MODERNIZATION OF THE EUROCAT AIR TRAFFIC MANAGEMENT SYSTEM (EATMS) ¹

Jerome DelaPeyronnie  Philip H. Newcomb  Vincent Morillo
Program Manager  Chief Executive Officer  FDP Senior Scientist
FDP Product Manager  The Software Revolution, Inc  Thales Air Systems
Thales Air Systems  11410 122nd Ave NE, Suite 304  105, Avenue Eisenhower31087
105, Avenue Eisenhower31087  Kirkland, WA 98034  TOULOUSE Cedex

Luo Nguyen  Mark Purtill
Operations Manager  Senior Scientist
The Software Revolution, Inc  The Software Revolution, Inc
11410 122nd Ave NE, Suite 304  11410 122nd Ave NE, Suite 304
Kirkland, WA 98034  Kirkland, WA 98034

ABSTRACT

This case study describes the specific modernization technology and processes employed by Thales Air Systems S.A. (Thales), a leading European and major global provider of ATM systems and its modernization solution provider, The Software Revolution, Inc, to modernize its air traffic management systems into EATMS from 2005 through 2008. The systems discussed in this paper commenced with an initial pilot project in 2005 that assessed the transformation validity and determined whether real-time Java or C++ would be the preferred target language.

After confirming the suitability of Java as target language, high economies of scale and schedule compression were achieved when TSRI’s architecture-driven modernization approach was applied across three Ada83 variations of the Eurocat product-line comprising in total approximately 1.6 million lines of code. A two-fold improvement in code and design quality metrics as measured by key design metrics was achieved during the transformation of the system.

Introduction

This paper summarizes Thales Air System’s and The Software Revolution’s non-distortive model-based architecture-driven modernization of Eurocat, Thales Air System’s fully proven

¹ This paper is extracted in part from Chapter 5 of Information System Transformations: Architecture Driven Modernization Case Studies, William Ulrich and Philip Newcomb, Morgan-Kaufman, 2010.
air traffic management system that operates at 280 air traffic control centers worldwide. Eurocat modernization was undertaken as part of the European Air Traffic Management System (EATMS) modernization initiative. Originally written entirely in Ada 83, Thales recently achieved acceptance for the deployment of jFDP, a new version of Eurocat’s flight data processing (FDP) subsystem written in pseudo Real-time Java FDP. The first site acceptance of jFDP was approved in March 2010, and a first commissioning is planned for February 2011. Customers approved all formal factory tests and site deployment is proceeding per plan. Upon deployment, Eurocat will be the first FDP written in Java ever put in ATC operation. Operational commissioning of other European sites is planned thru 2012.

Thales Air Systems is a world leader in the development, operation and maintenance of mission-critical information systems for the aerospace, defense and security markets. Thales’ Eurocat air traffic management system, Eurocat written in Ada 83, is a fully proven air traffic management system that has demonstrated its operational capabilities in 16 European countries Australia, Asia and Africa. Eurocat is suited for applications ranging from the management of single approach or tower to complex en-route control in transcontinental, oceanic or very high-density environments. Thales determined that replacement of the Eurocat FDP Ada language with Java was pivotal to upgrading Eurocat to meet EATMS’s new technical standards to enable rapid conformance with new ATC systems and regulations modernization requirements. EATMS is the key starting element of SESAR project, the European initiative that is roughly equivalent to FAA “Next Gen” (NGATS) in scope, defining new concepts in avionics and air traffic management to cope with expected traffic growth and safety requirements at the 2025 horizon.

Because the risk of manual error was unacceptable, fully automated modernization of the Eurocat Ada 83 FDP into Java was imperative to assure Eurocat system performance, and safety for the modernization of this mission-critical real-time system that manages over 10 million passenger flights annually. After evaluating alternative modernization solutions, Thales chose The Software Revolution Inc. (TSRI)’s JanusStudio™ automated modernization capability as the most effective solution for meeting their modernization objectives. The motivation for the choice of this method was the assurance of perfect functional replication of the existing software in a new language and platform, a capability that was provided flawlessly by TRSI’s JanusStudio™ automated architecture-driven modernization solution.

Safety evidence, demonstrating the transformation process to be non-distortive of original functionality, convinced Thales customers to accept modernization by means of an automated process. Central to the success of project was the metrics-guided modernization process and technology that maximized the utilization of the skills and technology of a multi-national team through continuous iterative perfective metrics-driven adaptation of tools and methods. Key to customer acceptance was the 100% automated code transformation process, and iterative semi-automated perfective refinement and reengineering of EATMS system design and architecture to meet the rigorous and exacting performance and architecture requirements of the next generation EATMS.
Established in 1995, TSRI has its roots in projects at the USAF Knowledge-Based Software Assistance Program and the Boeing Artificial Intelligence Lab. The company's JanusStudio™ technology is used in legacy modernization projects to transform existing applications written in a variety of procedural languages into object-oriented software written in Java/J2EE, C++ or C# .NET. TSRI has completed more than 80 modernization projects, mainly for customers in the government and defense industry. TSRI undertakes modernization projects using the JanusStudio™ technology to achieve a highly automated iterative modernization process, which includes documentation, transformation, refactoring, re-design, re-architecture, and web-enablement into modernized systems. JanusStudio™ is typically used for modeling and transforming systems in their entirety through a series of adaptive and perfective refinement iterations of transformation rules and executable refactoring specifications. The source and target AST representations provide detailed “As-Is” and “To-Be” documentation that support multiple design methodologies, including Unified Modeling Language (UML), Object-Oriented Analysis and Design (OOA/OOD), Business Rule Modeling (BRM) and Structured Analysis and Design (SASD).

TSRI technology and process achieve fully-automated transformation of the software, platform adaptation, as well as fully machine mediated iterative perfective refinement to optimize code structure, design and architecture. The quality of code achieved generally exceeds human software in quality (as measured by key quality metrics), correctness (as measured by incidence of errors) and producibility (speed, accuracy and completeness). TSRI offers engineering and testing services to adapt and tailor its services and products of to the specific needs of individual projects. TSRI is a member of the Mainframe Migration Alliance, the Architecture Driven Modernization group at OMG, the Enterprise Re-Architecting Consortium and the Natural Modernization Strategies Information Forum (NAMS-IF).

**EATMS OVERALL PROJECT OBJECTIVES**

The core function of this system, the Flight Data Processing system is the most complex function of the Eurocat system, and the one that is deemed to change dramatically for the next generation ATC systems that will gradually transition from radar based systems to flight plan trajectory based systems. The FDP transformation was the main project undertaken by Thales and TSRI. The key business drivers of these initiatives included:

1. Management of the obsolescence of the Ada software language and recruitment of software engineers,
2. Modernization of development method & tools,
3. Improved code safety and quality owing to code rejuvenation,
4. Code-base consolidation and preparation for a migration to next generation Eurocat, that includes CORBA based middleware (CARDAMOM) and Java-based model-driven development tools for the evolution of these systems

The transformation strategy allowed a faithful code replacement of the FDP system, by creating a perfect “replica” of the existing operational ones in Java. The replication of existing application in new language and platform was undertaken in advance of any operation and
functional changes with the prime objective being to cleanly separate all risks of technological changes from the developmental risks associated with functional evolution. After considering the alternatives of manually rewriting or redevelopment of FDP from scratch, Thales decided on automated modernization of these systems into real-time Java as the most effective approach to facilitate meeting these objectives.

The rapid ascendance of Java as the most widely used object-oriented programming language is depicted in Figure 6-1. The ascendance and dominance of Java factored heavily into Thales’ decision to migrate its applications into real-time Java, but not before Thales and TSRI performed a detailed pilot project oriented to determine the optimum real-time target language of choice --- C++ or Java.

![Figure 6-1: Real-Time Language Usage Application History (source: AONIX)](image)

EATMS Modernization Pilot Project

To begin the EATMS modernization process, Thales undertook a pilot project with TSRI in early 2005 targeted at investigating the feasibility of automated language transformation, supported by a qualitative and performance assessment of the generated code. Key objectives of the pilot project were to:

Draft 5/20/2010 | Copyright 2010 The Software Revolution, Inc
(1) Ensure functional non distortion of produced Java/C++ code versus ADA legacy code

(2) Assure the generated code complied with coding safety rules and established coding standards;

(3) Assure the documentation produced by the tools allowed detailed comparison between the source and target code.

(4) Assessment of whether Java or C++ could meet EATMS performance objectives.

(5) Predefine refactoring schemes for the code transformation

(6) Assess chances of success for the full transformation of the FDP process.

The target system selected for the pilot project was a high-performance real-time Ada 83 application, the Trajectory Prediction Manager (TPM), which consisted of 17,000 lines-of-code. TPM was a piece of FDP not too closely linked to the rest of the software. It was highly representative of the FDP application because of its algorithmic complexity (predictions of speed laws and vertical evolution of the aircraft), its CPU demand and its functional key role in the Eurocat system.

The initial phase of the pilot, which included adaptation of TSRI translation process to translate TPM code into C++, generation of a hyper-model of the “As-Is” and “To-Be” system to support code inspection, and engineering support throughout the testing process, was completed in five (5) weeks. The transformed code was manually inspected and the target C++ code was determined to be easy to read, and easy to modify.

During performance testing the C++ code was integrated with existing FDP Ada processes, integration tested and within few days found to be functionally equivalent in the full Eurocat environment. Critical performance testing included demonstration that TPM trajectories generated by the C++ were identical to those produced by the Ada. Graphical display of trajectory outputs allowed a quicker and easier comparison of the two sets of sources:

No more than five bugs were detected in the translated C++ due to translation errors. These bugs were corrected by modifying the transformation rules and the code was retranslated and delivered by TSRI to Thales. TSRI subsequently adapted the transformation process to generate the Java version of TPM from Ada 83 in two weeks, delivering the Java version of TPM and the TSRI hyper-model “As-Is” and “To-Be” documentation upon completion of the transformation.

After the assessment the TPM software in C++ Thales contracted with TSRI to proceed with the transformation of TPM into Java. The Java version of TPM was produced in 3 weeks following delivery of the C++ version of TPM. Since integration between Ada with Java and C++ modules within the FDP test bed was not yet seamless at the time of the pilot, both the
Java and C++ versions of the TPM module were tested using stubbed flight plan routes and ATC constraints for two different Aircraft types and tested in the Eurocat test environment. During testing, trajectories were found to be identical between Java TPM and Ada TPM, with only 10-15 bugs detected in the Java translation, slightly more than with C++, with the difference attributable to the Ada to C++ transformation rules being slightly more robust by virtue of having been used on a greater number of previous projects. As with the C++ TPM, the Java TPM was corrected by modifying transformation rules and then retranslated, achieving a 100% automated and error free transformation.

Overall the transformation from Ada to C++ and Java resulted in fewer that .1% incidence of errors in the initial transformation and 100% correct transformation after modification of the transformation rules. TSRI engineers typically accomplished corrections of the C++ transformation rules and its retransformation and delivery within a 24 hour cycle from notification of an error. TSRI engineers accomplished correction of the Java transformation and its retranslation and redelivery within a 24-hour window. TSRI’s ability to perform problem identification and resolution within 24-hour window during the pilot engendered confidence that TSRI’s technology and process could support the full Eurocat conversion.

During the initial performance tests of the C++ TPM, without any code optimization, performance was ten times slower than the Ada TPM. A probe package was subsequently used to identify areas of the C++ code requiring optimization. One significant but very simple optimization was found and implemented for the C++ TPM. Following this optimization the C++ TPM performed three times better than the original non-optimized Ada. The same optimization was then performed on the Ada, following which the optimized Ada TPM outperformed the optimized C++ TPM, but by a reduced 1.66 to 1 performance ratio. The Ada optimization was applied only to the TPM Module. The rest of the Eurocat Ada code was not optimized. The response time of the optimized Ada TPM module was improved by a factor of three compared to the original.

Both the C++ as well as the Java transformations of TPM produced faithful replicas of the original Ada 83 TPM implementation. There was very little functional distortion of TPM. The initial transformed code was not fully object-oriented; however, this was expected due to the absence of refactoring during the TPM pilot. The code was assessed to identify the kinds of refactoring steps and operations that would subsequently be incorporated into refactoring specifications developed for the full conversion.

The Transformation Blueprints™ of the “As-Is” and “To-Be” hyper-model design documentation permitted Thales engineers to carry out detailed inspections of the code-level translation permitting line-by-line comparison of the Ada and the C++, and design-level comparison between graphical data-element tables, structure charts, control-flow, state-machine, cause-effect models of the hyper-text code model of the Ada and the Java and C++. A representative set of Thales specific Java coding rules were defined, and the TPM pilot code was visually inspected using th Transformation Blueprints™ to verify that the transformation process had correctly implemented them.
Performance Following Code Optimization

To address the Java start-up and response time instability, the Java TPM was optimized using the Excelsior JET compiler. Upon completion of this optimization effort the optimized Java code achieved performance comparable to the optimized C++, and both the optimized Java and the optimized C++ achieved overall performance time superior to the original Ada. The excessive start-up time for Java was eliminated and the excessive variability in the response time for Java was eliminated. The time performance ratio of the optimized C++ TPM to the optimized Java TPM was 1.09 to 1. As shown in Figure 6-2, controls imposed on garbage collection behavior enabled the Java TPM to achieve performance levels superior to the non-optimized Ada TPM that were acceptable to the FDP function. With optimized compiler and garbage collection settings (GC and AOT compiler) the real-time Java gave comparable performance to the C++, and provided acceptable response times.

Figure 6-2: Optimized Performance Call Duration Times

PERFORMANCE FACTOR

The final time performance comparison between the original Ada and the optimized Ada, C++, and Java is shown in Table 6-1.

<table>
<thead>
<tr>
<th>Language</th>
<th>Optimized Ada</th>
<th>Optimized C++</th>
<th>Optimized Java</th>
<th>Original Ada</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Performance Ratios</td>
<td>1</td>
<td>1.66</td>
<td>1.81</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 6-1: Final Language-based Time Performance Ratios
When considering these performance results it is important to recognize that the TPM code was designed for the Ada language. As a consequence there were many code-level decisions made in the implementation of TPM that favored the Ada language over the C++ or Java languages. For example, Ada uses fixed length arrays while both the C++ and Java used resizable arrays implemented as dynamic vectors or lists. While correctable, these translation choices were decided by the default behavior of the translation. Subsequent phases of the JFDP project addressed many of these low-level translation choices by modifying transformation rules to optimize the choice of code-level representation.

**MEMORY UTILIZATION & OPTIMIZATION**

The memory utilization ratio of Java to C++ was 4 to 1 as shown in Figure 6-3. It should be noted when considering the Java memory performance that inconsistent time performance response and high initial levels of memory utilization were corrected by tuning during optimization.

![Figure 6-3: Memory Utilization Ratio](image)

Before optimization, the use of a Just-in-Time (JIT) Java compiler was used. This resulted in long response times preceding the first call to any function for compilation. By transitioning to the use of an Ahead-of-Time (AOT) compiler, this up-front time performance degradation was eliminated altogether. The impact of garbage collection was the primary cause of inconsistent response time. After a long garbage collection, response time decreased.

**PERFORMANCE BENCHMARK CONCLUSIONS**

A key objective of the pilot was to benchmark the performance and suitability of C++ and Java using code that was sufficiently representative of the ATC systems operations to support the choice of language for Eurocat modernization that would meet all application performance requirements as well as EATMS and FAA NGATS directives regarding software quality assurance, tools and methods and project objectives to improve code quality, code sustainability and maintainability.
Key Conclusions of the Pilot

The TPM pilot was undertaken at the time of a choice between C++ and Java for future FDP applications (EATMS projects). The TPM outputs proved that the Java language could be used for FDP kinds of applications, provided application performance was monitored and tuned to meet performance requirements. Another key conclusion of the TPM project was that there was a need to select an appropriate Java Virtual Machine (JVM) compatible with performance requirements. The transformation rules should also be adapted to introduce code-level optimizations tuned to the target language to improve the target language response time. In addition, for each application, specific GC tuning was found to provide best results, and any significant modification in an application should be assumed to require new GC adjustments. A key finding of the pilot regarding the Java garbage collection was that with tuning, response time dispersion could be optimized, and functional performance of tuned Java was well within constraints for application in the ATC domain.

The outcome of the pilot was the selection of Java for the Eurocat modernization and the selection of Java as the primary language for use in new ATC applications.

FULL EATMS PROJECT MODERNIZATION OBJECTIVES & CONSTRAINTS

After Thales and TSRI completed the pilot project, the team undertook the modernization of the Thales FPL (refer to § Appendix A) system. A key objective of the transformation of Eurocat was to achieve exact functional replication of Flight Data Processing (FDP) systems original Ada 83 code into real-time Java code. Perfect functional replication of JFDP (Java FDP) at the code level was needed to assure that the FDP software specifications (SSS/SRS) were unaffected by the transformation and remained strictly identical to the original. This requirement ruled out a human translation as the consistency of a human translation could not be guaranteed. The use of automated rule-based transformation guaranteed both uniformity of the translation, as well as high level of assurance that deficiencies when detected would be uniformly corrected.

To assure the consistency of translation between all language elements of the source Ada 83 and target Java code, it was essential that the translated code be obtained by employing a highly automatic process in which at least 99.5 % of code could be translated fully automatically. In addition to code translation, code refactoring was employed to improve code quality and the software’s design to meet EATMS design requirements so that the modernized code would serve as a foundation for future system enhancement. These objectives were achieved through the use of model-based transformation technology from TSRI that would ultimately achieve a 100% automation level by encoding the entire translation and refactoring process as rules in TSRI’s rule engine.

Another key requirement was for the interfaces of Java FDP with other systems to remain strictly identical to the original Ada 83. This was essential for the Eurocat baseline, which
involved integration of Eurocat and ADS/CPDLC functionalities. In particular, there could be no change to the interfaces of the FPL system with other Eurocat systems. The introduction of JFDP in Eurocat was to be done smoothly at no cost for other CSCIs. In addition, the JFDP had to interface with other existing libraries, in particular the C libraries (kinematics) that were called through JNI, preserving legacy ADA interfaces and the data and parameters files, prepared off-line, that are widely used for FPL processing.

Finally, the following were key JFDP refactoring constraints that had to be met by the modernization project:

1. JFDP code and inner interfaces needed to be restructured, while preserving functional behavior;
2. JFDP needed to easily be adapted to new interfaces;
3. JFDP must be able to load any dataset (refer to § Appendix A) distributed on the system;
4. JFDP must interact with other ADA processes as if JFDP was written in Ada;
5. JFDP needed to be monitored, restarted, and switched just like in Ada; and
6. FPL Xtermlogs (traces) of JFDP must be identical to the original logs.

An automated transformation followed by automated and repeatable incremental refactoring process was vital to achieving technological change while preserving full Eurocat functionalities. Early in the project manual re-coding was investigated and quickly discarded as a long, expensive and error prone process. Feasibility studies and pilot projects undertaken in 2005 proved automatic transformation to be the most promising way to produce JFDP, and answered questions pertaining to fine tuning the process and technology used in the project. Incremental refactoring was combined with transformation achieving a step-by-step iterative process that could be repeated as often as needed and that would validate each step of that process through the use of automated CSCI tests.

EATMS REFACTORING QUALITY METRICS

Significant improvements in quality and safety aspects in the code were achieved due to refactoring-related modularization of the JFDP code. The removal of dead code and global variables produced smaller easier to maintain procedures and classes. Refactored modules achieved consistent content and produced modularized code that was easier to read and understand. The inter-module dependencies were reduced decreasing the side effects that are considered when analysing a module. Global behavior is more easily understood by means of the transformation of global variables into members of classes.

Impact analysis was equally simplified reducing the analysis time and cost associated Engineering Change Requests (ECRs) and Problem Correction Reports (PCRs). Implementation of code modifications was better controlled due to improvements in quality metrics and integration of UML models into maintenance tools and procedures. In addition, the technical sophistication of tools improved through the use of Eclipse, Junit, Jprofiler, Emma, Checkstyle, findBugsKlocwork and the integrated of hyper-model of design documentation and code from TSRI. These tools were significant improvements over the Ada-oriented tools logiscope and rulechecker, which they replaced. These transformation benefits and the tools
built around Java were key elements to convince ATC community of the value of Java for operational ATC applications.

Figure 13 documents an average factor of two (2) improvements on key code quality metrics that were achieved by the modernization of EATMS into Java.

<table>
<thead>
<tr>
<th>Ada Metrics (FDP)</th>
<th>Java Metrics (JFDP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of unwanted module dependencies</td>
<td>~1000</td>
</tr>
<tr>
<td>Number of CSCs Levels in hierarchy</td>
<td>2</td>
</tr>
<tr>
<td>Number of Ada Packages:</td>
<td>1006</td>
</tr>
<tr>
<td>Average Lines of Code per Package:</td>
<td>500</td>
</tr>
<tr>
<td>Number of Ada Sub-Programs</td>
<td>3511</td>
</tr>
<tr>
<td>Average Lines of code per procedure</td>
<td>52</td>
</tr>
<tr>
<td>Number of Large Ada Packages</td>
<td>1500+ lines of code: 57 violations</td>
</tr>
<tr>
<td>Number of large Ada procedures</td>
<td>150+ lines of code: 269 violations</td>
</tr>
<tr>
<td></td>
<td>Number of module dependencies</td>
</tr>
<tr>
<td></td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Number of Package Levels in hierarchy</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Number of Java Classes:</td>
</tr>
<tr>
<td></td>
<td>1976</td>
</tr>
<tr>
<td></td>
<td>Average Lines of code per Class</td>
</tr>
<tr>
<td></td>
<td>195</td>
</tr>
<tr>
<td></td>
<td>Number of Java Methods</td>
</tr>
<tr>
<td></td>
<td>15669</td>
</tr>
<tr>
<td></td>
<td>Average lines of code per method</td>
</tr>
<tr>
<td></td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Number of large Java Classes</td>
</tr>
<tr>
<td></td>
<td>1500+ lines of code: 28 violations</td>
</tr>
<tr>
<td></td>
<td>Number of large Java Methods</td>
</tr>
<tr>
<td></td>
<td>150+ lines of code: 121 violations</td>
</tr>
</tbody>
</table>

**Figure 6-4: Post-Modernization Code Quality Metrics Comparison**
JFDP Modernization Test Strategy

Test Strategy Principles

The main goal of the JFDP test strategy was to ensure the faithfulness of the transformation process applied to the ADA code from functional point of view, and assure that the real-time requirements of the FDP system, as well as its integrity and degree of service, were maintained when FDP was reimplemented from Ada into Java. In the absence of any intended functional change, the focus of regression testing is system-wide testing to achieve a wide scope of coverage under a broad set of realistic test conditions rather than unit level testing to achieve deep validation of individual units of the system. Unit testing is indicated when functions are being developed for the first time and the full range of required behaviors must be exercised and validated. The JFDP modernization scenario functions did not involve any changes to FDP functionally, therefore stand-alone unit-level testing, which segregates the focus of the validation effort on repeated executions of the same lines of code is not a sensible use of testing resources.

Four (4) major steps of the JFDP qualification were performed:

1. CSCI tests, involving only the FDP function in a stand-alone mode addressing functional requirements testing
2. Integration tests, involving complete system platform focusing on FDP interfaces, system behaviour and performances
3. Validation tests, involving complete system platform focusing on functional requirements testing
4. Shadow tests, involving customer tests in live ATC environment and real controllers

CSCI Tests Principles

A highly level of automated testing was achieved for JFDP modernization and tightly coupled with the transformation and refactoring processes. The Eurocat FDP test bed provided the means to exercise both the Ada and Java code in complete ATC simulations using data from live airspace flight data processing environments. Following every major transformation and refactoring cycle, 12 distinct test dataset were used to simulated execution of JFDP under a battery of rigorous flight scenarios using data captured from live and simulated flight data processing scenarios.

A tool named Airspace allowed first the automatic live replay of existing FDP tests scenarios, built to test functional requirements of the FPL system. Then another Airspace module was designed to extract the JFDP outputs and allow a smart difference between two distinct runs of the same scenario. In a full test cycle, 272 CSCIs were unit-tested using an eight (8) hour integrated system test of the modernized system. Detailed functional and time performance comparisons were made using the test performance log files that permitted side-by-side comparison of Ada and Java. The exact same test sequences were performed for both the Ada
and the Java, and each Java test sequence output was compared to the Ada test reference sets to assure non-functional distortion between the Ada FDP and JFDP. This set of CSCI tests combined with automatic comparison of the outputs of Ada and Java test runs was the key to achieving confidence in the transformation process and providing evidence of the rigor of the process to external auditors.

**INTEGRATION TESTS PRINCIPLES**

Integration tests for the legacy FDP system were selected and run both in ADA and Java on a system platform. These system integration tests were selected with the following criteria:

- Specific impact of the translation on Interfaces: test wrappers around JFDP interfaces.
- Assurance that interactions with other CSCIs are still running correctly, in particular tests for safety requirements.
- Degraded mode tests.
- Performances tests (load test, time to start, time switch).

Test wrappers were used around JFDP interfaces. A set of wrapper scenario-specific tests were performed to ensure all JFDP wrappers were executed for all transformed modules, and that they were interoperable with Ada modules in the test environment. A complete range of interface tests were performed to assure that all CSCIs would run and interact with JFDP correctly.

**SYSTEM VALIDATION TESTS**

In addition to CSCI and integration testing, additional validation system tests were selected to extend the range of testing to functional areas that were not covered by the live data simulation testing, using outputs of EMMA tool after CSCI tests run. Results were used to design CSCI-specific tests and to create JFDP system tests required for safety analysis. Validation tests that were relative to requirements flagged “Safety” were also selected in priority.

These tests were run first on the system platform in ADA and then in Java.

Combining live data simulation with specially designed CSCI-specific validation tests facilitated rapid convergence on all transformation and refactoring issues. Tests results brought a high level of confidence to the Java transformation and refactoring processes, confirming the expectations of the Pilot Project. This disciplined process combined a well-structured refactoring plan along with a rigorous automated validation process that facilitated rapid identification and correction of any issues resulting from each iteration and asymptotic issue incidence throughout the 10-month duration of the project on initial Eurocat baseline.

Tests status results on initial Eurocat baseline were:

- 100% tests OK for 272 automatic tests;
- 100% tests OK on 27 integration tests, and;
- 100% tests OK on 99 validation tests procedures.
As shown in Figure 6-5, the total number of bugs (85) was very small for a project that eventually involved the transformation of 1.7 million lines of code, a per LOC error incidence of .00005%.

![Figure 6-5: Test Results Bug Reporting](image)

After presentation of these positive results there was a decision made to launch the ultimate step of JFDP qualification: testing it with live air traffic data with real controllers acting as if they were controlling the airspace with the system. This is called shadow testing in ATC world. The objectives of the live operational tests were two-fold:

1. Ensure the non functional distortion proved in the laboratory was still also proved in live testing on customer site
2. Observe the system behavior with long-term running proceed to endurance tests to prove that Java technology was able to cope with real life of Air Traffic Control environment.

This step was key to prepare the safety folder for certification authorities and ensure that both refactoring process and Java technology could be used for ATC operations. The JFDP system was installed several months and tested by some customer experts, doing in particular specific functional tests and performance tests. On top of these tests procedures controllers of several countries were invited to work with the system in a two days shadow session.

The results of these live tests met the expectations and proved that both JFDP system and Java technology were usable. Only 3 transformation bugs were raised (just in the first couple of days) and only two problems were due to Thales wrappers. There was no crash, no loop.
occurrence and no problem linked to the JVM. Thousands of hours of JFDP running were performed and a five days duration of non-stop usage. The two days of shadow testing managed 2700 flight plan creations and allowed 7 controllers to use the system as if it was in Ada. There were no response time performance issues was due to Java.

**KEY FINDINGS**

A key focus of the EATMS project was to assure JFDP software performance satisfaction of Eurocat strict run-time performance requirements. During testing to-date there have been no Java Virtual Machine crashes, no crashes in the Java code, and no performance/response time issues. The times to start, restart and switch the system were far below 1 minute. 99% of FDP responses times are below 1 second even during load tests and the highest CPU consumption is approximately 30-35% for Dual Core PC.

More important Java side-effects associated with garbage collection, Just-in-Time compilation, and memory consumption have been mastered and it has been eventually proved that JFDP software complies with all Eurocat performance needs. The margin of final response times is such that even initial processing that is slow downed by J.I.T stay below the required limit. In the same manner the duration and frequency of Garbage Collection didn’t preempt CPU enough to disturb JFDP processing, because memory size was optimized and because JFDP process loads a lot of classes at initialisation, optimizing the memory variations. The fact that Garbage Collection and Just-In-Time compilation effects were still below the requirements contradicted the initial fears raised during the Pilot Project.

Use of Java technology requires more careful handling of performances and optimization of memory, but there is unarguable evidence that it is possible to operate a central core server like FDP in a real ATC environment.

**PROJECT CONCLUSIONS**

The EATMS Ada to Java Modernization Project was an exceptionally sophisticated and highly disciplined application of automated transformation and semi-automated refactoring against a high-performance mission-critical software application. Key to the success of the EATMS modernization project was the tuning of the software technical and management processes to achieve a metrics-guided modernization process that maximized the utilization of skills and technology from all members of the team, with continuous perfective adaptation of the tools and methods to achieve and maintain a high-level of automation in all phases of the project. Detailed attention to process, methods and technology permitted re-application of modernization methods and tools that achieve high economies of scale when applied to multiple related systems in Thales’ Eurocat product-lines.

**PROJECT AFTERMATH**
Re-factoring has continued on the software since the original transformation, mainly on parts of the software that are frequently enhanced. Of particular value was the full separation of modules to minimize inter-module dependency, which allows dedicated teams to work with smaller configuration modules as independent products associated with their own release cycles.

One major advantage of the use of Java as the language for FDP was its superior support for introduction of improved systematic unit-level test policy that reduced effort of testing for anticipate bugs, increased the effectiveness of testing, and decreased the number of defects found by system-level tests by detecting bugs more rapidly at the unit-level by reducing the time to achieve requisite levels of statement and branch coverage. Some software components achieved 70% statement test coverage and 60% software branch test coverage with levels of effort that were significantly lower than before the ADA to Java transformation.

Another activity that has been undertaken since the conversion is the convergence of the European jFDP baseline and the Export jFDP code baselines, the separate versions of jFDP that are used in Europe and at sites throughout the rest of the world. Central components of the European and Export version of jFDP were identified as potentially common because they shared 80% common requirement specifications. Their convergence achieved common interfaces and common low level objects to be defined across both Java baselines. Systematic use of the Java interfaces pattern, specifically the encapsulation of implementations of all types and services exported and imported was achieved for the common modules, and now the two versions of the jFDP systems can link using the converged common modules. Such a high degree of product convergence was not technically possible before the Java transformation enabled much more systematic and more highly automated CSCI-level and unit-level regression testing to be used during final tuning phase and operational switchover.

ACKNOWLEDGEMENTS

The authors would like to acknowledge Roger Knapp, Robert Couch, Francesca Anderson, Greg Tadlock and Randy Doblar of The Software Revolution, Inc for their contributions to the successful execution of the EATMS project.
APPENDIX A
Acronyms & Definitions for Eurocat systems

- **ADS**: Automatic Dependent Surveillance. Surveillance system based on aircraft data downloaded to the ground system. In ADS-B stations are scattered in the controlled ADS-B airspace to populate a network with aircraft positions and reports.

- **BADA**: Base of Aircraft DAAta. Aircraft Performance database allowing modeling of commercial aircraft managed by ATC applications.

- **Dataset**: set of parameters and data files that are produced by an offline application for online usage in Eurocat system. It contains both parameters used to tune the system behavior and operational data that adapt the Eurocat system to a particular airspace.

- **FDP**: Flight Data Processing. Eurocat CSCI that manages all aspects of flight plan data, including Flight data processing, ground to ground messages, environment data etc.

- **JFDP**: Java image of the FDP core function (FPL).

- **FPL system**: core functionality of the FDP CSCI (64% of code and requirements). The FPL is the heart of the future evolutions brought by EATMS projects.

- **FPL record**: record of Flight Plan Data stored or computed by FDP in a disk database.

- **FDS**: Flight Data Server. Module that interfaces FPL process with the rest of Eurocat system via UBSS services.

- **TPM**: Trajectory Prediction Manager. Piece of software that computes the vertical evolution of the aircraft trajectories.

- **UBSS**: Unix Based System Software. Thales Middleware for ATC systems, running on top of unix/linux OS.