

Table of Contents

- | | |
|--|--|
| <ol style="list-style-type: none"> 1. Objective 2. Collection of Customer Source Code 3. Parsing and Analysis 4. Partitioning 5. Normalization 6. Source Code Conversion <ol style="list-style-type: none"> 6.1 NatLogic Base Classes 6.2 Support Base Classes 6.3 Translation Engine 6.4 Manual Remediation 7. Data Migration 8. Performance | <ol style="list-style-type: none"> 9. Application Testing <ol style="list-style-type: none"> 9.1 Unit test environment 9.2 Acceptance test environment 9.3 Data Conversion testing 9.4 Parallel testing 9.5 Macro level 9.6 Micro level 9.7 Multi-user testing 9.8 Problem reporting 10. External Interfaces <ol style="list-style-type: none"> 10.1 Batch JCL 10.2 Work Files |
|--|--|

1.0 Objective

This document is intended to provide an understanding of the step-by-step process for migration of a Natural/Adabas/DB2 application to a functionally equivalent Java based web application using the FBD Associates Inc. (FBDA) tools. These steps are illustrated in Figure 1 below.

(Note that, for simplicity, while Oracle is shown for the RDBMS in Figure 1, the FBDA tools support the conversion to multiple commercial and open-source RDBMSs, including the conversion of Natural/DB2(zOS) applications to Java/DB2(LUW) using the same general processes as for Natural/Adabas/DB2 to Java/RDBMS described herein).

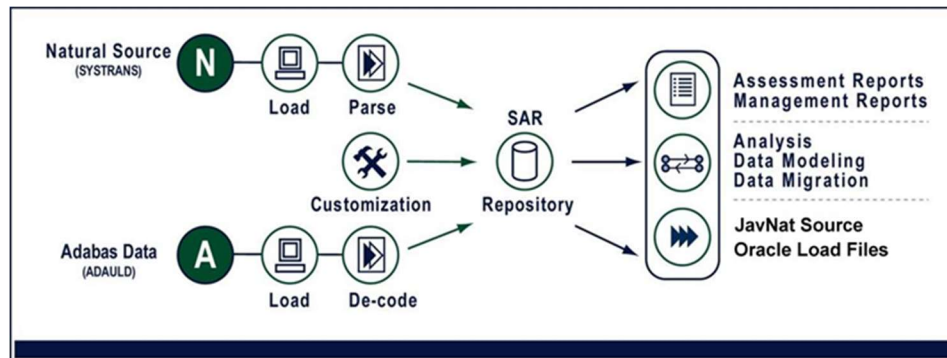


Figure 1 – Process Architecture

2.0 Collection of Customer Source Code

The process begins with the collection of all the Natural source code from the customer's application portfolio in SYSTRANS (or SYSOBJ) format.

The SYSTRANS must include the Adabas Field Definition Tables (FDTs) as well as the Natural Data Definition Modules (DDMs), Maps, Data Definition Areas (global, local and parameter data areas), and executable modules (programs, subprograms, subroutines, help routines). This can be accomplished using the Software AG ADAREP and SYSTRANS/SYSOBJ utilities.

To assist in this activity, FBDA will provide special instructions for source code collection. For details regarding the collection of source code email info@fbda.ca.

The SYSTRANS/SYSOBJ files can be emailed or ftp'd to FBDA for parsing and analysis processing.

Note: STEPLIB (if any) information must be provided to ensure accurate analysis and transformation.

3.0 Parsing and Analysis

The FDTs, DDMs, Maps and all other Natural source objects transmitted in the SYSTRANS/SYSOBJ are parsed, and the parsed data extracted and loaded into the relational database (RDBMS) based Syntax Analysis Repository (SAR).

Initially, an automated analysis of the parsed data is performed to determine the feasibility of conversion and identify problematic code (e.g.: dynamic code, external calls).

Using FBDA's JavNat Migration Support Tools (JMST ®), detailed application profile assessment reports (Discovery and Analysis or "DNA" - see below) are generated, containing a comprehensive description of the source code and application details which aid in estimating the size of the migration.

JMST is an interactive web-based tool that reads the SAR database and assists a user in performing several functions that support application conversion and data migration such as:

- View Application Profile Reports including general information such as:
 - application metrics
 - modules not parsed
 - missing objects (executables, ddms, data areas, copy code)
 - un-referenced subroutines
 - dynamic calls
- View reports showing potential conversion issues

- View source code in HTML format
- Normalize Adabas FDTs based on DDMs
- Create the DDL for schema definition, the control file and SQL for database triggers
- Read Adabas data files (ADAULD) and produce files for relational database loading

The NatMiner/DNA component of JMST produces comprehensive inventory reports that ensure the completeness and integrity of the code prior to conversion.

Figure 2 provides an example of the NatMiner/DNA HTML reports. Approximately 40 reports are provided depending on the features of the customer source code.

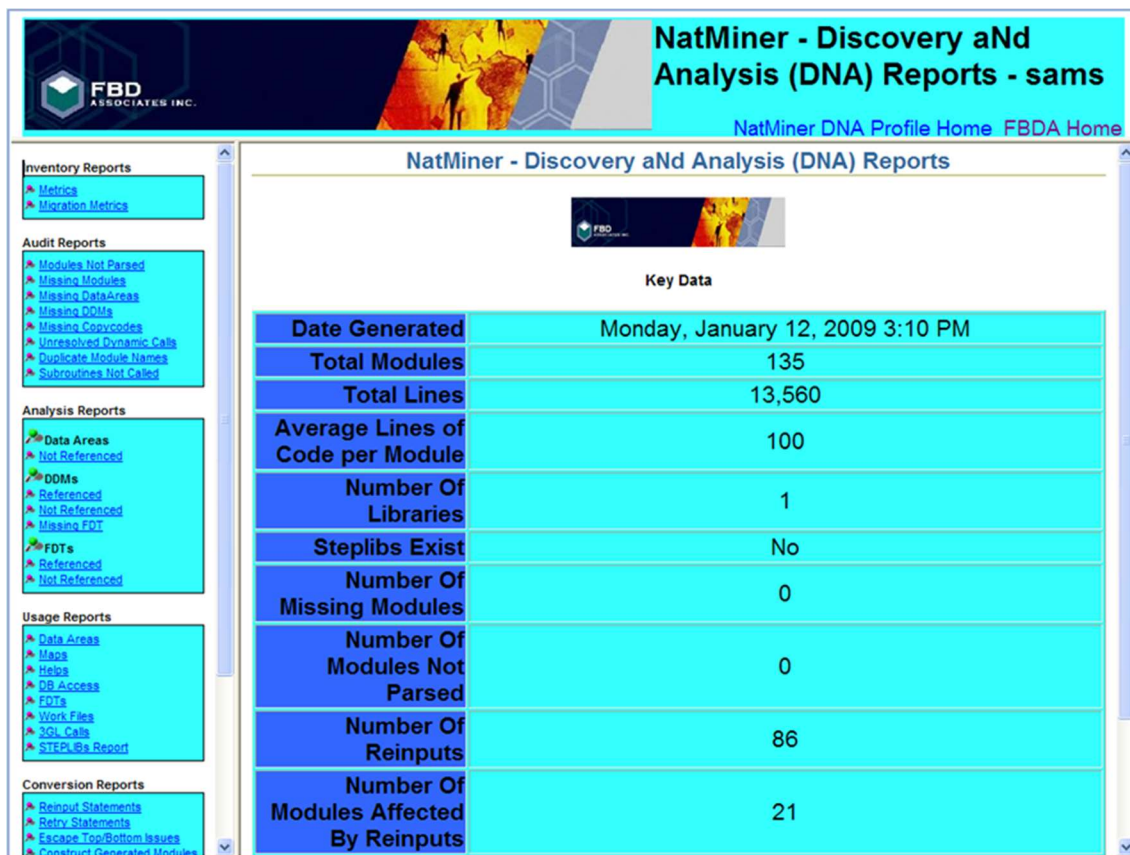


Figure 2 – Sample NatMiner Report

4.0 Partitioning

Using the JMST tool and the DNA reports together with the Natural source code and any available documentation, the application is analyzed and organized into logical units of work (partitions) that will be tested and delivered incrementally consistent with customer test and production support resource availability.

Partitions will be determined for the most part by functions within the application. Several functions that are made up of only a few modules may be combined into one partition while

other, more complex functions may be split over more than one partition.

Partitions are designed to follow the normal business procedures. For example, security and support file maintenance would be likely candidates for the first partition. Subsequent partitions would then use and build on the files already created and populated.

To achieve the highest quality results in partitioning, it is essential that the customer be involved in this activity from the beginning. The proposed plan will be reviewed, amended as

required and ultimately approved by the customer before work begins.

The source code translation will be implemented in phases with each phase consisting of one or more of these partitions. As the source code for each partition is translated it will be compiled and subject to unit level test to confirm functional performance.

5.0 Normalization

Once the missing modules are accounted for and the information in SAR is deemed to be complete and timely, the next step is normalization of the Adabas FDTs to create a field and table names that will be used in the creation of the new schema, as well as some ability to change data types and sizes. While changes may be provided in various formats (Excel, CSV etc.), optimally they should be recorded in the SAR database via the web using FBDA's JMST tool.

When normalizing an FDT, the DDM fields are presented side by side with the FDT information in JMST and can be used as a default for the new name. In the Figure 3 example there is only 1 DDM for the FDT.

Unless the complexity of the FDT-DDM relationship causes a conflict (e.g. 2 DDMs have different names for the same Adabas field), the JavNat field and table names generated correspond to the DDM names as a default.

Where an Adabas field has no corresponding field in a DDM, its name is built from the Adabas short name. However, the customer has the option of overwriting table and field names via JMST so that the new schema definition will be more meaningful to application developers.

By default, the relational tables will be built from the conversion details stored in SAR, according to the following mapping rules:

- Each FDT is converted to at least one corresponding relational table in the target DBMS whose unique key will be the ISN of the original Adabas record. This table will be comprised of all the fields from the FDT, except for Periodic Groups (PEs), Multiple Usage fields (MUs), and Group fields.
- PEs are normalized, becoming Child tables with Foreign Keys to the parent table.
- Similarly, by default MUs are normalized, and Child Tables created with Foreign Keys to the Parent Table.
- Optionally, an MU field may be mapped to a partitioned text field that contains all occurrences of the MU delimited by a special character, or a separate field might be created to hold each occurrence of the MU field. (e.g.: ADDRESS_LINE1, ADDRESS_LINE2, etc.)
- Each Superdescriptor (SP) and Subdescriptor (SB) is converted to a physical field in the corresponding table in the target database. A database trigger is created which maintains the structure of the SP or SB components when these are modified in the database. Under certain circumstances, SQL can be used to copy the

functionality of the SP/SB, eliminating the need for a physical field.

- Descriptors become indexes in the new schema.
- Phonetic descriptors will be replaced by database triggers that implement a Soundex algorithm.

Schema generation is an iterative process involving customer review and input until a satisfactory schema is agreed upon.

At this point, DDL is generated, including:

- CREATE TABLE, PRIMARY KEY and FOREIGN KEY statements
- CREATE INDEX statements
- PL/SQL script to build sub/superdescriptor triggers
- SQL to generate ISN sequence

The generation of the DDL is done online and takes seconds. It can then either be saved to SAR for later use or copied and pasted directly into SQL Plus to create the database tables, indexes and triggers.

6.0 Source Code Conversion

Once normalization is completed the source code can be converted. To ensure the generation of the correct DML it is essential that the FDT normalization details be stored in SAR before proceeding. This is done by clicking on the button **"Save Relational Data"** in the last step of normalization when one is satisfied with the results.

The JavNat source code conversion strategy is based on key components as follows:

6.1 NatLogic Base Classes

The NatLogic base classes are a collection of custom Java methods that implement the Java equivalent functionality of the Natural language syntax. In general, each Natural statement is mapped to a Java method on a one-to-one basis. The JavNat methods are designed to give the converted code a Natural "look and feel."

6.2 JavNat Transformation Framework

The JavNat Transformation Framework (JTF) is a collection of custom Java classes to provide the Java equivalent functionality for certain Natural runtime components and features. Some examples are:

- NatSession - maintains and controls the JavNat runtime environment.
- NatSysVar - manages the system variables such as *DATX and *COUNT.
- NatStackItem - manages items passed on the stack
- NatWorkFile - manages the disk I/O for the Natural Read/Write Workfile statements
- NatDDM - maps View structures to the relational schema.
- Java Batch Framework (JBF) - components to support Java batch operations in the new environment

6.3 Translation Engine

A SAR driven tool (Natural2Java Converter) translates each Natural module source code, on a statement-by-statement basis, to a functionally equivalent Java class using methods of the NatLogic base class and native Java statements to create Java logic that will execute exactly as it was executed in the Natural environment. Each Natural source module is converted to a Java class that extends the NatLogic base class.

The new Java classes corresponding to the Natural program modules use the NatDDM base classes to generate SQL statements that are executed through JDBC to the relational schema. Logic is generated to read data from the relational schema and store it back into the Natural view structure in the same way that Natural functions. This applies to programs, subprograms, help routines, maps, LDAs PDAs and GDAs.

Within the support base classes there are methods defined to handle each Natural system variable. These methods perform the data manipulation necessary to obtain the required results. For example, a method called sDATN will return the system date in the format YYYYMMDD.

Appendix B provides a more detailed description of the translation engine operations.

Appendix C shows an example of a Natural program and the equivalent JavNat output by the converter.

6.4 Manual Remediation

The converted JavNat programs, subprograms and subroutines will sometimes require some manual remediation to compile and function as designed. The Natural 'REINPUT' statement, as well as a handful of other situations will always require some manual inspection and/or resolution. The manipulations required to ensure a successful compilation and functioning of these statements are known and documented. This effort will vary depending on the application, and our experience has been that on average this may take 15 minutes per module.

The converted JavNat versions of MAPS, DDMs, and DATA AREAs compile with no further changes.

7.0 Data Migration

JavNat also provides tool-supported data migration from Adabas production files to the relational tables. An automated process that can be managed using NatMiner® or independently, combines data from SAR with the newly created relational schema to enable the following:

- Creation of a relational based data staging area derived from the transformed Adabas schema.
- Population of the staging area with data from the Adabas ADACMP files to facilitate subsequent data validation, data type mapping and data warehousing activities. Adabas data will be extracted from data files produce using standard Adabas utilities.
- Generation of control files based on the new relational schema, that can be used to create files for bulk loading of the Adabas data into the production relational tables.

The data migration process can be repeated as many times as needed. Data migration will almost always require some

degree of data validation and mapping. JavNat can provide automated support for activities such as:

- Validation that a field specified as a date in Adabas and stored as an A6 contains valid YYMMDD occurrences and uses lo-values or blanks as a null value.
- Validation of implied foreign key existence and constraint checks.
- Mapping an Adabas A6 date field to a relational date field with blank occurrences replaced with NULL.
- Mapping an Adabas SSN representation to an appropriate relational representation.
- Expansion or replacement of state, region or status codes.

8.0 Performance

All application platforms and databases have specific issues when it comes to performance and customization. Performance issues that appear during testing can be tuned using a number of methods:

With the object-oriented nature of Java, cosmetic and functional changes can be made to many of the JavNat classes to customize the application where needed.

Customers are provided a complete set of source code to allow for in-house customization or can contract for this service on an as required basis. Customization methods might include:

- a) DBAs can tune the new relational database using such techniques as adding or removing database indexes or may involve changes to the new schema that requires an automated re-conversion of the application.
- b) Changing the functionality of a particular JavNat I/O method for better performance.
- c) Adding new JavNat I/O methods to handle specific database access situations.
- d) Replacing JavNat methods with native Java JDBC calls where needed.
- e) Changing or adding JavNat methods to improve logic.
- f) Replacing old Natural-like business logic with new object-oriented business rule concepts.
- g) Re-coding sections in native Java statements.

9.0 Application Testing

The JavNat conversion process is based on a strict statement-to-statement conversion approach. Consequently, there is a one-to-one correspondence between converted JavNat code and the Natural application.

The logic and business rules are identical to the original application, and the user interface has the same appearance unless it has been customized at the user's request.

The JavNat conversion process simplifies testing in that testing can be done in parallel with the existing application using a Natural test environment. Existing test scripts can be applied without change. Database updates and reports can be compared to ensure that the converted application is functioning as required.

It is recommended that 2 environments be set up for testing in both the JavNat and the Natural environment as follows:

9.1 Unit Test Environment

This environment will not be too closely controlled. It is for unit testing and since the testers may not be completely familiar with the application, the data entered may not always make perfect sense. The purpose of these tests is to determine whether each screen seems to function as it should, and it is not intended for system testing.

9.2 Acceptance test environment

This environment will be strictly controlled, and testing will follow scripts which take the user through a logical cycle(s) of processing including both batch and online. When 'parallel' testing is discussed below it is with this environment in mind.

Since the snapshot of the data will most likely be out of date, the **system date** can be easily set in the JavNat environment to correspond to the date of extraction. Of course, this must also be done in the Natural environment.

9.3 Data Conversion Testing

Verification of the data conversion would involve doing an automated comparison between the converted relational tables and the original data.

To accomplish this, the client would provide a dump (flat file) of the data prior to conversion. To avoid the possibility of interim change, this should accompany the ADAULD when it is provided for testing. A dump of the converted data in the same format would be obtained using FBDA tools. These two files can then be compared using standard 'diff' software and the results reviewed.

9.4 Parallel testing

The Adabas data supplied for testing is loaded into a Natural test environment. The converted data is loaded into relational tables. Using 2 terminals (or 2 screens on the same terminal) users perform the same functions in both systems and compare screens. For parallel testing it is essential that scripts be prepared and followed rigorously.

These tests will follow the process from beginning to end, including such things as rollover, month end, year end and any other process cycles, batch reports and any external interfaces there may be. It should exercise every part of the system.

Verification should be done through a regular comparison of:

- database contents
- printed outputs (reports)
- output (work) files produced

9.5 Macro level

At the end of a day (or cycle or whatever unit is desirable) of testing, the data comparison should be done between the Adabas, and relational files and any other outputs produced.

9.6 Micro level

Note should be taken of every field that appears on a screen. For example, attention should be given to:

- date and time formats
- negative numbers
- special editing of fields (edit masks such as 'XXX'- 'XX'-'XXX')
- leading zeros
- trailing blanks
- decimal places
- upper/lower case
- functioning of all PF keys
- error messages
- position of cursor on entering screen

9.7 Multi-user testing

Once unit testing has been done and the initial system testing has flushed out most of the bugs, the application should be subjected to multiple users working simultaneously. As early as possible tests that involve two or more users at a time should be performed. This type of testing should be scheduled as partitions are rolled out, not deferred until the project is completed.

10. External Interfaces

10.1 Batch JCL

For zOS deployments, the existing mainframe batch is supplemented with FBDA provided JCL PROCs as appropriate to allow continued use of existing JCL to execute the Natural batch modules now converted to Java.

For non-zOS deployments, any existing JCL used to submit batch will be submitted to the FBDA provided JavNat Batch Framework (JBF) as it was on the mainframe. The JCL will execute the now Java equivalents of any Natural and non-Natural steps.

There may be some development work needed if there are 3rd party executables in the JCL or if some of the features have not been implemented.

Other batch implementations such as transforming/replacing JCL with python are available via partners.

NATRJE functionality will also submit JCL to the JBF using the REST web services build into JBF.

10.2 Work Files

The converted application will have a special Java class which provides the equivalent functionality as the mainframe work files via an access to the target environment file system.