Project NACA: automated migration from Mainframe & Cobol to Linux & Java

Didier Durand & Pierre-Jean Ditscheid
Publicitas & Pxedia
Submission #9161
AGENDA

- Project context & history
- Migration strategy
- Technology

June 30th, 2007: mainframe stopped!

NACA = New Architecture for Core Applications
Publicitas

> Advertising sales company for publishers & other media
  – exclusivity of sales for 400+ Swiss titles
  – non exclusive intermediate for 1000s
  – handling of approx 60% of Swiss press advertising
> 2008 turnover: 1.12 billion euros
> 1'650 employees
> presence in 20+ countries (80% of business in Switzerland)

> Didier DURAND, 44, 16 years by PubliGroupe (various positions), currently Head of Architecture & Technology
> Pierre-Jean Ditscheid, 44, 10 years by PubliGroupe, senior Java expert.
Project roots

- Originally, 2 high fixed-costs machine in our IT: network + mainframe
- network fixed by migration to TCP/IP
- our budget = % of company revenues
- revenues decline → IT budget decline
  - down to the point when 100% of budget = pure operations costs → no credits for projects!
  - no projects → no future!

Let's kill high fixed costs by moving to Open Source (Linux + Java)!
Pub2000 - homegrown ERP

> Pub2000
- Homegrown (early 90s) application to handle sales orders (ad insertions in newspapers)
- 100% of source code available
- 1'500 internal users
- 750k transactions / day & 800k pages printed / month (230 types of docs)
- 500 app screens / 1'400 relational tables

> Before
- Mainframe IBM z800 (350 mips) / COBOL / CICS / DB2
- TCP network - tn3270 emulation
- 4 millions lines of Cobol - 2'150 programs

> After
- Cluster of Intel servers with Linux Red Hat with Java / Tomcat / IBM UDB
- Screens automatically migrated to HTML / CSS / Javascript
- 4 millions lines of Java (1to1 conversion) - 2'150 program classes (strong object orientation)
Business case - roots

100% = n * millions CHF/an

> Analysis

> IBM software is the hot spot
> More competition is needed on the platform software itself!
> Platform competition will bring more competition on 3rd party software
> Hardware/ peripherals are not initial priority: let stay on mainframe but with Linux, Java & OSS!

Move to Open Source == 70%+ of cash-outs quasi-cancelled!
Business case - the nice surprise!

Impact of Moore's law has allowed us to get rid of mainframe hardware and replace it by entry level Pentium-based servers. **Additional savings**!
New software landscape - 1

Open Source Technologies

Proprietary Technologies

Tomcat

Apache

XSTLC

Java

Webmin

JUnit

bmcsoftware

Veritas

Tex

Knowledge Tree

Peri

cvs

eclipse
## Software equivalences

<table>
<thead>
<tr>
<th>Legacy</th>
<th>NACA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operating System MVS</strong></td>
<td>MVS</td>
</tr>
<tr>
<td><strong>Transaction Manager</strong></td>
<td>CICS</td>
</tr>
<tr>
<td><strong>Screen Management</strong></td>
<td>BMS</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>DB2</td>
</tr>
<tr>
<td><strong>Batch Control</strong></td>
<td>JCL</td>
</tr>
<tr>
<td><strong>Printing / Output Management</strong></td>
<td>PSF + Archie</td>
</tr>
<tr>
<td><strong>System Admin</strong></td>
<td>TSO, JES, BMC, etc...</td>
</tr>
</tbody>
</table>
Intangible benefits

> In addition to huge cash-out savings, migration to new platform also had big "intangible" benefits for us

- **open source = standards.** By migrating to Linux/Java, we come back into the **de facto** market standard. We can then easily adopt corresponding technological standards: XML, SOAP, SOA, Web/ AJAX, HTML, etc…

- migration → modernization:
  - advanced monitoring tool (Webmin and components) for our operations crew
  - ability to use modern productivity tools for developers: Eclipse, unit testing packages, continuous build, etc.
  - and more….

- many bonuses:
  - Pub2000 is now open and ready to interoperate: Apache Axis implemented to setup an SOA based on legacy code
  - PDF is a no-brainer under Linux (very expensive packages on mainframe!): 100% of millions of pages now generated as PDFs and emailed to customers → Publicitas got greener thanks to Linux
  - Document management made easy through Knowledge tree PHP package: 15+ millions docs now archived and available online!
  - some internal code replaced by OSS equivalent libraries
Driving principles - 1

> Priority 1: savings, savings & savings!
  – Management uninterested by technology debates!
  – Let's speak their language: plan driven only by economies to be achieved
  – Project could auto-finance itself quickly!

> Iso-functionality is a must
  – Avoid all manual "always wanted to do" things that could kill the project
  – only iso-migration allows for 100% automated migration of Cobol source code
  – no implication of end users needed: no need of multiple agendas to be fulfilled

> No big-bang \(\Rightarrow\) "numerous reversible small steps" strategy
  – big bangs always fail!
  – small steps generate at worst small trouble = good acceptance by end-users
  – they show permanent measurable progress
  – disadvantage: slower (but less risky) pace to the project
Driving principles - 2

> Keep the existing crew
  - such a migration can't happen against the "legacy" people
  - trying to oppose the young wolves of OSS to the old crocodiles of mainframe is the worst idea
  - legacy crew knows the old system best: teach them the new set of commands for Linux Java (they want to learn it...) and they 'll deliver the same service afterwards!
  - just inject some young OSS guys to assist and transfer the required competences

> Build the new system bottom up:
  - system engineers can learn OSS on their "own stuff" (printing subsystem, storage, etc...) and get comfortable
  - when they feel good, they are ready to discuss with / support developers and host user applications
Driving principles - 3

> **Full automation is required**
  - such a migration generates tests and re-tests of same thing over a quite long period of time (4.5 years!)
  - Doing all tests manually would kill the project: either by costs or by (bad) risks taken
  - Permanent application build and continuous integration are imperative!
  - Massive black-box testing is required
  - Automation \(\rightarrow\) limited involvement of end-users needed.

> **Be patient & stubborn (4.5 years for us!):**
  - of course, issues arrive! But there is always a solution.
  - "small steps migration" allows for some pauses when an obstacle has to be erased
Why transcoding to Java?

> We could have stayed with Cobol on Linux (Microfocus, Opencobol, etc.)
> So, why did we transcode to Java:

– we quit the **dead end of Cobol**: no more language evolution
– we move to a **modern language**: fully-fledged object orientation
– we go where the mass of developers is:
  - we can recruit more easily
– we enjoy **many improvements** for free:
  - the Java platform is 100% alive and evolves improve
  - Java5 → Java6 significant performance improvements at no other cost than upgrading
– we can get a whole set of **state-of-the-art free developers tools** aimed at highest productivity:
  - Eclipse and plug-ins, code analysis tools, continuous integration tools
– we can integrate tons of **external packages**:
  - to speed up our devs, make our app leaner (basic functions / objects)
  - to make it more open at minimal time & expenses (ex. SOA with legacy combined to Apache Axis)
Why iso-functionality in transcoding?

iso-functionality increases project momentum:
- no specs to define = "same as existing application"
- acceptance is straightforward = "results of new applications must be 100% equivalent to old ones"
- months of discussions / negotiations avoided
- project objective is clearly seen by all parties: no frustration at end!

only iso-functionality allows full automation: no human intervention

automation →
- speed: all application code transcoded as many times as needed
- repeatability: same input → same output.
- homogeneity: equivalent constructs in various programs always produce same java code. No dependence on people's style and/ mood....
- automated testing: some basic unit tests can be generated while transcoding
- agility: you discover a big bottleneck halfway though → just modify transcoder, rerun it on full application → issue solved!
Why did we develop our homemade transcoder?

We run an RFP process in 2003: results unsuccessful:

- nobody really 100% automatic
  → manual adaptations: heterogeneous style and quality
  → cost of repetitions

- syntax of generated Java unsatisfactory:
  → not really readable and maintainable
  → no full leverage of debugging capabilities of Eclipse use (editing, debugging)

- no clean separation between "language" and runtime library:
  → runtime evolution must be independent to have global impact
Transcoder architecture

Pub2000 CICS Inventory
- 780 programs
- 2'200'000 lines Cobol
- 1'930 includes
- 580 DB2 tables
- 500 BMS maps
Migration Path

- **100% of data on DB2**
- **Cobol remains reference**

Java on Tomcat

Data Migration to UDB

- **6-9 months**
- **2-3 months**

Mainframe Switched off

Big Bang Avoidance = Success key!!

Progressive Migration

DRDA connection

CICS

DB2

Activity

0%

100%

Tomcat

In instantaneaous way back to old system
Automated black-box testing

Cumulative effect:
- Several thousand scenarios to be accumulated
- Run every night
- Used after migration when Java further maintained by people
- Very useful to validate system changes
NACA Tools = open source

> In July 2008, Publicitas decided to open its tools: v1.0 transcoder + runtime library released.
  – 1.1 update in March 2009
  – 1.2 update in June 2009 (Oracle support, Cobol extensions, Microfocus file format)
  – 600+ downloads
  – various projects started

> GPL "philosophy": we want improvements and uses by 3rd party professional services to remain open
> License: GPL (NacaTrans) / LGPL (NacaRT): LGPL to allow use by application software publishers

> NacaTrans: transcoder Cobol to Java - 83'000 lines of Java code - 690 classes.
> NacaRT: runtime framework (Cobol verbs, SQL, statements, CICS emulation, multiple optimizations & cache mechanisms, 3270 → html,, etc.) - 153’000 lines of source code source - 890 classes.

> Google Code project: http://code.google.com/p/naca/
> Forum on Google Groups: http://groups.google.com/group/naca-automated-cobol-to-java-transcoding
Key success factors

> A very simple communication toward management:
  - "We will achieve huge savings at the end"
  - "The first savings will allow project auto-financing"
  - Generate lots of intermediate milestones that prove it

> Strategy of "Numerous small reversible steps" was a key success factor:
  - no big-bang
  - permanently visible progresses
  - smaller issues (when they arise. They will!...)

> Iso-functional migration is key:
  - requirements are crystal-clear from day 1 → speed
  - involvement of end-users is minimal
  - 100% automated migration is possible
Benefits

> We save 3+ millions euros / year

> New Intel/Linux system is as stable as but faster than old mainframe system
(Thank you, Moore's law!)

> The platform is our core business application is now state-of-the-art: it is the
basis of a new Pub2000 being currently redeveloped → smooth evolution path

> Our developers enjoy modern tools → better productivity

> We now have a permanent internal disaster recovery center (much cheaper on
Intel servers ....)

> Our architecture allows for horizontal growth by very small increments:
- CPU optimization is no longer a requirement (HR cost of optimization much
more costly than additional server)
- invoicing much simpler: no more consumption-based billing
- the 4-year lifecycle of mainframe (with painful last 6 months) is dead!

> We also succeeded in the migration of system and dev team!
Technical insights

- Naca software components
- NacaTrans: Transcoder
- NacaRT: Runtime library
- NacaTools: Inventory and tests management
Naca software components

> **NacaTrans**: Cobol to Java transcoder (Naca Open sourced).
> **NacaRT**: Runtime library (Naca Open sourced).
> **JLib**: Common base library for NacaTrans / NacaRT (Naca Open sourced).
> **Miscellaneous batch tools.**
  - External sort.
  - DB Import / Export.
  - File encoding conversion …
> **Eclipse NacaTrans plug-in.**
> **NacaTools**: Central web site: automatic tests and scenario registration / replay / comparison.
> **Custom Hllapi extension of QWS3270 emulator for scenario capture.**
> **Open source S3270 command line 3270 emulator for scenario replay.**
> **JMX Administration console.**
> **DB2 / UDB. Commercial license.**

**Bold** items are open sourced.
NacaTrans: Main goals

> Java tool, performing a 100% automatic conversion of Cobol source files to Java.
> Original semantic must be kept.
> Performance must be at least as good as mainframe.
> Generated code must be human maintainable java code.
> Generated Java remains procedural, not object oriented, on purpose to keep Cobol application programmers on board. However, base librairies must be strongly typed.
> Original Cobol code is kept in comments, aligned with java on a line per line basis.

Implementation details are handled by an abstraction layer (NacaRT/ JLib).

E.g. following SQL clause looks like static SQL, while it’s in fact dynamic, jdbc SQL:

```java
sql("SELECT * FROM RS0403 WHERE CDPROJ=#1")
  .into(vrs0403a.dvrs0403a)
  .param(1, tuazone.tua_I_Cdproj);
```

> Java compiler must grant syntax correctness:

```
call(MYSUBPRG.class) is checked by Java compiler,
call(“MYSUBPRG”) cannot.
```
Sample: Generated Java code

```java
ols.getJobInfo(jobInfos, actinfo1, actinfo2);

if (isGreater(lgpar, 0)) {
    if (isEqual(ors_Maj, "INI") || isEqual(ors_Maj, "UPD")) {
        console().display("**IB80001 01 A Param. de maj erronee + " + val(ors_Maj) + " doit etre INIT ou UPD");
        move(1100, comp_Code);
        tools.dumpProgram(comp_Code, dump_Code);
    } else {
        console().display("**IB80001 01 A Param. de maj erronee + " + val(ors_Maj) + " doit etre INIT ou UPD");
        move(1100, comp_Code);
        tools.dumpProgram(comp_Code, dump_Code);
    }
}

if (isSpace(ors_Jnlprno)) {
    move("007", ors_Jnlprno);
}

if (isNaN(ors_Jnlprno)) {
    console().display("**IB80001 01 A Param. no de produit erronee + " + val(ors_Jnlprno));
    move(1100, comp_Code);
    tools.dumpProgram(comp_Code, dump_Code);
}
```

---

**JAZZON09**

The International Conference on Java Technology

June 22-25, 2009 Zurich

---

**publicitas**

---

**netcetela**

---

**Sun Microsystems**
NacaTrans: Technology

> Implemented in java as a multi-pass compiler, taking Cobol/BMS input files, targeting java source code. Passes are:
> - Lexer (intermediate lex file generated)
> - Syntax analyzer (intermediate XML representation of the syntactical graph generated)
> - Semantic analyzer,
> - Algorithmic analyzer,
> - Generator (final .java files generated)

> It uses its own lexer/syntax analyzer, as Cobol grammar is not orthogonal, and CICS/SQL extensions were required.

> Hosts multiple transcoder engines:
> - Cobol program / Cobol copy / BMS Map / Filepack

> Support of dead code detection / removal

> Executable as:
> - Standalone execution configuration in IDE
> - Eclipse plug-in
> - Ant based integration server

> Support conversion of SQL DB/2 - UDB syntax to Oracle (in tests)
NacaRT: Naca’s runtime library

- Java base library used by all transcoded files.
- Same runtime for all program types (online, batch, call, stored procedure).
- Acts for a transoded program as it’s container. Can run as:
  - Tomcat servlet
  - Standalone java application
  - Debuggable Eclipse program
  - UDB stored procedure container / UDB stored procedure container emulator
  - JUnit test
- Doesn’t rely on J2EE application server technology.
- Uses Sun JVM 1.5 / 1.6, or IBM JVM 1.5 for UDB Stored procedure.
- Scales well in multi-core x86 architectures: multi-threading usage, even in batch.
- Caches help to minimizes blocking JVM garbage collections.
- Textual data representation is handled:
  - internally as UTF-16 chars
  - externally as either Ebcdic or Ascii chars (on a case per case basis).
- About 600 classes
NacaRT: core functionalities

> Variable Management:
  – Working storage section, Linkage storage section with complex constructions and multiple redefines
  – Copy
  – Variable declaration, level 1, xxx
  – Multiple variable types (Pic X, pic 9, pic 9 comp-3, ...):
  – Redefines
  – Array access with a maximum of 3 nested indexes
  – Constants: Space, Low value, High value ...

> Operations:
  – Move, comparisons, ...
  – Math operations / String operations, ...

> Control flow management:
  – Section, Paragraph, NextSentence
  – Goto, Perform, Perform through, GotoNextSentence

> Program chaining:
  – Call, CICS LINK, CICS Exec, passing parameters by value or reference.
  – COMMAREA
Naca Online sample outputs
NacaRT: Online / Batch

Online: Runs in Tomcat

> Handle BMS maps as Copy files, and XML files.
> Can hide byte organization of BMS map layout and edit fields, using high level Form/Edits objects, easing fields screen attribute manipulations.
> Handles multi-languages labels.
> Manages CICS transactions.
> Manages user session using multithreaded session pooling.
> Handles html screen management for Online programs. Online programs generate XML data, rendered as HTML.

Batch: Every batch step is run in it’s own private JVM process.

> Files are managed by logical name (cf. DDNames of JCL). Physical path is provided externally.
> Can mix Ascii or ebcdic files (in sync with Java UTF-16 internal representation).
> Internal / External sort is multi-threaded for performance reasons.
> Handles fixed or variable length sequential file access.
> Mutiple file formats: Custom / Micro Focus sequential file format.
> Implements Batch utilities: External sort / Db load / unload / File encoding Ascii / ebcdic
NacaTools: Inventory and test web site

Inventory:
- Legacy programs, files, documents inventory management.

Tests (online and batch):
- Scenario 3270 capture
- Scenario 3270 replay
- Scenario java execution
- Scenario java replay
- Scenario comparisons results between 3270 and java executions
- Database synchronization

Manual transcoder server piloting
- FTP access to mainframe librarian.
- CVS access to get NacaTrans revision and store last version of Cobol files.
- Utilities

Program deployment on all environments

Servers monitoring
NacaRT: Error reporting


De: alert-naca@consultas.ch
À: UDE Support; Cdc Pub2000
Cc: 

nacalLib.exceptions.CESMAbendException: ABend code : D911 : ABend code : D911
Time: 2009.05.15 13:29:14
Program: RS7DM00
Context:
nacalib.basePrgEnv.BaseCESMManager.abend(BaseCESMManager.java:157)
nacalib.basePrgEnv.BaseCESMManager.abend(BaseCESMManager.java:150)
RS7DM00.abandon (RS7DM00.java:7133)

Last SQL Status in error:
SQL Exception (-911):DB2 SQL error: SQLCODE: -911, SQLSTATE: 40001, SQLERRMC: 2 SQLState=40001 | SQLCo
UTIENTC, CLINO, CLICCH, SAIDAT, SAIREF, MUTDAT, MUTREF, CDESTCD, CDEST12 FROM PROD.RS7A01 WHERE UTIE
Params:(2:703660),(1:10005) | Values: | From program=RS7DM00

Current program SQL Status:
SQL Exception (-911):DB2 SQL error: SQLCODE: -911, SQLSTATE: 40001, SQLERRMC: 2 SQLState=40001 | SQLCo
UTIENTC, CLINO, CLICCH, SAIDAT, SAIREF, MUTDAT, MUTREF, CDESTCD, CDEST12 FROM PROD.RS7A01 WHERE UTIE
Params:(2:703660),(1:10005) | Values:

Last Screen data:
<?xml version="1.0" encoding="ISO-8859-1"?>
<form keypressed="" lang="DE" name="rs7ca60" page="rs7ca60" value=""/>
</field name="p3rtrt" updated="false" value="7c0060"/>
</field name="execdate" updated="false" value="13:29:00"/>
</field name="loginc" updated="false" value="PUB-CAP"/>
</field name="dicfra" updated="false" value="L7CJ"/>
</field name="nmage10" updated="false" value="" PUB-
value="703660"/>
</field name="adrnom" updated="false" value="Dipl. Ing. Fust AG"/>
</field name="utien10" updated="false" value="" PUB-
value="002569"/>
</field name="clidch" updated="false" value="010"/>
</field name="prtseq" updated="false" value="14"/>
</field name="prtcdg01" updated="false" value=""/>
</field name="p3rtcdg01" updated="false" value=""/>
</field name="m7abloc01" updated="false" value=""/>
</field name="txtmled01" updated="false" value=""/>
</field name="p3rtcdg02" updated="false" value=""/>
</field name="txt1gn02" updated="false" value="Nie zwei Inserate des Kunden auf derselben Seite platzieren."/>
Thank you!

Publicitas
Didier Durand,
Head of Architecture & Technology
Business engineering
didier.durand@publicitas.com
M: +41 79 212 21 53
More technical insights....
(reserve slides)
NacaTrans: supported Cobol features

- Lots of Cobol verbs …
- Cobol variable declaration, including complex working/linkage section constructions (redefines, filler, multi-dimensional occurs, …).
- SQL: DB/2, UDB, Oracle support.
- Copy are in most cases shared by multiple programs.
- Variable name conflicts is handled.
- Cobol CICS programs with subset:
  - Online / BMS Maps files / Transactions …
  - Link/ Exec / Queues …
- Cobol Batch programs.
  - Files, with logical name support.
  - Internal/ External sort.
- Cobol DB/2 Stored procedure programs: Transcoded Cobol programs runs inside UDB as java Stored procedure, using IBM JVM.
- About 700 classes.
Nacart - Stack for online programs

Online transcoded program

Nacart transactional layer package

Nacart Nacalib package

JLIB

JDBC UDB/Db2 Drivers for us

Xerces XML Parser

Berkley Db for Java Edition

Struts

XSLTC

Tomcat container

Sun JVM >= 1.5, 32 or 64 bits
NacatRT - Stack for batch programs

Batch transcoded program

NacatRT NacaLib package

JLIB

JDBC Drivers (UDB/Db2 for us)  Xeres XML Parser  Berkely Db for Java Edition

Sun JVM >= 1.5, 32 or 64 bits
NacatRT - Stack for UDB stored procedure programs

- Transcoded stored procedure
- NacatRT NacaLib package
  - JLIB
  - JDBC Db Drivers
  - Xerces XML Parser
- IBM JVM 1.5 32 bits
- IBM UDB

IBM UDB
NacaRT internals: Programs management

Program: Code. Loaded on the fly by NacaRT’s internal class loader. Only the maximum number of simultaneously running Programs are stored in memory.

Program instances: Object instances. They are pooled and reused for load time performance.

The 1st time a program is loaded the following operations are done:

> Load and resolve the byte code files.
> Instantiates it.
> Compute variable buffer, variables definitions
> Compute initial value buffer
> Executes the program instance
> When finished, pushes the program instance in pool, ready for reuse.

The program instance, once loaded, is ready for reuse: next executions are much faster:

> Retrieve an instance from pool.
> Reset the variables’ buffer to already computed initial values.
> Execute the program instance.
NacaRT internals: Variables management

> Variable: Represents a storage item declared by Cobol code.
  - Private to a given program instance: single thread access is guaranteed.
  - Strongly typed (currently 35 types). One for each Cobol type (pic X, Pic 9 comp-3, ....).
  - Values are stored in a buffer, owned by the program instance, shared by all variables of it's working storage.
  - Costs only 12 bytes + buffer storage.

> Variable buffer: Value storage area.
  - Private to a given program instance. Single threaded access.

> Variable definition: Defines a variable
  - Shared by all program instances of a given program.
  - Strongly typed, matching Variable's type.
  - Implements a conversion matrix, that efficiently performs all custom data transformation during a move or compare operation.
  - Accessed in multithreaded way.
  - Defines variable properties: Name, position in buffer, length, attributes, ...
  - Separation help reducing memory usage and definitions are computed only once, when the 1st program instance is launched; Cost at least 80 bytes.

> Pub2000 is more than 500 000 unique variable, and uses more than 3000000 variables instances.
NacaRT: buffer management for variables

```plaintext
Var date = declare.level(1).picX(8). var() ;  // 01 DATE PIC X(8).
Var day = declare.level(5).pic9(2).valueZero().var() ;  // 05 DAY PIC 9(2).
Var month = declare.level(5).pic9(2).valueZero().var() ;  // 05 MONTH PIC 9(2).
Var year = declare.level(5).pic9(4).valueZero().var() ;  // 05 YEAR PIC 9(4).
Var sys_Time = declare.level(1).picX(8).var() ;  // 01 SYS-TIME PIC X(8).
Var filler$1 = declare.level(1).redefines(sys_Time).filler() ;  // 01 FILLER REDEFINES SYS-TIME.
Var sys_Time1 = declare.level(5).pic9(5).value(0).var() ;  // 05 SYS-TIME1 PIC 9(5) VALUE ZERO.
Var sys_Time2 = declare.level(5).pic9(3).var() ;  // 05 SYS-TIME2 PIC 9(3) VALUE ZERO.
Var quantity = declare.level(1).picS9(4).comp3().var() ;  // 01 QUANTITY PIC S9(4) COMP-3.
```

Variables’ position in buffer: Each storage unit is 2 bytes long as being an UTF-16 char (this buffer is 36 bytes long)
NacaRT: Performances

> Caches, caches ... lots of caches, and fine tuning ...:
  
  – Program / Program instance.
  
  – Variables, dedicated to a single Program instance.
  
  – Variable definitions, shared among all same Program instances.
  
  – Initial values shared among all same Program instances.
  
  – Move corresponding source and destination.
  
  – ... Lots more ...

> Blocking Garbage collection avoidance:
  
  – Concurrent GC was problematic
  
  – Many local variable are stored in long term collections to avoid garbage collections, as some GC operations are blocking, suspending all processing for a few seconds, including screen generation.

> Mainframe DRDA performance limits are managed during migration phase:
  
  – SQL connection pooling.
  
  – SQL statement pooling.
  
  – SQL variable to column mapping.
  
  – ...

> 64 bits JVM is critical for code and data available memory space.
NacaRT: Online

> Handle BMS maps as Copy files, and XML files.
> Can hide byte organization of BMS map layout, using high level Form objects.
> Can hide byte organization of map fields, using high level Edit objects. Edits eases fields screen attribute manipulations.
> Handles multi-languages labels.
> Manages CICS transactions.
> Online program can launch asynchronous transactions.
> Manages user session using multithreaded session pooling.
> Handles html screen management for Online programs.
> Online programs generate XML data, rendered as HTML.
> XML data can be used for scenario replay tests.
> Authentication with Ldap.
> Javascript/HTML emulates 3270 with all PF Keys.
> Javascript is sent to browser to limit bandwidth and support PF Keys, including F1.
NacarT: Online data flow

Browser

Servlet

Action Compatibility

Session manager

XML Data IN

Sequencer

XML Data IN

DB

XML Data OUT

Routing info

Transcoded CICS programs

Servlet

List of (name, value)

HTML + JavaScript

Finalized XML

XML Data OUT extended

Merger

Standard Presentation layer

XML Data OUT

Finalized XML

XLT C

XML Structure
NacaRT: Batch

- Every batch step is run in its own private JVM process.
- JVM is launched by command line.
- Files are managed by logical name (cf DDNames of JCL). Physical path is provided externally.
- Can mix Ascii or ebcDIC files (in sync with Java UTF-16 internal representation).
- Support file generation.
- Internal / External sort is multi-threaded for performance reasons.
- Handles fixed or variable length sequential file access.
- Multiple file formats:
  - Custom
  - Micro Focus sequential file format.
- Implements Batch utilities:
  - External sort.
  - Db load / unload.
  - File encoding Ascii / ebcDIC conversions.
- Step scheduling uses external scheduler (Control-M - commercial license).
- JCL is transcoded in Linux KSH by existing third part utility.
NacaTools: Test during migration

Comparison between 3270 screens with HTML outputs. Both environments can execute the same application through a registered scenario. Results are comparable to check conformity.

NacaTools server

Transcoded programs repository

Execution server

MainFrame

3270 Flow -> XML

Workstation executing a modified 3270 terminal emulator. It can:
- register a scenario
- execute a registered scenario
- execute manual screens

Screen Content storage

Registered scenarios

Comparison of the 2 XML

Naca Tools
- Tomcat
- Java >= 1.5

Workstation running a Web browser executing transcoded programs through registered scenario or manual commands

Report of the differences between 3270 / HTML flows

Naca framework

Open source libraries

Java >= 1.5

X86 Server

Transcoded programs repository

DB2

Access to DB2 through DRDA

DRDA

X86 Linux 32/64 bits Server

3270 Flow -> XML

NacaRT

J2EE framework

Open source libraries

Java >= 1.5

Servlets

Java, J2EE

Sun Microsystems

J2ME

3270 Flow -> XML

Open source libraries

Java >= 1.5

Servlets

Java, J2EE

Linux 32/64 bits Server

3270 Flow -> XML

Open source libraries

Java >= 1.5

Servlets

Java, J2EE

Naca framework

Open source libraries

Java >= 1.5

Servlets

Java, J2EE

NacaRT

Screen Content storage

Registered scenarios

Comparison of the 2 XML

Naca Tools
- Tomcat
- Java >= 1.5

Workstation running a Web browser executing transcoded programs through registered scenario or manual commands

Report of the differences between 3270 / HTML flows

Naca framework

Open source libraries

Java >= 1.5

Servlets

Java, J2EE

NacaRT