Publicitas SA Naca: Cobol transcoding software stack documentation

0. Introduction

0.1. *Contacts*

Please use the following contact information for details:
Web site: http://technology.publicitas.com
email: naca-contact@publicitas.com

0.2. *Licence*

Naca is currently made of 4 different projects:

- JLib, Publicitas Java base library, licenced under LGPL2 terms.
- NacaRT, Naca transcoded program base library, licenced under LGPL2 terms.
- NacaTrans, Cobol to Java transcoder, licenced under GPL2 terms.
- NacaRTTests, Test programs, licenced under GPL2 terms.

0.3. *Concepts*

Naca is a full set of tools / libraries that permit a complete migration for mainframe based central applications, to low costs server based infrastructure.

These servers can use any operating system supporting Java JRE 1.5 or greater.
Linux or Windows are fully supported, with 32 or 64 bits versions. 64 bits supports enables a much wider memory space than 32 bits.

The challenges are important, as mainframe applications are critical for owner company daily production.

1. Document coverage

This document does not cover all aspects of the various steps required for a complete migration from a host/MVS world to a Java/Server based environment.
However, the following points are detailed here:

- Legacy Cobol application automatic conversion to Java
- Utilities and tools used during migration preparation
- Some utilities and tools used in the new environment, to ensure daily production

Thus, this document is mainly targeted to issues involved by the automatic transcoding, and their solutions, as provided by the Naca software stack.

2. Naca software stack

Naca provides various tools, libraries and executable. They are usable in all or some steps of the conversion process.

The list of these software resources includes:

- Automatic CICS Cobol to Java transcoder.
- Automatic Batch Cobol to Java transcoder.
- Automatic DB2 stored procedure Cobol to UDB Java stored procedure transcoder.
- Filepac scripts transcoder.
- Runtime Java library used by all transcoded programs, in supported category.
- External Sort utility.
- Database utilities.
- File transfer from host to server utility.

- File encoding/decoding utilities, including complete support for EBCDIC / ASCII conversions.
- Modified 3270 terminals emulation tools.

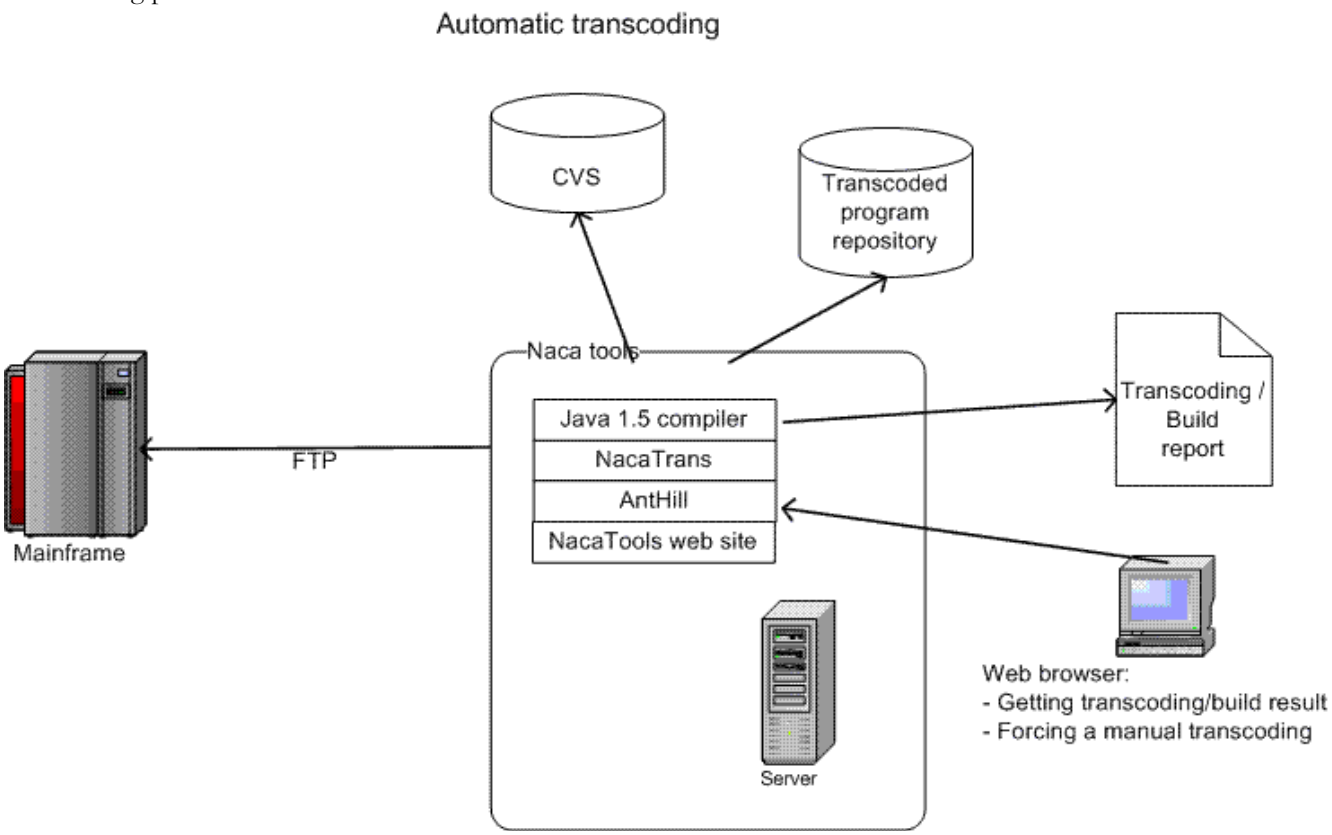
The list of required web servers includes:

- Standard source code repository ("CVS" for exemple).
- Standard build server ("AntHill Pro" for example).
- Custom integration server for daily and on-demand transcoding and build.
- NacaTools server, helping to centrally manage inventories, tests, operations, monitoring, statistics, ?
- Database instances in DB2 and UDB forms..
- Test servers for batch operations..
- Test servers for online operations.

3. Transcoding and runtime concepts

Transcoding is automatic and 99% of time does not require configuration file. The transcoder automatically reports when it needs a specific configuration file for a given program.

Transcoding process:



Naca deals with Java code generation and execution.
The generated code thus must meet some strong points.

The generated java uses a Cobol-like syntax. It's as close as possible from original Cobol syntax, within of course the limits of the Java language.
Generated code doesn't look like classical native java and is not object oriented from the application point of view.
This is a by design strong choice, to enable a smooth migration of Cobol developers to the Java environment. The goal is to keep business knowledge in the hand of people who wrote the original Cobol programs.

All Java transcoded programs must sports the following characteristics:

- Maintainable by the Cobol team
- Give similar results as original code
- Achieve at least similar performance response time as their host origins.
- Application monitoring and statistics

3.1. *Maintenance*

Human maintenance implies that Java generated programs have the following characteristics:

- They are humanly readable.
- The Java generated code is to be maintained by trained Cobol developers, with a thin leaning curve of the Java environment and tools. They are not intended to be maintained by trained Java developers.
- Transcoded program must achieve a 1 to 1 line matching with the Cobol program. This implies that they are not object oriented, and that the original line ordering is kept.
- A maximum of code checking must be done at compilation time, not at runtime. This guarantees a maximum correctness of a program, even when it's maintained manually.
- SQL statements must be directly visible in the source code, as is the case in MVS Cobol. It's technically dynamic SQL, but syntaxically looks like static SQL.
- They must be testable easily.
- Central compilation, or during development, in a developer workstation's Eclipse is required.
- They can be debugged in an Eclipse environment within the developer's computer.
- Remote debugging must be supported.
- Execution must provide at least the same error diagnostics as the Cobol root. In fact, better diagnostics are offered, for example by checking at runtime if a given occurs index is in the validity range.

3.2. *Results correctness*

This is achieved by the following rules:

- Transcoded programs are a direct transcription of the Cobol source, using Java syntax instead of Cobol syntax.
- The NacaRT library provides a software layer that emulates a complete Cobol runtime, with all it's specificities. This is the key point that guarantees similar execution.

3.3. *Performances*

Performances are achieved by the using following techniques:

- NacaRT handles many caches for a lot of internal resources. The following resources are strongly cached. For example:
  - Programs and program instances
  - Variables
  - SQL connections, statements, resultset columns, ? In fact, SQL is very deeply cached, with more than 5 levels of caching.
- Screen IO requires CPU intensive XSL transform. These transforms are in fact compiled once for during the 1st program instance execution, and reused later.
- Online programs sessions are naturally executed concurrently. Each session runs in it's own private thread. Thus, the whole system benefits directly form modern multi-core CPU architetures.
- Batch program are sequential and mono-threaded by nature. However, internal or external sorting are multi-threaded to achieve good results. File accesses are cached in NacaRT, on top of Java JRE own cache, which itself lives on top of the OS own file system cache.
- Java JRE >= 1.5 is mature and it's various memory management pools are efficient.
- Garbage collection is sometimes a blocking operation done at will by Java JRE. It's not possible to avoid completely these steps, but a lot is done in NacaRT to keep and reuse objects as often as possible. This helps limiting these kinds of blocking garbage collections. However, a fine tuning of the JVM parameters has to be set.

3.4. *SQL Support and characters encoding*

3.4.1. Database

MVS Cobol targets a DB2 database, and uses EBCDIC character set.

Java uses internally Unicode UTF-16 character representation, which is completely different from EBCDIC. Transcoded programs may be run using a mainframe DB2 instance, or a UDB instance. UDB can be setup in a selectable character encoding. However, the recommendations are to use Unicode UTF-16 character sets.

3.4.2. Batch files

The same situation occurs for batches.
Host needs ECBDIC, while Naca must support either ASCII or EBCDIC encodings.
Note:ASCII is a subset of Unicode UTF-16.

3.4.3. NacaRT support

NacaRT provide a mean to deal with these various character representations. Internally, everything is based upon Java standard, UTF-16.

When reading or writing data to a database, the database driver does automatically the required conversions at runtime.
When reading or writing data to a file, NacaRT file access layer does also the required conversions at runtime. It takes care to convert only required fields, depending on their Cobol type as described in the program source code. For example, numeric encoded comp-3 fields must not be converted, as their value does not depend on their character set.
This is a powerful mechanism that covers all required conversions aspects, even in external sort.
For example, it's possible to sort an EBCDIC file by using ASCII comparisons, if required.
External sorting uses collating sequences to compare key-segments in the required representation.

3.5. *Monitoring*

Transcoded online programs are executed in a Tomcat container, that hosts Java JMX (Java Management eXtensions) services. NacaRT adds various JMX Mbeans components that permits the following remote operations, through a JMX console:

- Checking memory, threads and loaded classes usage.
- Access to NacaRT general statistics, showing for example:
  - Number of active cached DB connections.
  - Number of active cached prepared SQL statements.
  - Number of correctly executed transactions.
  - Number of erroneously executed transactions.
  - Number of loaded programs
  - Number of currently running programs.
- Application site can be closed or open at will, or by using a planed open/close calendar.
- Log settings can be updated

Another important source of statistics is provided by registration in a central DB of a record when an Online transaction is run, and when it's terminated. This in-DB transaction log provides a mean to bill transactions.

4. Required environments

Naca targets a Java environment for enterprise legacy applications.

There are multiple environment to consider, depending on it's usage.
Most components are open source based, thus helping lowering considerably the cost of ownership of these infrastructure elements.

4.1. *Production environment*

4.1.1. Production server infrastructure.

Depending on application scaling, multiple physical servers may be required.

Each server must match these requirements:
- Standard Intel/AMD X86 hardware with optional multi-core/multi-CPU support.
- Linux 32 or 64 bits. 64 bits is recommended for large applications support due to memory constraints on 32 bits platforms.. the preferred distribution is RedHat Enterprise, but other choices are possible.
- Sun's Java JRE >= 1.5 in 32 or 64 bits edition.
- Apache's Tomcat.
- NacaRT library, and the required third part libraries.
  These required libraries are all open source software only.
- Transcoded application binaries.
- Production environment configuration files.

4.1.2. Central database server

It can be either:
- MVS based:
  - IBM DB/2 with DRDA access. Note: The access to DB2 through DRAD impose a performance penalty. However, cares has been taken by NacaRT to limit at most this penalty.
- X86 based:
  - IBM UDB on x86 computer.

4.1.3. File server to store flat files for batch processing

- Linux box on Intel X86 hardware.

4.1.4. End user workstation:

- Any Windows computer with Microsoft Internet Explorer 6 or 7 support.

Note: There is no additional hardware / software requirement. All screens outputs produced by Naca are HTML based. Thus, there are no deployment requirements for user workstations.

4.2. *Development Environment*

This environment is used before migration to prepare it, and after to maintain the applications.

4.2.1. Server infrastructure

- Source code repository (CVS for example), using a Windows / Linux box.

- NacaTools web server
  - Linux box on Intel X86 hardware
  - Tomcat
  - Ant or Anthill Pro,
  - Ant scripts for build processing
  - NacaTools web site

- File server to store flat files for batch processing
  - Linux box on Intel X86 hardware

4.2.2. Developer workstation

These persons are in charge of maintaining the Cobol or Java source code.

Requirements:
- Intel/Amd X86 computer with at least 1Gb of Ram (2Gb recommanded).
- Windows or Linux.

- Sun Java JDK >= 1.5.
- JMX console.
- Tomcat.
- Eclipse >= 3.2 with Tomcat integration plug-in.
- NacaTrans Eclipse plug-in.
- CVS / Subversion repository integration.
- NacaRT + third part libraries.
- Source code of Cobol files.
- Tools to access to various Linux boxes.

4.2.3. Tester workstation

The testing phase prior migration imposes the creation of scenarios. These scenarios are scripts that are created by running a instrumented QWS3270 terminal emulation software. The persons who create these scenarios require this commercial component.

Requirements:
- X86 hardware running Microsoft Windows. XP Professional (Vista untested).
- Jolly Giant's QWS3270 and it's naca custom modification.

5. Tools details

5.1. *Cobol to Java transcoder: NacaTrans.*

This tool is a critical component of the naca software stack.
It can convert the following type of Cobol source files:
- CICS programs
- Batch programs
- UDB Stored procedure
- Copy
- Called programs

This tool takes a Cobol source file or list of Cobol source files and produces one or many output Java source files. The dependencies for a given Cobol source file are taken in account during transcoding operation. For example, if a CICS program requires a copy source, it's also transcoded. The list of file to transcode can be grouped by category (Batch / Online) and by CICS transaction.
The transcoder produces the following files:
- Java output files representing:
  - Online programs, from Cobol - CICS input programs.
  - Batch programs, from Cobol - batch input programs.
  - Batch programs, from Cobol - batch input programs.
  - Store procedure programs from Cobol DB2 stored procedures.
  - Called programs from Cobol routines, used as a called program.
  - Copy files from Cobol Copy.
  - Copy file and XML resource file from CICS Map screens Copy.
- An error/warning output list.
- A statistics file, giving metrics of the transcoded sources.
- A log file.

The transcoder can be used in multiple containers, depending on it's usage:
- As a centrally managed resource, either through the integration server or directly through the build server. This is used for daily/continuous transcoding build.
- As a local developer's tool, into he's workstation. This is used as a standalone developer tools.
- As an Eclipse plug-in. This feature enables Cobol developers to maintain Cobol source files, even after the migration is completed. This permits Java untrained Cobol developers to maintain Cobol source code: they can develop / maintain Cobol source files, transcode them in Java within their Eclipse environment, and then run

them directly in Eclipse. This helps lowering stress and formation pressure to the Cobol teams.

The programs produced by NacaTrans are not bound to a specific container. In fact, they can run the same in these various containers:

In production:
- Within a Tomcat web-apps container.
- As a Command line executable.
- Within a UDB instance, as a stored procedure.
- Legacy transcoded programs can be reused by normal java applications. Thus, they are interoperable.

In development stages:
- Within a Tomcat web-apps container.
- As a Command line executable.
- Within the NacaTools server, for testing purpose and playback scenario execution.
- Within a UDB instance, as a stored procedure.
- Within a Tomcat web-apps container using remote debugging.
- Within the Eclipse debugger, optionally using a profiler (JProfiler for example).

5.2. *Naca runtime library: Naca's NacaRT*

This java files generated by the transcoder makes a wide usage of a Naca specific runtime library.
This library supports and emulates a full MVS Cobol syntax support and some Online and Batch utilities.
It also supports execution of transcoded Filepac scripts.
NacaRT is another critical part of the Naca Software Stack.
It requires Sun or IBM JRE (Java Runtime Environment) version 1.5 or greater.

5.3. *Naca operation management: Naca's NacaTools*

This component is implemented as a web server.
It provides multiple screens, to ease daily operations before, during, and after migration.
These covered operations include:

- As a initial preparation step, all programs must be inventoried and registered. This inventory part is centrally managed by the "NacaTools" server.

- During preparation of the migration, all programs must be converted. The various stages of conversion for all inventoried programs are maintained by this web-server.

- All converted programs must be deeply tested before migration. These tests can be run manually or automatically, and their status are registered centrally.

- Statistics can be generated, reporting the advancement of conversions and tests.

- Before and during migration, files must be transferred from host to servers.

- After migration, some programs may be still maintained as Cobol source files. NacaTools automates the daily conversion of these programs..

- During tests and after migration, servers must be monitored. A screen that sums-up all servers' status is available.

6. Testing

Tests are critical to a successful migration preparation. It requires various steps for a given transaction, and all programs must be thoroughly controlled. What's more, testing online transaction implies different controls from testing batch

programs.

6.1. *Testing Online transactions*

The following operations are typically done for a test:

Scenario creation; done only at a preliminary stage:

- First, the Database is setup in known state.
- Then, a scenario is created. This is done manually by running an instrumented 3270 terminal emulator that executes the targeted application in the host environment.
- These scenarios are kept and managed centrally in the NacaTools Web server, with the queries that had setup the database in it's initial state.

Scenario playback, executed at will:

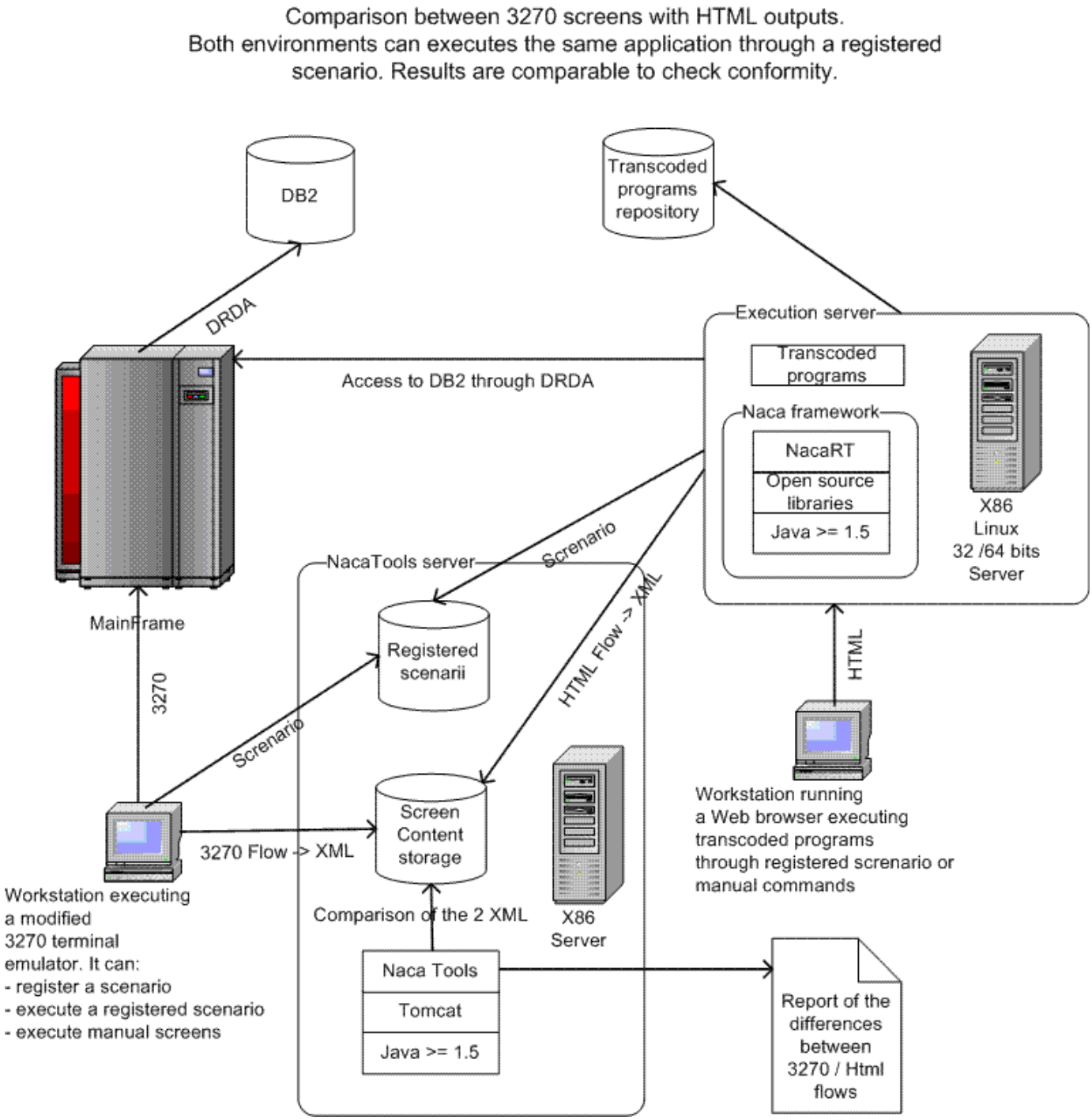
- This database tables are setup in the initial state.
- Then can scenario is replayed at will in the host environment, or in the Naca environments. This playback can be manual or automatic, by virtue of the NacaTools web server.
- Screens outputs produced in both execution modes are comparable, and the NacaTools web server provides a mean to automate these comparisons.
- The comparison's results generate a status, reflecting the correctness of the Naca mode execution. Some screen fields can be safely ignored by the automatic comparer. For example, the displayed current date or time is not to be taken in account by the comparison process.
- Database state after execution can be compared to the database state after scenario creation. Thus, the database state after execution in Naca environment must be identical to the database state after scenario capture within the Host environment.
- A playback status is affected to a scenario, giving coverage statistics.

All the scenario playback sequence (database reset, scenario execution by Naca or host environment, comparison of produced outputs, comparisons of database state) can be automated in the NacaTools server.

Thus the following tools are offered:

- CICS scenario registration, using specially modified QWS 3270 terminal emulation software. These executions are targeted to the mainframe computer.
- Automatic or manual CICS scenario playback through this 3270 terminal emulation software, capturing outputs screens. These executions are targeted to the mainframe.
- Manual CICS scenario playback through a web browser, executing programs into the Naca environments.
- Automatic CICS scenario playback through specific open source terminal emulator, executing programs into the Naca environments.
- Automatic comparison of outputs produced by executing a scenario in 3270 mode, or naca mode.
- Before scenario registration and playback, the database can be set-up in a known state.
- After scenario playback, the database state can be checked.
- All these operation can be centrally managed by the NacaTools web-server.

To sum up, showing only screen output comparisons:



6.2. *Batch testing*

Batch programs are tested by using a different pattern.

The following steps are done to test a program, within a application chain.
A complete batch chain test is different from a single program test. This point is not covered in this document.

The first steps for testing a single batch program are done on the mainframe environment:

- Program is identified.
- Program input files are identified.
- Database is set in an known state.
- Program is run on the mainframe, producing displayed outputs and output files.

The input files, outputs files and database initial setup are then stored in the NacaTools website.
This file transfers are done by using a NacaTools option. The files character encoding may be changed from EBCDIC to ASCII at will during the transfer. Care is taken to not converter non convertible fields, for example comp-3 numeric

encoded fields.

Then, at will, the transcoded Java program can be run from the Naca environment.
A run test consists in the following operations:

- Database is setup again in the initial stage.
- The program is launched either form the command line, or through NacaTools. It is provided with file logical names. Files can be either encoded in EBCDIC or ASCII forms.
- After program execution, output produced files are compared to the host originated output files. These comparisons take care of course of the files' respective encodings. That is an ASCII Java produced file can be compared to an EBCDIC host produced file.
- Display outputs are also compared between the two executions mode.
- At last, it's possible to compare the database tables.
- All these comparisons are centrally done by the NacaTools web server, and generated a test status to the program.

Note about database: The mainframe environment implies DB2 usage for Database operations. However, Naca supports Host DB2, or UDB. These two different instances can used at will during program execution in the Naca environment, to check for differences in DB operations, in particular in regards to data EBCDIC / ASCII encoding. NacaRT hides lots of encodings complexities.

7. NacaRT Cobol support

NacaRT supports a lot of Cobol verbs, in a Java as similar as possible syntax form it's Cobol roots.

The various Cobol statements and specificities are described below

7.1. *Variables*

Cobol has a very specific way to deal with variables declarations. These manner are enforced by NacaRT.

7.1.1. Working storage section and linkage section

These sections support:

- Structured typed variable declarations.
- Copy, supporting copy files or Map definition files.
- Copy replacing, supporting copy files or Map definition files.

Copies are not inlined at transcoding or compilation time, but rather are resolved at runtime. The effect is identical from the program owner point of view, but it enables an update of all programs that declares a single modified Copy source file.
The Copy replacing construct enables to replace the level only, not part of the variables names..

7.1.2. Variables

the variable have the following properties:

- They are organized by levels and sub-levels with number ranging form 1 to 49, as in Cobol.
- They can redefine themselves explicitly or implicitly.
- They can be grouped.
- Occurs are supported an any simple or composite variable, with up to 3 indexes. That is NacaRT supports a maximum of 3 indexes. this can be changed if required.
- Levels 77 are not redefinables.
- Levels 66 and 88 are manageable as conditions, which eases their manipulation. This is optional.
- They all have a unique name, within the scope of a transcoding program.
- This unique name is similar to the Cobol original one, except when a name conflict occurs. This happens for example by the inclusion of a Copy file which defines a variable with a name already assigned to a variable. Cobol deals in that situation by requiring the usage of the "Of" scope resolution keyword. In Naca, the transcoder

detects these situations and generates a unique name to avoid these conflicts. This generated name remains however easily readable.
- They are typed. This is detailed in the next chapter.

7.1.3. Supported variable types

The variable supports multiple types, and respects the original default Cobol alignment and padding rule when writing into these variables.

The supported types include:

- Group.
- Alphanumeric.
- Numerical
  - Comp-0 sign or unsigned, with or without decimals.
  - Comp-0 with embedded sign.
  - Comp-0 with separated trailing or separated leading sign.
  - Comp-3 sign or unsigned with or without decimals.
  - Comp-4 sign or unsigned with or without decimals.
  - Numeric edited with mask.
  - Alphanumeric edited with mask.
- Edits. This is an abstraction from the fields of a CICS map, and helps greatly screen handling
- Form, maps and redefinition of maps: These are CICS Map abstractions helping screen handling.

7.1.4. Linkage section and parameter passing

Linkage section of a called program receives arguments passed by caller program by value, reference, or length.

7.2. *Execution flow*

Cobol program uses perform and goto keywords for flow execution. Paragraphs and sections are used to delimiter part of the code. Returns are implicit.
All theses characteristics are fully supported by NacaRT:

- Procedure division declaration.
- Paragraph declaration.
- Section declaration.
- Execution of the program at the beginning of the Procedure division. Parameters can be provided by caller. Then can be passed by value, reference, or their length be specified.
- Perform paragraph, automatically returning to caller when reaching paragraph's end.
- Perform section; executing all paragraph in order until the end of the section.
- Perform through paragraph to paragraph
- Perform until.
- Perform varying.
- Goto paragraph.
- Goto section
- Goback.
- Exit program.
- Exit.
- Stop run.

7.3. *Program types*

All transcoded program are derived from a NacaRT base class. This base class provides some functionality required by application code. However, they all share a common Cobol language support.

- OnlineProgram: CICS program, running in a CICS transaction.
- BatchProgrram: Batch program.

- CalledProgram: It be either a
  - Stored procedure, executed in UDB.
  - Call routine.

7.4. *Operations on variables*

A typical Cobol program manipulates lots of variables. They are described below.

7.4.1. Access

- Variables are accessible by their name.
- A variable may be defined in the working storage section, linkage section, or a copy file.
- If a linkage section variable refers to a variable passed by reference by a caller program, then writing in the linkage section variable modifies the caller's variable.
- A variable can be access by it's definition, or through another one that redefines it. Typing is taken in account however.
- If a variable is part of one or many occurs, then it's indexe(s) must be specified. A maximum of 3 indexes is supported.
- Filler variable reserve memory space that cannot be used directly.

7.4.2. Comparisons

- All comparisons can be done on ASCII or EBCDIC data.
- All combinations of comparisons are supported
- Comparisons can be done between two variables, or one variable and a Java primitive native variable. Type conversions are provided as required by Cobol specifications.

7.4.3. Cobol constant

- High value
- Low value
- Zero
- Space

Constants can be compared to any variable

7.4.4. Conditions support and not-redefinable variables

- Level 88 & 66 declaration supports
- Level 77 is supported.

7.5. *Supported Keywords*

7.5.1. Move

- move from any variable type to any variable type, as enabled by Cobol MVS syntax
- move from java native type to any variable type.
- move from any cobol constant to any variable type.
- move from Edit (or Edit redefinition) to any variable type.
- move from Edit (or Edit redefinition) to any Edit (or Edit redefinition).
- move from variable to Form (or map redefine).
- move from Form (or map redefine) to variable.
- moveAll from any variable type to any variable type or Edit.
- move corresponding.

7.5.2. Initialize

- initialize form.
- initialize any variable type.
- initialize replacing with numeric value.
- initialize replacing with alpha numeric value.
- initialize replacing with numeric edition.

7.5.3. Program calling with argument passed by reference, value or length

- Call.
- CICS XCTL.
- Commarea is supported in online programs.

7.5.4. SQL Support.

- DB2 / UDB support. Some Oracle DB is provided in NacaRT, but not deeply tested.
- Dynamic SQL.
- Cursor Open, close, Fetch.
- Automatic closing of cursors at transaction shutdown.
- Update/Delete current record.
- Tablespace prefix added dynamically. This dynamic tablespace prefix addition greatly simplifies deployment of transcoded program in different DB environments, as this information is not stored in the DB statement, but in a unique configuration file.
- Select, insert, update, delete clauses.
- Sorted Procedure call, passing in/out/inout parameters..
- SQLCode access.
- SQL Diagnostic code access.
- Checking against an SQL Code.
- explicit commit.
- explicit rollback.
- For online programs (CICS), transactions are managed.
- connection pooling, statement pooling is managed for performance reasons.

7.5.5. String handling

- Concat with optional delimiter (variable / space / size).
- digits.
- String.
- Unstring.
- Inspect Replacing / Tallying.
- substring handling:
    - substring extraction.
    - moveSubStringZero.
    - moveSubStringSpace.
    - moveSubStringLowValue.
    - moveSubStringHighValue.

7.5.6. Math

- Add.
- Opposite.
- Subtract.
- Divide.
- Multiply.
- Compute.
- ComputeRounded.
- Inc (increments).
- Dec (decrements).

All Cobol originated variables can be manipulated by these mathematics methods.

7.5.7. Miscellaneous

- AdressOf, enabling pointer support of any variable.
- setting JVM return code.
- Current Date / time support.
- Display.
- Console input / accept.

7.6. *CICS operations*

These operations are reserved to online programs.

- Map support.
- Map redefines support.
- BMS maps are transcoded into XML resource and a Java copy file.
- Simplified handling of map entry fields; they are transcoded in Edits Java objects, with simple access to attributes.
- Pointer support for CWA / TCTUA / TWA.
- Key handling.
- Commarea support.
- Return transaction.
- Abend.
- Condition handling.
- CICS link/XCTL with commarea support and arguments passed by value / reference / length.
- Temporary queue.
- CICS retrieve.
- Return code.
- Send map.
- CICS Start asynchronously.

7.7. *Batch operations*

7.7.1. Sequential file support

- Fixed length records support.
- Variable length records support.
- If the file record length is unknown at runtime, then a deterministic management of fixed / variable length records settings is achieved.
- Files are identified by a logical name. This logical name is mapped to a physical name for Java File system access by various means.
- Files open in input, output, input and output, extended.
- File close.
- Automatic close of open files at program execution end.
- write.
- write from.
- rewrite.
- rewrite form.
- read.
- read into.
- internal sort:
  - sort.
  - release.
  - return sort.
- Files can be read in EBCDIC or ASCII
- Files can be written in EBCDIC or ASCII

7.8. *Batch utilities*

- External sort using same sort criterias as MVS external sort tool. Files can be encoding in EBCDIC or ASCII at will.
- DB2 / UDB Database table loader / unloader / function executor / table transfer from one DB instance to another.
- File encoding Converter (ascii -> ebcdic or ebcdic -> ascii).
- Mail generation.
- File generation support.

7.9. *Stored procedure*

- DB2 Cobol stored procedure transcoded into UDB java stored procedure.
- They respect the specifications of stored procedure interfaces.
- They run in an IBM JRE >= 1.5, in the context of the DB engine instance.
- IBM JRE doesn't supports JMX services, so they NacaRT deactivates JMX in this running mode.
- They are run in a UDB instance.

- They are accessible by any external stored procedure caller, written in any DB compliant language.