javnat™ MIGRATION TOOLS

Natural/Adabas to Java/RDBMS

JAVNAT - ENABLING THE FUTURE

Until now users faced with the challenge of migrating their legacy Natural/Adabas applications to more modern technologies have been stymied by the cost, time and generally poor quality of manual translation. Often replacement with a package application provides the only effective solution but at the expense of the robust custom functionality of their legacy applications.

The state-of-the-art JavNat migration tools are designed to automate the conversion of legacy Natural applications to Java. A powerful transformation engine implements an efficient, high quality transition from a terminal based Natural/Adabas environment to a multi-tier Java application integrated with a modern relational database.

Natural source code is analyzed and parsed in detail and the parsed output stored in an RDBMS based Syntax Analysis Repository (SAR). All subsequent schema and data migration, as well as source code conversion, is then driven from the SAR database. The functionality of the resulting Java/RDBMS application is identical to the original Natural/Adabas application. The transformation strategy also preserves the "look and feel" of the Natural application while enabling continuing development in a modern web oriented environment. The strict functional equivalence approach with JavNat provides the additional benefit of simplified testing and training.

The JavNat process is illustrated on Page 4 and includes:

Assessment of the application
RDBMS schema generation
Source code conversion
Data Migration
Performance Tuning and Customization

ASSESSMENT - Getting Started

The JavNat process starts with the collection of all the Natural source code from the customer's application portfolio, including File Description Tables (FDTs) and Data Definition Modules (DDMs), in Systrans format. This can be easily accomplished using the Natural Systrans/Adarep utilities and we will provide special instructions for source code collection. As it is crucial that all source code be supplied, the focus of preliminary processing is to identify any missing or extraneous modules.

The Systrans file is transmitted to the JavNat processing center where the FDTs, DDMs, Maps and all other Natural source objects are parsed into Abstract Syntax Tree format.

Then the parsed data are extracted and loaded into SAR. For each Natural module there is a corresponding entry in the SAR database with its associated statements, data elements, references, and all other information required to rebuild the application preserving all of its functionality.

ASSESSMENT - Analysis

Initially, an automated analysis of the parsed data is performed to determine the feasibility of conversion, and to identify problematic code (e.g.: dynamic code). Detailed assessment reports are generated, containing a comprehensive description of the source code and application details which aid in estimating the size of the migration. The SAR database and reports can also support application mining for migration planning and the development of test scenarios.

Comprehensive inventory reports showing missing modules ensure the completeness and integrity of the code prior to conversion. During the assessment, any modules and/or DDMs that are identified as unused will be purged from SAR database before proceeding with subsequent processing stages.

RDBMS SCHEMA GENERATION

Once the parsed data is loaded into SAR, the customer has the option of specifying 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.

The JavNat default field and table names for FDTs are cryptic, as they are built from the Adabas short names. However, the customer has the option of overwriting table and field names 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 with the exception of 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.

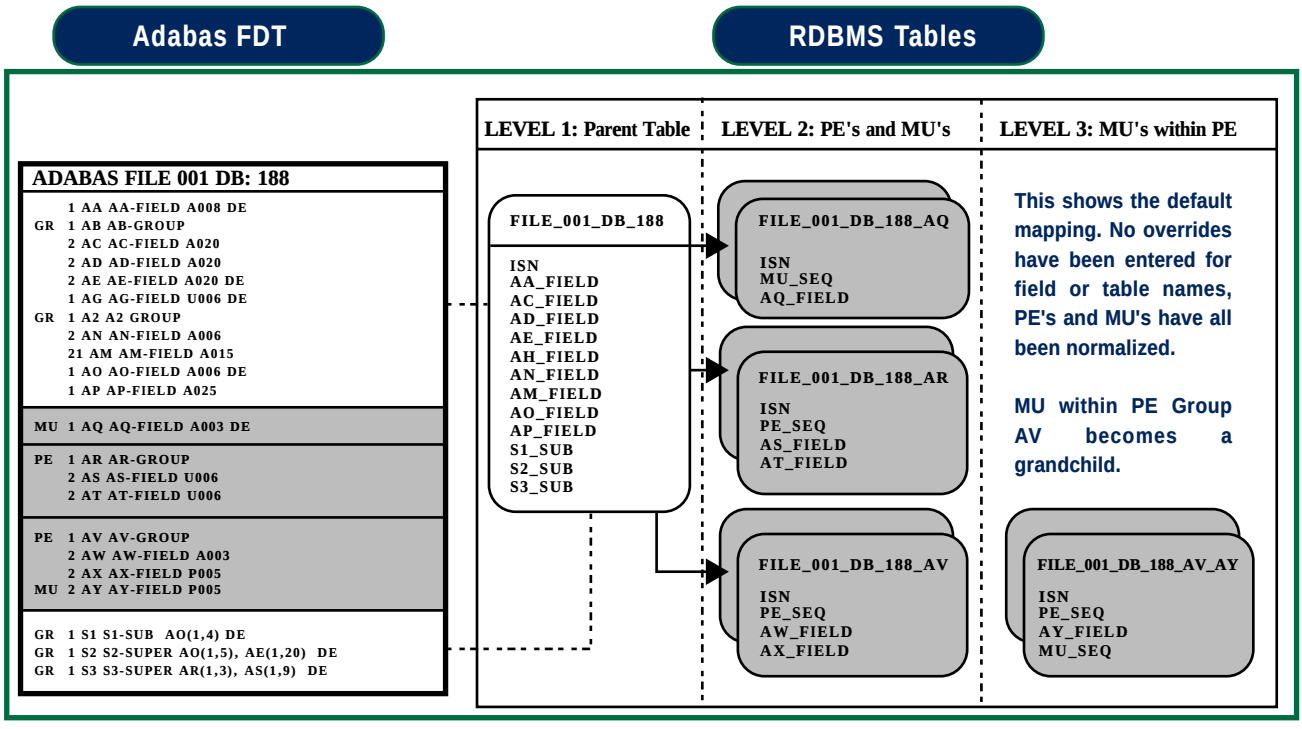
Descriptors become indexes in the new schema.

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.

Once customer approval is obtained, the data from SAR is used to generate the CREATE TABLE, PRIMARY KEY and FOREIGN KEY statements.

An example of a simple Adabas to RDBMS schema conversion is illustrated below.



SOURCE CODE CONVERSION

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

NatLogic Base Class: a series 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."

Support Base Classes: a series 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 RDBMS schema.

Translation Engine: a SAR driven tool that converts 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 RDBMS schema. Logic is generated to read data from the RDBMS schema and store it back into the Natural view structure in the same way that Natural functions. This applies to programs, subprograms, helproutines, 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.

In the JavNat generated DML example below, note the use of such methods as "NoRecordsFound", "Compress", and "ReadPhysical", which will be familiar to Natural developers.

NATURAL	CONVERTED JAVA

010 DEFINE DATA LOCAL
020 1 EMPLOYEES VIEW OF EMPLOYEES
030 2 PERSONNEL-ID
040 2 FULL-NAME
050 3 FIRST NAME
060 3 NAME
070 3 MIDDLE-NAME
080 2 SEX
090 2 FULL-ADDRESS
100 3 ADDRESS-LINE (1:4)
110 3 CITY
120 3 POSTAL-CODE
130 3 COUNTRY
170 1 #NAME (A40)
0 180 END-DEFINE
	//Define Data
	Local() ;
	View(1, "EMPLOYEES", new dEMPLOYEES());
	Decl(2, "PERSONNEL-ID",'N',8);
	Decl(2, "FULL-NAME",'G');
	Decl(3,"FIRST-NAME",'A',20);
	Decl(3,"NAME",'A',20);
	Decl(3,"MIDDLE-NAME",'A',20);
	Decl(2,"SEX",'A',1);
	Decl(2,"FULL-ADDRESS");
	Decl(3,"ADDRESS-LINE",'A',20,1,4);
	Decl(3,"CITY,'A',20);
	Decl(3,"POSTAL-CODE",'A',10);
	Decl(3,"COUNTRY",'A',3);
	Decl(1,"#NAME",'A',40);
	//End-Define

260 READ EMPLOYEES PHYSICAL 270 REJECT IF CTY NE 'PERPIGNAN' 280 WRITE 290 1T NAME 30T PERSONNEL-ID 300 END-READ	ReadPhysical ("EMPLOYEES"); { while (!EOF("EMPLOYEES")) if(Var("CITY") .ne("PERPIGNAN")) continue; Write (1,Var("NAME")); Write (9, Var(PERSONNEL-ID")); WriteRow (1); Next ("EMPLOYEES");} Close("EMPLOYEES");

900 FIND (5) EMPLOYEES WITH PERSONNEL-ID > 20
950 REJECT IF SEX = 'M'
970 ACCEPT IF COUNTRY = 'UK'
980 COMPRESS FIRST-NAME NAME INTO #NAME
010 END-FIND
	Find(5,"EMPLOYEES","PERSONNEL_ID > 20");
	while(!EOF("EMPLOYEES"))
	{ if(VAR("SEX") .eq("M"))
	continue;
	if(!VAR("COUNTRY") .eq("UK"))
	continue;
	Compress(false);
	Comp(Var("FIRST-NAME"));
	Comp(Var("NAME"));
	Intro(Var("#NAME"));
	Next("EMPLOYEES"); }
	Close("EMPLOYEES");

Natural "Look and Feel"

Business Logic Unchanged

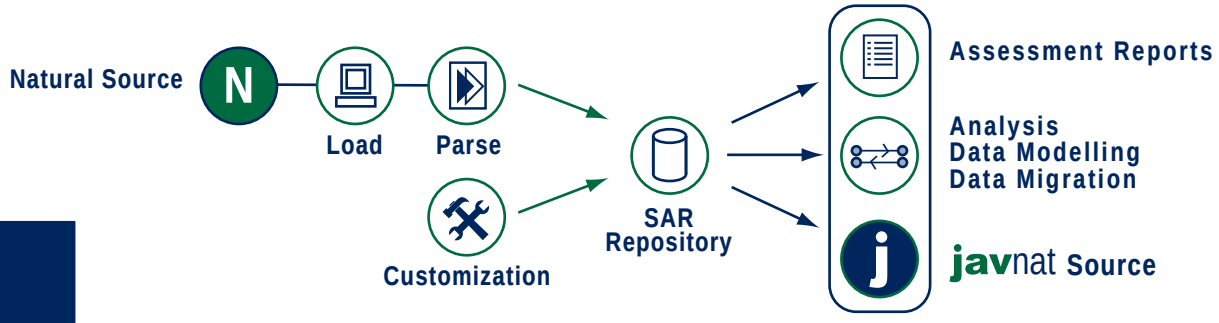
Easy to Maintain!

The Local Data Area is translated into Java. The result is very similar to the Natural data definition. Note: dEMPLOYEES is an extension of the NatDDM base class.

Once declared as an instance of dEMPLOYEES, all database actions trigger the reading of corresponding tables and transfer of the contents to the defined data area. Updates will read the contents of the defined data area and will apply changes to the appropriate tables of the relational schema.

The Java method corresponding to 'Find' will accept a number of optional parameters which translate clauses such as WITH, WHERE, SORTED BY, (LIMIT) etc.

The Natural COMPRESS statement has many options. These are handled by the parameters passed to the Java Method "COMPRESS".



javnat™ PROCESS OVERVIEW

DATA MIGRATION

JavNat also provides tool-supported data migration from Adabas production files to the RDBMS tables. An automated process combines data from SAR with the newly created RDBMS schema to enable the following:

Creation of an RDBMS based data staging area derived from the transformed Adabas schema.

Population of the staging area with data from the Adabas files to facilitate subsequent data validation, data type mapping and data warehousing activities. The Adabas data will be extracted from data files produce using standard Adabas utilities.

Generation of control files, based on the new RDBMS schema, that can be used to create SQL load files for bulk loading of the Adabas data into the production RDBMS 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 actually 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 an RDBMS date field with blank occurrences replaced with NULL.

Mapping an Adabas SSN representation to an appropriate RDBMS representation.

Expansion or replacement of state, region or status codes.

PERFORMANCE TUNING & CUSTOMIZATION

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

Database Performance

DBAs can tune the new RDBMS as required using techniques such as:

- *Adding or removing database indexes.*
- *Changing the functionality of a particular JavNat I/O method for better performance.*
- *Adding new JavNat I/O methods to handle specific database access situations.*
- *Replacing JavNat methods with native Java JDBC calls where needed.*

Logic Performance

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:

- *Changing or adding JavNat methods to improve logic.*
- *Replacing old Natural-like business logic with new object oriented business rule concepts.*
- *Re-coding sections in native Java statements.*

For More Information Contact
Frank Driscoll
White Rock, British Columbia V4B 1L9
fbdris1@attblobal.net Tel: (604) 538-5094