information Resource Dictionary System (IRDS) [PRAB90]. The data architecture is modified from the IRDS to fit August-II's specific needs as shown in Figure 3.

August-II Dictionary Definition: Layer 1. As is the case with layered data architecture, each layer in August-II is used to describe the layer below it as shown in Figure 3. The top layer, "August-II Dictionary Definition" (Layer 1), defines the scope of the August-II dictionary. So far, we have determined that there are four elements to this "meta" layer. (Although this technology allows us to add more elements as seen fit.) The August-II Object-Type (AOT) defines the objects or "things" that are can be described in the Data Model Definition of the second layer. The August-II Relationship-Type (ART) describes the relationships the objects within the data model definition can have. The August-II Attribute-Type (AAT) defines the data that AOT and ART can use. And finally the August-II Behavior-Type (ABT) defines the capability of allowing explicit behavior within each of the data models used.

August-II Data Model Definition: Layer 2. The second layer, "August-II Data Model Definition (Layer 2)", is used to describe the data models that August-II is capable of recognizing. Each time a new current data environment is defined to August-II, a new data model definition is added to this layer. Figure 3 shows the data model definition for the ERCM.

August-II Application Schema Definition: Layer 3. The third layer of the August-II Information Dictionary is the Application Schema Definition (Layer 3). Layer 3 is used to describe the data within the current environment as well as the translated CDM resulting from the August-II Translator. To do this, different ADMD second layers are be used as definitions. The output of the August-II Extractors is an entry in this layer of the Information Dictionary. The current data environment is describes "as is" without any translations. Layer 3 for the current environment is then used as input to the August-II Translator which creates a flexible conceptual data model as another Layer 3.

The data layering architecture allows August-II the capability of having various types of current data environments as input. Each are described within the scope of the data model used in the environment so that an accurate representation is achieved.

Figure 2: General Architecture of August-II

August-II receives input from any current environment and produces a conceptual model and information dictionary representing that environment. It is modularized to allow for easy adaptation to the various current data environments.

Layered Data Architecture

The Information Dictionary.

The Information Dictionary in August-II is a repository about the current data environment as created from the Extractors as well as the conceptual data model that is the output from the Translator. Different data models are needed within the Information Dictionary in order to accurately describe each kind of input data environments as well as the flexible conceptual data model created by the Translator. Layered data architecture is used to facilitate the need for the different data models.

Layered data architecture as proposed by the Reference Model Task Group of the ANSI X3H4 committee, is used in the Information Resource Dictionary System (IRDS) [PRAB90]. The data architecture is modified from the IRDS to fit August-II's specific needs as shown in Figure 3.

August-II Dictionary Definition: Layer 1. As is the case with layered data architecture, each layer in August-II is used to describe the layer below it as shown in Figure 3. The top layer, "August-II Dictionary Definition" (Layer 1), defines the scope of the August-II dictionary. So far, we have determined that there are four elements to this "meta" layer. (Although this technology allows us to add more elements as seen fit.) The August-II Object-Type (AOT) defines the objects or "things" that are can be described in the Data Model Definition of the second layer. The August-II Relationship-Type (ART) describes the relationships the objects within the data model definition can have. The August-II Attribute-Type (AAT) defines the data that AOT and ART can use. And finally the August-II Behavior-Type (ABT) defines the capability of allowing explicit behavior within each of the data models used.

August-II Data Model Definition: Layer 2. The second layer, "August-II Data Model Definition (Layer 2)", is used to describe the data models that August-II is capable of recognizing. Each time a new current data environment is defined to August-II, a new data model definition is added to this layer. Figure 3 shows the data model definition for the ERCM.

August-II Application Schema Definition: Layer 3. The third layer of the August-II Information Dictionary is the Application Schema Definition (Layer 3). Layer 3 is used to describe the data within the current environment as well as the translated CDM resulting from the August-II Translator. To do this, different ADMD second layers are be used as definitions. The output of the August-II Extractors is an entry in this layer of the Information Dictionary. The current data environment is describes "as is" without any translations. Layer 3 for the current environment is then used as input to the August-II Translator which creates a flexible conceptual data model as another Layer 3.

The data layering architecture allows August-II the capability of having various types of current data environments as input. Each are described within the scope of the data model used in the environment so that an accurate representation is achieved.
Step-By-Step Reverse Engineering

By using a step-by-step process translations do not have to be performed as the information from the current data structures is extracted. The extraction and translation processes are kept as separate and distinct functions.

The first step in the reverse engineering is to use Extractors that extract information from the current heterogeneous database or file/record definitions. Next, the Translator is used to isolate the different data constructs found within the current data environment. Similar constructs, say aggregation, are represented differently within different data environments. Once aggregation is extracted from the current data environment, it is represented in the same way in the conceptual data model of August-II. August-II is designed to allow for a variety of data definition languages (DDLs) as input to the Extractors which generate an Application Schema Definition (Layer 3) in the August-II Information Dictionary for use as input to the Translator.

Step 1: The Extraction.

The Extractors take input from the current data environments. There is one Extractor for each type of data environment. For example, if the current data environment is a relational database system using ORACLE, then an Extractor exists that inputs the ORACLE relational schema, extracts the information, and outputs a data model in the August-II Information Dictionary format. Once an Extractor is built, it becomes a permanent part of August-II.

Figure 3 shows a generic relational database definition. Naturally, the exact data definition statement for the relational database management system is used as input to August-II.

The output of the August-II Extractor is a description of this relational database schema in Layer 3 of the Information Dictionary. Remember that the output of the Extractor is a Data Model and not a Conceptual Data Model, so there is no explicit behavior to represent in August-II. (This will change when the explicit behavior found in the application programs is added to the extraction process.)

The output of the August-II Extractor is Layer 3 of the Information Dictionary. For the example being studied the result is shown in Figure 5 implemented in a relational database. We have combined some August-II Relationship-Types into appropriate tables for a more efficient implementation. (e.g., the Attribute-has-Length relationship is shown within the Attribute Table). Every "object instance" that resides within the August-II Dictionary is given a unique system defined identifier. In the relational model, for example, the August-II Relationship-Type of Relation-has-Name is used to show...
the correspondence between the system assigned identifier and the relation name used in the current environment.

CREATE TABLE Employee
(E# CHAR(6) NOT NULL,
Name CHAR(30),
Salary DECIMAL(7,2),
Building# SMALLINT,
Room# SMALLINT,
PRIMARY KEY (E#));

CREATE TABLE Employee-Project
(E# CHAR(6) NOT NULL,
P# CHAR(6) NOT NULL,
Hours-Spent DECIMAL(4,2),
PRIMARY KEY (E#, P#));

CREATE TABLE Department-Project
(D# CHAR(4) NOT NULL,
P# CHAR(6) NOT NULL,
Budget DECIMAL(8,4),
PRIMARY KEY (D#, P#));

CREATE TABLE Job
(J# CHAR(6) NOT NULL,
Description CHAR(30),
Salary-Range DECIMAL(7,2),
PRIMARY KEY (J#));

CREATE TABLE Employee-Job
(E# CHAR(6) NOT NULL,
J# CHAR(6) NOT NULL,
PRIMARY KEY (E#, J#));

CREATE TABLE Location
(Building# SMALLINT NOT NULL,
Room# SMALLINT NOT NULL,
Description CHAR(30),
Capacity INTEGER,
PRIMARY KEY (Building#, Room#));

Figure 4: A Relational Database Definition as Input to the Extractor.

Step 2: The Translation.

The Translator receives the Information Dictionary System Layer 3 as created by the Extractor as input. The Translator determines the data constructs (structures) that reside in the current environment and translates them into the structure of the Entity-Relationship flexible conceptual model. The model is stored in the August-II Dictionary for use by the Merger or the Information Dictionary System. For example, in Figure 6, we show the Layer 3 result of the Translator of our example relational database (Figure 4) implemented as a relational database. ER-Entity, ER-Relationship, and ER-Attribute are all objects within the ERCM. The August-II Relationship-Types between these ER objects include (but not limited to) the following:

- Entity-has-Attribute
- Entity-has-Identifier
- Entity-involved-Relationship
- Entity-has-Behavior
- Relationship-has-Attribute

Note here that the Behavior-Type is used to show the explicit behavior determined during the translation process that is needed for the ERCM.

Step 3: The Merge.

The users of August-II may not wish to translate their entire current environment at any one point in time. For example, in the student information system, there are several applications each having their own set of files (e.g., admissions, registration, alumni, etc.) A user might decide to reverse engineer the admissions data separate from the others at first in order to see what is going on in that system. Later, the user may decide to add the translated registration system to the translated admissions system to see how these fit together. All the data to be merged must be previously translated and stored into the August-II dictionary within a Layer 3 Application Schema Definition. The Merger takes the specified ERCMs and combines them appropriately.

Step 4: The Formatting.

The Formatters access the August-II Information Dictionary conceptual data models (Layer 3) to produce output files that are used as input to data systems (probably a commercial database management system) as specified by the user. There is one Formatter for each type of system that needs input from August-II. As with the Extractors, once a Formatter is built, it becomes a permanent part of August-II and is available to all users.

Summary of Testing Results of August-II.

The basic algorithms of the reverse engineering process have been tested in several different mediums over the years. First, we tested the algorithm for COBOL to ER in Prolog. The basic algorithm worked but the processing was too slow. Next the relational SDM to ERCM translation algorithm of [DAVI87] was tested using Objective-C with a real live ORACLE database as input. The algorithm worked so well that the users of the ORACLE database were shocked to find out that they had two identical quintary relationships.
The IMS SDM to ERCM algorithm of [DAVI90] was written in Assembler Macro Language. We tested the reverse engineering on thirty-eight real IMS DBDs (database definitions). When we took the results to the users, they confirmed the accuracy of our algorithm and were very glad to finally have a handle on their current databases.

Recently, we have been rewriting the reverse engineering algorithms in Borland C++ 3.1. We have completed the recoding of the relational SDM to ERCM algorithm and are currently testing it on the ORACLE database. We are adding the handling of the subtype/supertype construct to this algorithm. We are starting on the IMS SDM to ERCM algorithm.

The user interface of August-II was never really user-friendly in the previous versions (even though this is a main objective of August-II). We have been concentrating on making the user-interface very simple and consistent. We are currently testing the interface for the process of translating a relational database to the ER model of August-II.

**Conclusion**

August-II is a data reverse engineering tool that is able to extract information from various current data environments. The utilization of the layered data architecture within the August-II Information Dictionary allows for the heterogeneous inputs as well as the flexible conceptual data model (ERCM) output used. The step-by-step process of the reverse engineering allows the user to concentrate on one aspect of the reverse engineering process.

---

Figure 5. Entries in the August-II Information Dictionary System for a Relational Database System.
<table>
<thead>
<tr>
<th>Entity Table</th>
<th>ER-Entity Identifier</th>
<th>ER-Entity has-Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENT0001</td>
<td>LOCATION</td>
<td></td>
</tr>
<tr>
<td>ENT0002</td>
<td>EMPLOYEE</td>
<td></td>
</tr>
<tr>
<td>ENT0003</td>
<td>JOB</td>
<td></td>
</tr>
<tr>
<td>ENT0004</td>
<td>PROJECT</td>
<td></td>
</tr>
<tr>
<td>ENT0005</td>
<td>DEPARTMENT</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ER Attribute Table</th>
<th>Attribute ID</th>
<th>Attribute Name</th>
<th>Attribute Length</th>
<th>Attribute Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERATT0001</td>
<td>#</td>
<td>Salary</td>
<td>8</td>
<td>99999999</td>
</tr>
<tr>
<td>ERATT0002</td>
<td>Name</td>
<td>30</td>
<td>Char</td>
<td></td>
</tr>
<tr>
<td>ERATT0003</td>
<td>Salary</td>
<td>8</td>
<td>99999999</td>
<td></td>
</tr>
<tr>
<td>ERATT0004</td>
<td>Building#</td>
<td>3</td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>ERATT0005</td>
<td>Room#</td>
<td>3</td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>ERATT0006</td>
<td>ID#</td>
<td>6</td>
<td>Char</td>
<td></td>
</tr>
<tr>
<td>ERATT0007</td>
<td>Hours-Spent</td>
<td>4</td>
<td>99999999</td>
<td></td>
</tr>
<tr>
<td>ERATT0008</td>
<td>ID#</td>
<td>4</td>
<td>Char</td>
<td></td>
</tr>
<tr>
<td>ERATT0009</td>
<td>Budget</td>
<td>8</td>
<td>99999999</td>
<td></td>
</tr>
<tr>
<td>ERATT0010</td>
<td>ID#</td>
<td>6</td>
<td>Char</td>
<td></td>
</tr>
<tr>
<td>ERATT0011</td>
<td>Description</td>
<td>30</td>
<td>Char</td>
<td></td>
</tr>
<tr>
<td>ERATT0012</td>
<td>Salary-Range</td>
<td>16</td>
<td>99999999</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Entity-has-Attributes Table</th>
<th>Entity Identifier</th>
<th>Attribute Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENT0001</td>
<td>ERATT0001</td>
<td></td>
</tr>
<tr>
<td>ENT0002</td>
<td>ERATT0002</td>
<td></td>
</tr>
<tr>
<td>ENT0003</td>
<td>ERATT0003</td>
<td></td>
</tr>
<tr>
<td>ENT0004</td>
<td>ERATT0004</td>
<td></td>
</tr>
<tr>
<td>ENT0005</td>
<td>ERATT0005</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ER Relationship-has-Behavior Table</th>
<th>Relationship Identifier</th>
<th>Relationship Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERREL0001</td>
<td>INSERT emp-loc IN E-L</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PRECOND emp in Employee</td>
<td></td>
</tr>
<tr>
<td></td>
<td>emp-loc [E#] = emp [E#]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DELETE emp FROM Employee</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IMPLIES { DELETE emp-loc FROM E-L</td>
<td>emp-loc [E#] = emp [E#]</td>
</tr>
<tr>
<td></td>
<td>CREATE emp-loc IN E-L</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CREATE emp-loc [E#] = emp [E#]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relationship Table</th>
<th>ER-Relationship Identifier</th>
<th>ER-Relationship-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERREL0001</td>
<td>EMPLOYEE-LOCATION</td>
<td></td>
</tr>
<tr>
<td>ERREL0002</td>
<td>EMPLOYEE-PROJECT</td>
<td></td>
</tr>
<tr>
<td>ERREL0003</td>
<td>EMPLOYEE-JOB</td>
<td></td>
</tr>
<tr>
<td>ERREL0004</td>
<td>DEPARTMENT-PROJECT</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6: The output of the Translator in an Entity-Relationship Conceptual Data Model
process at a time. The designer can stop and restart the translations at their discretion. By keeping the output of the Extraction step separate from the final ERCM, we benefit by allowing the user to see the current environment as is in all its "nastiness". We maintain the link between the current environment and the resulting ERCM. This will assist the user in understanding what is happening currently and why which will enable the user to move to new technology easier.

By keeping the output of the Translation step separate from the Merger step, we allow the user to look at "pieces of the whole" as well as the entire enterprise data. The user can see how each piece fits into the "big picture". As we know from Structured Analysis and Design, users can see and understand the smaller problems easier than a very large system.

References


