Data Documentation and Retrieval Using Unity in a UniVerse® Environment

Jose JIMENEZ

University of Iowa, Iowa City, Iowa

jose_jimenez73@hotmail.com

Introduction

Data storage, however well implemented, is worthless without the ability to retrieve that data. While the logical and physical connections may be available, if the data layout is not well documented, it may be near impossible to retrieve the data except through previously developed interface programs. These programs may be limited in their implementation based on static ideas of their use. In order to implement new interface methods, it is necessary to have complete documentation. It may be possible to use an automated approach to document the tables and fields of a database system This project will explore using Unity[1], and ODBC connectivity to document a large system of tables in a UniVerse®[2] environment. It will also explore using a set of host based programs to generate equivalent documentation, and the modification of Unity in order to produce RETRIEVE queries used by UniVerse® instead of standard SQL queries used by typical ODBC data sources. The remainder of this proposal will examine motivation for the project, the project architecture and associated details.

Motivation

In an ideal situation, a company that creates a piece of software would document it, and understand it before selling it. In the same ideal situation, a company that creates and sells a suite of software with hundreds of tables would understand and document the complex interactions of these programs and tables. Each field of every table would be documented and, hopefully with little effort, that company would be able to tell which programs required each field. However, not all programmers are software engineers, and not all companies that create software do it by using the appropriate methods. The type of documentation described takes time, and time is money, especially in environments where hourly rates charged to customers are above one hundred dollars and projects range in the hundreds of thousands of dollars. Given a limited budget a client company might choose additional functionality over complete documentation and therein lies the root of the problem. **Incomplete documentation when it comes to software systems of any magnitude is equivalent to no documentation.** Any additional changes to the software or database system must be researched heavily and the only testing that will suffice is full integration with the “live” suite.

Once a client company has made a decision for functionality over documentation, it is up to the client company programmers and analysts to take up the challenge of documenting the suite of tables and programs and to integrate them with other business tools. Unity[3] is a tool for documenting ODBC accessible data with X-Specs in a semi-automated manner. Some limitations exist in using an ODBC client to access the data necessary to create X-Specs for the UniVerse® environment. Thus it would be preferable to use host tools to create the X-Specs using programs on the host system which provide more information about relationships and data than would be accessible to an ODBC client. Once these X-Specs are created, Unity can be used to build data queries to access the data. By documenting tables more precisely using X-Specs, the complexities of future modifications to the system are reduced.

Details

This aim of this project is to evaluate the efficiency and efficacy of developing X-Specs using Unity verses creating the same X-Specs using a host based system of programs. To do this, a sample set of tables will be identified from those in use at a local company. Using Unity, these tables will be explored and mapped. Thereafter, host programs will be developed in order to create X-Specs using internally available data that is not available to the ODBC client. The resulting X-Specs will be compared in order to evaluate the different approaches. Additionally, Unity will be modified to create host specific queries. These queries will be aimed at accessing data in its native environment.

Environment

This project will entail examining the use of Unity and X-Specs in conjunction with a UniVerse® hosted database system. Unity is a Windows based tool for integrating multiple database types. In this situation the database system will be hosted on an RS6000 system with an AIX operating system. The client system, running Unity, will be a machine running a Windows 2000 professional operating system.

UniVerse® is a relational database environment from IBM with built in ODBC connectivity and its own programming language called UniBASIC®. Each database under UniVerse® is composed of a number of tables. These tables contain a collection of fields associated with records with unique identifiers. Each table has a dedicated table dictionary that defines the fields available for reporting. This table dictionary does not always contain an entry for each field in the table. An entry in the file dictionary contains the information required to retrieve and display the data from the field it pertains to in the table. This can include formatting instructions, conditional statements, and a label for the data among other things. One of the interesting aspects of this dictionary table is that it can contain “virtual” fields. These fields can be calculated data based on data internal to the table, calculated data independent of the data in the table, or based entirely on data from another table. In a UniVerse® environment, there is a special entry in the dictionary for each table that determines what entries, and therefore what data, is accessible using ODBC. This dictionary item is named “@select.” Figure 1 displays the architecture of this environment.

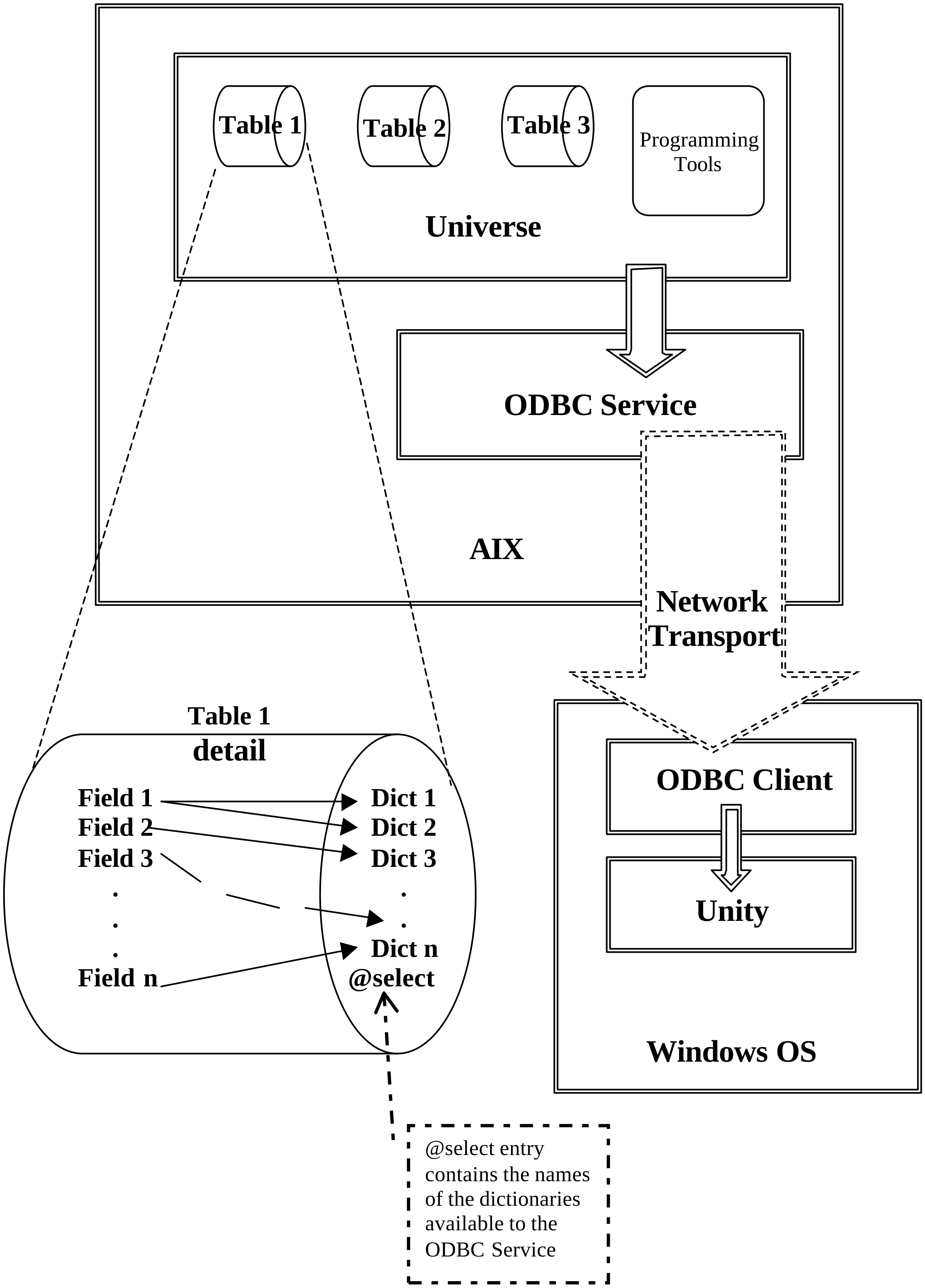

Figure 1: The UniVerse to Unity connection.

UniBASIC® has built in functionality for processing information stored in these databases and dictionaries and for developing user interfaces. There are also several procedural languages for automating data flow and user interfaces. UniVerse® also provides a powerful query language for generating reports. This query language is called RETRIEVE. These tools make the UniVerse® environment a powerful tool for building enterprise software.

In particular this project will evaluate the performance of two approaches of documenting data layouts in a business environment composed of over 500 data tables. Each of these data tables has from 0 to over 200 data fields. The number of virtual data fields is not limited in any significant way. In the first approach to documentation, the capture process of Unity will be used to build X-Specs for the data tables. The second approach will require new programs to be written in UniBASIC®. These programs will increase the information available to build the X-Specs. They will retrieve the additional data from the dictionary tables themselves.

Implementation

The first approach will be to set up X-Specs using existing ODBC connectivity. One drawback to using this approach includes the multi-valued nature of the data tables. Under UniVerse®, a single field can hold multiple values. These multiple values can be "related" to values in other fields. How this relationship is captured and exploited will need to be determined. If information about these relationships is not available to an ODBC client, Unity will not be able to present data about these fields correctly.

Another difficulty will be the selection of dictionary items to document. As stated earlier, each field can have multiple dictionary items. For example, a field that contains a company name can have a dictionary item that displays the field in its raw form. Another dictionary item for the same field can display the field in all capital letters. Yet another can display only the first 15 characters of the field. Each of these may be used within the UniVerse® environment for different purposes, such as a number of different reports. However, selecting from among them in order to make one or more of these fields available for ODBC access will not be a simple task. Simply selecting all the dictionary items may be acceptable where there are small numbers of virtual fields. However, selecting a large number of virtual fields may lead to complexity in documentation and querying and may introduce redundancy. There are other similar issues with selecting the appropriate dictionary items for ODBC access.

In a UniVerse® database environment, tables have dictionary items for each field that define the source of the data. These dictionary items are stored in a logical table that, under UniVerse®, can be inspected programmatically. UniVerse® provides its own programming language called UniBASIC®. The dictionary table that accompanies each data table can be read and an X-Spec generated for each field defined. As in Unity, this may not be a complete X-Spec and may require the user to complete the process. However, this method will give additional information not available to an ODBC client. This additional information can provide insight into relationships between fields, relationships between tables, and formatting of fields.

This project will be a first attempt to deploy Unity in a UniVerse® environment. The UniVerse® environment is one of a subset of database environments typically referred to as “multi-value” databases. Many of these environments have data table “dictionaries” that describe the data more completely than using ODBC. Further, since joins are defined in dictionary items under UniVerse®, mapping the query-building capability of Unity onto RETRIEVE will be a unique challenge.

Deliverables

This project will produce a report comparing the documentation of the sample data in the test environment using Unity with using host based programs for documentation. In addition, host programs and modifications to Unity will be produced as required. The host based programs to assist in creating X-Specs will also be completed. Additionally, Unity will be modified to create RETRIEVE query statements, or a version of Unity with this feature will be created.

During the first phase of the project, which should be completed within the first month of the project, sample tables will be selected from those available. These tables will be tested for ODBC connectivity and the fields currently available will be mapped using Unity. During this phase, the types of limitations inherent in accessing multi-valued information via ODBC will be documented. This information will be required for comparison with the host based retrieval method.

The second phase of the project will focus on minimizing the issues with ODBC connectivity found by the testing in phase one. During this phase it may be necessary to create a program on the host system to create “clean” or SQL compliant dictionary items for fields that currently are not compliant. This will increase the number of fields available to the ODBC client. This phase should be completed within the second month of the project. At this time it may also be possible to create programs to add additional fields to the @select record of each table. These programs would “inspect” dictionary items not in the @select dictionary and decide whether or not they should be placed there. This decision would be based, among other things, on whether the dictionary entry is for a field that already has an entry in the @select dictionary and the suitability of the dictionary item compared to others for the same field.

Work during the third phase of the project will repeat the mapping of the data tables using the tools present in Unity. The information recorded during this phase will also be used in the report for comparison against other results. This, being the second time through a process, should give us an idea not only of the time required to document using Unity, but, also, an improved estimate of time to implementation with an experienced user.

The fourth phase is a more difficult one. During this phase we need to implement a set of programs to increase the data available for the creation of X-Specs. Unity has many tools that can be used in the generation of X-Specs including mapping fields to semantic names. It may be impractical to port these to the host system in order to make them useful. Instead it may be enough to fill in the skeleton of an X-Spec for a table/field combination. Unity could then be modified to use the information contained in these partial X-Specs to complete the process or add information before presenting these to the user for completion. This phase will consume the majority of the time allotted for the project.

The fifth phase of this project will involve modifying Unity to generate RETRIEVE query statements that will execute on the host system rather than through an ODBC client. This will be useful because the multi-valued nature of the host system is best utilized on the host system itself.

Conclusion

In order to utilize information, it must be accessible. Accessibility includes both documentation of available data and tools for accessing the data. Unity provides a tool that can assist with the documentation of ODBC accessible data and with the retrieval of the information. However, with a poorly documented multi-valued database system with a lack of information available via ODBC, these tools may not be enough. This project will provide some extensions to Unity that will allow it to work with a multi-valued database system. Additionally, this project will research additional steps and programs to help increase the availability of information via ODBC.

References

[1] R. Lawrence and K. Barker: Unity - A Database Integration Tool. TRLabs Emerging Technology Bulletin December 4, 2000

[2] IBM: IBM Software: Database and Data Management: U2 product family: UniVerse: Overview. http://www-3.ibm.com/software/data/u2/universe/

[3] R. Lawrence and K. Barker: Integrating Relational Database Schemas using a Standardized Dictionary**.** SAC'2001 - 16th ACM Symposium on Applied Computing March 11-14, 2001 Las Vegas, USA, pages 225-230.