SPACE DYNAMICS SOFTWARE ELECTRA

Philippe Pavero (1), Guillaume Rochais (2), Mirentxu Beuvelot (3)

(1) Capgemini, 109 av Eisenhower, BP53655 - 31036 TOULOUSE Cedex, France, Email : philippe.pavero@capgemini.com
(2) Capgemini, 109 av Eisenhower, BP53655 - 31036 TOULOUSE Cedex, France, Email : guillaume.rochais@capgemini.com
(3) Capgemini, 109 av Eisenhower, BP53655 - 31036 TOULOUSE Cedex, France, Email : mirentxu.beuvelot@capgemini.com

ABSTRACT

Since 2007, CNES has been developing a method and a tool –ELECTRA– for evaluating public risk related to space operations. ELECTRA calculates the risk for ground populations due to four types of events: uncontrolled re-entry, failure of propulsion during controlled re-entry, final orbit re-entry or rocket launching. For each considered failure along the nominal trajectory, ELECTRA performs a Monte-Carlo simulation of the possible trajectories induced by all dispersed parameters on the environment, the spacecraft and then the debris resulting from the fragmentation or the explosion after the entry in the atmosphere. Then, ELECTRA computes the risks taking into account the energy of the remaining fragments, the population density and protection afforded by buildings. Originally built in Fortran, the development of a new version of ELECTRA tool in Java is under progress since 2014. This article focuses on the functionalities, the architecture changes and finally the validation of the tool.

1. INTRODUCTION

1.1. Context

As French National Space Agency, CNES has the prerogative for some missions, one of which is to develop efficient state-of-the-art tools for evaluating risks related to space operations. The development of the ELECTRA tool in Fortran, undertaken in 2007, meets the requirement for precise quantification of the risks involved in launching and re-entry of spacecraft. At the beginning, ELECTRA was implemented for internal CNES safety needs, but soon it has been provided to space operators, in the frame of the French Space Operations Act, to assess human risk associated to their operations. Since December 2010, ELECTRA has been deployed and used operationally to monitor the risk associated to each launch from Guyana Space Centre. The tool has also been used to estimate uncontrolled re-entry risk of all CNES LEO missions.

Following the development of CNES new celestial mechanics libraries in Java, the decision to port ELECTRA started with two prototype phases in 2014 and 2015. Their goal was to validate the precision of the results on the simpler uncontrolled reentry mode, and lay out the architecture of the new ELECTRA tool.

1.2. Tool characteristics

The tool currently comprises three products:

- The ELECTRA Fortran, developed in Fortran 95 with GUI in GENESIS (a language developed by CNES for the last 20 years). ELECTRA Fortran takes advantage of the experience accumulated at the CNES in the BIBMS libraries. This version runs on Suse Linux Enterprise Server 10SP4 and RedHat 6.4.
- The ELECTRA Java, developed in Java 1.6 with GUI in GENIUS (a Swing-based framework recently developed by CNES). This version is based on the new PATRIUS [3] library developed by the CNES as a replacement to BIBMS. This version runs at least on Suse Linux Enterprise Server 10SP4, Redhat 6.4 and Windows 7.
- The viewer called ORESTE, based on MapFish which uses a multiple layers technology. This technology simplifies the customization and evolution. Most layers come from the cartographic database Global Insight Plus and also from the population database GPW (Global Population of the World) of the University of Columbia.

2. ELECTRA FUNCTIONALITIES

ELECTRA tool can process risks for four types of contexts: rocket launching, satellite controlled re-entry, satellite uncontrolled re-entry and final orbit re-entry. For the first two cases, ELECTRA takes into account degraded cases due to a premature stop of rocket propulsion or a failure of

satellite deorbitation manoeuvre. ELECTRA computes two complementary estimations of the risk: the probability of causing at least one victim and the expected value of the number of victims.

The risk computation is done by:
- Assessment of fragment impact location and their probabilities of occurrence,
- Consideration of population distribution and habitat protection.

2.1. Uncontrolled re-entry

Assessment of the "impact" risk during Uncontrolled Re-entry is calculated in a different way because the debris fallout zone is usually unknown. The potential fallout zone corresponds to the area of the Earth's surface between latitudes +i and –i (i being the inclination of the spacecraft orbit). To achieve more precise risk, the ELECTRA tool discretizes the [+i, -i] latitude range into N latitude bands correlated with the population grid and taking the following elements into account:
- The population density is variable according to the latitude band,
- The probability of falling in a latitude band depends of the latitude of the band and of the spacecraft orbit inclination.

2.2. Computation of trajectories and impacts for controlled re-entry, launching and final orbit

To obtain the impact coordinates (latitude and longitude) and the energy of impact of each fragment, ELECTRA first calculates the trajectory of the intact vehicle until its fragmentation and then the trajectories of each fragment to the ground. This computation is done for each considered propulsion failure.
Several sources of dispersion and/or poor knowledge of some parameters can influence directly on the impact point of debris:

- The initial position of the falling object (state vector, trajectory),
- The deorbitation maneuvers characteristics (only in controlled re-entry),
- The intrinsic characteristics of the vehicle,
- The altitude of the feared event (rupture or explosion) leading to the spacecraft fragmentation,
- The fragmentation characteristics (including the ejection velocity at the loss of integrity),
- The environment.

Since the effect of these uncertainties on the risk is difficult to predict, the ELECTRA method uses a Monte Carlo process to compute the risk for controlled re-entry and launching phase, taking into account all these dispersion sources.

The forces taken into account in the computation of the trajectory, as well for the spacecraft as for the fragments, are:

- The gravitational attraction of the Earth: with definition of the maximum order and maximum degree of the terms of the spherical harmonic development to be considered.
- The gravitational attraction of the Sun and the Moon
- The atmospheric forces (drag and lift)
- The forces due to solar radiation pressure.

In the case of **controlled re-entry**, the maneuver allows to target an uninhabited re-entry area. But for degraded cases where the thrust is not the expected one, fragments can impact inhabited areas. ELECTRA allows the definition of maneuver failures with their associated probability of occurrence. For each failure case, the intact vehicle trajectory is calculated taking into account the failed maneuver (cf. Fig 1.). The extrapolation of the intact vehicle starts with an initial state vector, to the failure instant and finally until the fragmentation. At this point, each debris is extrapolated to the ground.

Maneuvers are simulated as continuous thrust. ELECTRA allows the definition of the characteristics of the engine (specific impulse and thrust level), the maneuver command type on the satellite (maneuver commanded in time or delta-V) and the maneuver characteristics (date, expected direction). Each of these inputs can be scattered by a Gaussian or uniform law.

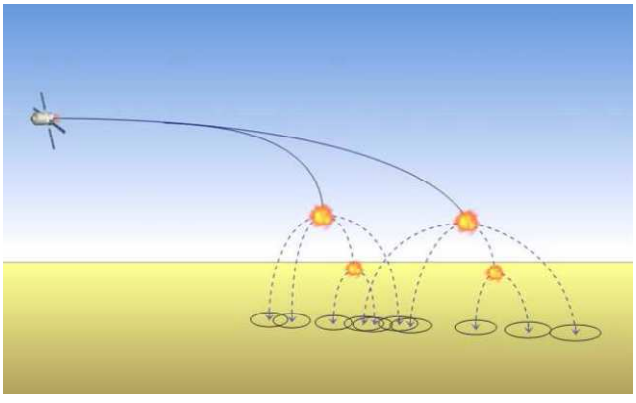

Figure 1 : Impact calculation for Controlled Re-entry

For the risk due to a **rocket launching**, ELECTRA considers failures related to the rocket propulsion premature

stop. The boosted phase trajectory is given in an ephemeris file which contains position and velocity of the launcher sampled at a given frequency. ELECTRA considers a stop of the boost at each point of the ephemeris and calculates the trajectory of the launcher to fragmentation and then, debris trajectories to the ground (cf. Fig 2.). The probability of occurrence of such a failure can evolve during the boosted phase.

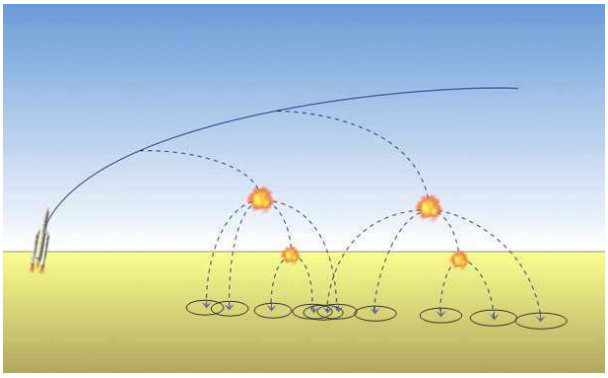

Figure 2 : Impact calculation for Launching

For the risk on **final orbit**, ELECTRA considers an uncontrolled-reentry, but with knowledge on the vehicle position and characteristics, albeit with uncertainties modeled by dispersion on ballistic coefficient. From this knowledge, an algorithm computes equidistant re-entry points at a given altitude, called final orbit, and assigns a probability of occurrence to each of them. ELECTRA then estimates the risk by computing the trajectory from each re-entry point, considering fragmentation and thus the impact from each debris.

Depending on the distance between each re-entry point and the dispersion on the ballistic coefficient, the final orbit can comprise many re-entry points, which leads to significant computation time. In order to drastically decrease the computation time, ELECTRA only computes the full trajectory for one in several re-entry points, called anchor. Each anchor is approximately the middle point of an angular portion of the final orbit, that is to say that the distance between each point in this angular portion and the anchor is smaller than a fixed value. ELECTRA then derives the impacts of every re-entry point in the angular portion from those calculated from the anchor.

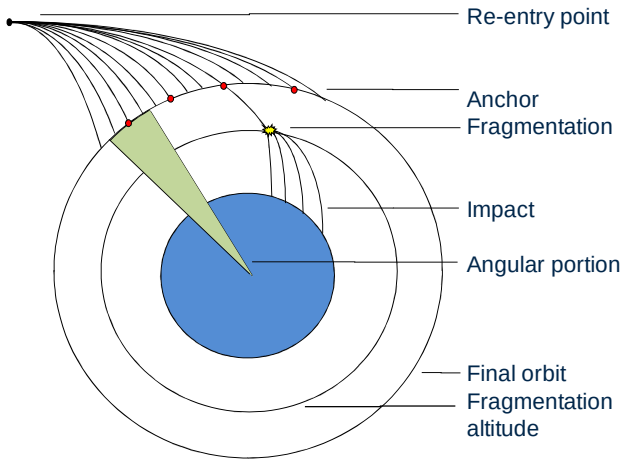

Figure 3 : Impact computation in Final Orbit

2.3. Computation of risk

Once impact points have been computed with a sufficient number of simulations with respect to the level of confidence objective, ELECTRA calculates the probability of incurring at least one victim, and the expected value of the number of victims by taking into account the density and the vulnerability of impacted populations.
ELECTRA uses as input a population file in GPW format. This file describes the number of inhabitants per unit of a global grid of quadrilateral latitude-longitude cells at a chosen resolution (0.25°, 0.5° or 1°) for a given year.
For vulnerability, ELECTRA uses coefficients that represent the level of protection (offered by a building or other means). These coefficients depend on the latitude, the time of day (day /night), the season (summer / winter) and the debris impact energy. Thus, for each computed impact, ELECTRA knows the density and the level of protection of the population as well as the casualty area of the fragment and can therefore calculate the probability and expected number of victims associated with this impact.

2.4. ELECTRA GUI

The graphical interface of ELECTRA allows the user to enter all the parameters needed to compute the risk associated with one of the presented modes. The main behavior of the GUI is defined only once for all modes. Each mode is associated with a Spring XML context, which defines what parts of the GUI should be displayed for this mode. This way, the different situations are both independent, yet all the same.

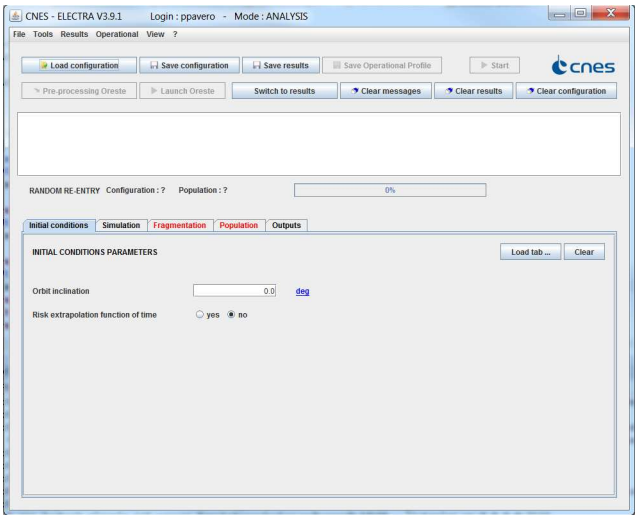

Figure 4 : ELECTRA GUI for uncontrolled re-entry

The GUI also yields several tools, mainly to display or modify input data. For instance, ELECTRA can display a GUI to define fragment files, used to model the lists of debris produced by the fragmentation of the vehicle during re-entry. Most of the parameters of this GUI can be dispersed (surface, mass, ballistic coefficient…).

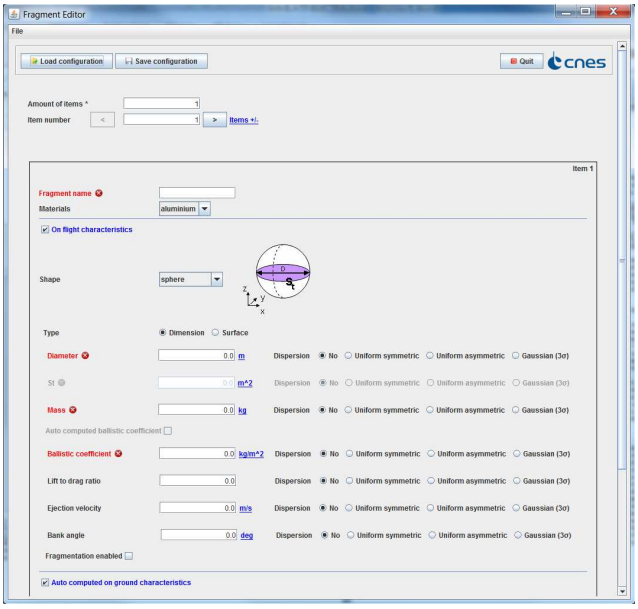

Figure 5 : Fragment editor

3. COMPARISON OF THE FORTRAN/JAVA ARCHITECTURES

The build of a new version of ELECTRA is the occasion to improve the architecture of the software. It is also the occasion to modify some parts that made maintenance difficult. This chapter highlights the key differences between the Fortran and the Java version of the tool.

3.1. On the language

Obviously, the change of programming language leads to fundamental differences between the tools, especially with the target language being Java. The point here is not to make an exhaustive comparison of the two, but to focus on the main observed benefits.

Java accelerates the process of porting ELECTRA to a new platform. The Fortran version only runs on a Suse 10sp4 or Redhat 6.4 platform. With some care given to file handling, the Java version runs naturally on Windows 7, Suse 10sp4 and RedHat, where it is validated, and can most probably be used on many other platforms. Validation is still needed to verify the results, but porting to a new platform no longer requires checking every library needed by the tool or installing appropriate ones when necessary. Moreover, the installation of ELECTRA is handled by an installation wizard. One of its roles is to search the system for all the required libraries and their versions. This will be highly simplified for the Java version since the only requirement will be the 1.6 version of Java.

Another key benefit comes from the development environment. For all development using Java, CNES has set a workshop based on Eclipse, with embedded quality plugins for Checkstyle, PMD, Findbugs. With fixed plugins, CNES can distribute a fixed configured set of quality rules, and ensure the coherence of this quality referential throughout every project.

The last key benefit is about unit testing. ELECTRA makes use of JUnit for unit tests, that were very difficult to develop and use in Fortran. Now, they are directly available in Eclipse. Most importantly, they are run nightly by Hudson on a continuous integration platform. Hudson gives us crucial feedback on integration of the code of ELECTRA, which is developed by a team of five persons. The project also uses Sonar to give metrics, quality rules compliance and code coverage, which leads to a much better quality of the code.

3.2. Simplifications

The first significant simplification comes from the change of CNES library suite. The Fortran version used a panel of Fortran libraries from the BIBMS suite for different applications. These tools had to be installed and configured separately, which defined a complex architecture. Since 2009, in the Sirius project, CNES started the development of a new Java suite of libraries and tools to replace BIBMS. ELECTRA is based on one of them, Patrius, which combined with Java functionalities replaces every BIBMS tool. Here is a quick overview of the correspondence of the Fortran tools with their equivalent in Java (SD=Space Dynamics):

MSLIB90	Basic SD components	Patrius
MECASPA	High level SD components	
PSIMU	Earth or Mars orbit extrapolation	
COMPAS	SD models data	
MAGE	Warning and error handling	Java

Similarly, the new GENIUS library, developed by CNES to implement scientific GUI, replaces both GENESIS and MADONA (file format). Contrary to the GENESIS code files that mixes Fortran to custom code GENIUS is based on Swing and thus, can be directly written in Java, which makes it much easier to integrate in ELECTRA.

The second significant simplification derives from the use of Maven. The different computation modes (random re-entry, launching…) are all clearly separated in dedicated Maven projects. This simplifies integration as working on code specific to one mode doesn't impact any other. Shared functionalities are placed in a "commons" Maven project. This architecture enables a clear chain of dependencies.
The independence of the modes makes it possible to install them separately.
The Fortran version contained complicated compilation, generation and installation scripts. In the Java version, everything is handled by Maven's Assembly goal, defined for each project. This goal also manages the packaging of every dependency. The result is an easy-to-install zip file that contains everything needed to run the software.

3.3. Specific improvements

ELECTRA uses many dispersed variables. As these variables apply to very diverse models, such as fragment size or atmospheric density, they are naturally scattered in the code. In Fortran, the values were drawn at the usage of the variable value, resulting in a code that was sometimes hard to follow. Moreover, these values can be written in an output file for investigation, and their dissemination makes it difficult to retrieve the information. To prevent this, the Java version centralizes the drawing of the variables. In effect, the variable are still attributed to their respective model, but they are all registered to a single service at creation. This service therefore has access to every variable, and is in charge of launches the drawings. It can also access to all drawn values, and is able to write the dispersion file more easily.

It is possible to execute the computations on parallel processes in ELECTRA. In Fortran, this was done through Open-MPI, which had two issues : it is sometimes hard to investigate problems with it, and it implies another dependency for the software. Another issues stem from the software code itself, which duplicates most of the code to provide either a sequential or parallel computation. These issues made the software difficult and time-consuming to maintain. In Java, the parallelization is addressed differently. This version uses the native Java Executor framework. A CompletionService is running, and waits for jobs (typically the computation of one failure or re-entry point) to be submitted. As soon as the result is available, it forwards it to an outputs handler. This handler also manages error cases. The management of the pool of thread is done by native Java objects. To make the code easier to maintain, there is no difference between sequential and parallel modes : every portion of parallelizable code (one for each mode) is written as a Callable object, which is a more flexible version of a Thread.

4. VALIDATION OF ELECTRA JAVA

The reference in validating the Java version of ELECTRA is given by the results of the Fortran version. Although this principle is quite simple, it can rapidly become complicated to apply, due to three main issues:

- The difficulty to extract results of specific functions from the Fortran version,
- The unavailability of models,
- The changes in the dispersion of variables.

The following paragraphs will focus on each case and explain how they were solved through examples.

4.1. Validating specific methods

In the Java version, the trend is to validate each computation method with unit tests. Thanks to object oriented design, it is possible to isolate these methods rather easily, even if some mocks are required. In the Fortran version though, this is usually not possible. In this case, the validation is done from scratch, and several means are used:

- Production of a reference with a script,

- Result analysis with Excel, by copying the algorithm,
- Special analysis, with a specific method.

The third case was applied on ELECTRA for the validation of the uniform distributions. The value drawing is done by Apache commons-math (which is a dependence of Patrius), but since the subject lies at the very heart of ELECTRA, the variable drawing had to be validated. In this case, the method was as follows, for a variable drawn uniformly in a given interval:

- Draw the variable a large number of times,
- Divide the interval in N boxes,
- For each box, count how many times the value is drawn in its interval,
- Check that each box contains the same number of drawings.

This is visually summed-up in the following graph that shows the deviation in percentage on the number of drawings per box, for 10'000'000 drawings and 50 boxes. This shows that the mean deviation is of 10-3 order, which is correct for ELECTRA.

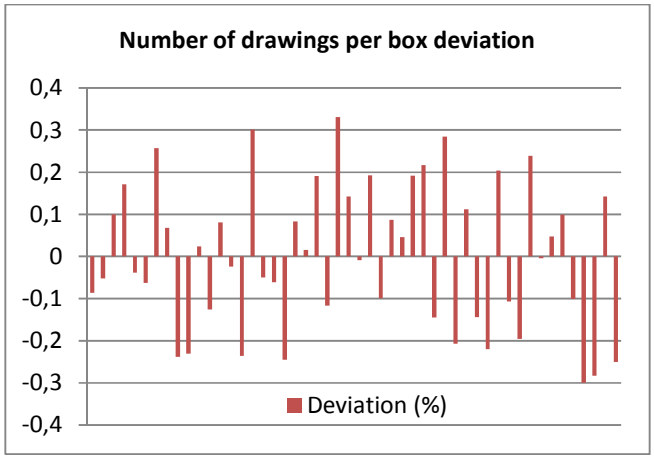

Figure 6 : Uniform distribution deviations

4.2. Unavailability of models

The Fortran version of ELECTRA is based on the BIBMS set of tools, whether the Java version uses the much more recent Patrius. Therefore, most of the models, such as atmosphere models, are only available in one version of ELECTRA. In order to validate all the possibilities of the newer version, one must begin with validating one case with the common models. In effect, considering the atmosphere models, only US76 is available in both BIBMS and Patrius, and is consequently chosen for the validation case. Moreover, what happens to the atmosphere models also occurs for the Earth potential models, the solar system ephemeris, or even the type of integrator.

Fortunately, the use of an object-oriented language, and a carefully designed library, meets this problem perfectly. The solution lies in the usage of interfaces. For instance, all the atmosphere models in Patrius implement the interface Atmosphere, which defines what each model can compute, for instance the atmospheric density at a given point and date. In ELECTRA, the computations only work with this interface, with absolutely no knowledge on the explicit type of model. As a consequence, each model can be substituted with another, and this has no impact on the ELECTRA algorithm. With this crucial point in mind, we can conclude that if ELECTRA is validated with one model, then the main algorithm is valid for any other model.

4.3. Changes in the dispersion of variables

The dispersion of the variables is done by using the Mersenne-Twister algorithm. This method generates random series of numbers, initialized with a seed. With this seed, it is possible to reproduce random drawings, and thus validate specific results, i.e. for each Monte-Carlo simulation, based on dispersed values. The issue here is that the two versions of ELECTRA use the random series differently.

In Fortran, one series was used for each failure or re-entry point, and the variables were drawn by using successive values of this series. In Java though, this was changed, to use one random series per variable. So, for the same configuration, the drawn values are different, and it is no longer possible to validate specific results based on dispersed values.

In order to bypass this issue, one has to consider large amount of drawings, and analyze the result from a statistical point of view. This principle was applied for the random re-entry mode, with a configuration using a dispersed fragment. The dispersions affect the resulting risk, which means that the risk computed on both versions of ELECTRA should converge to each other for a large number of simulations. The result of such an analysis is shown Figure 7.

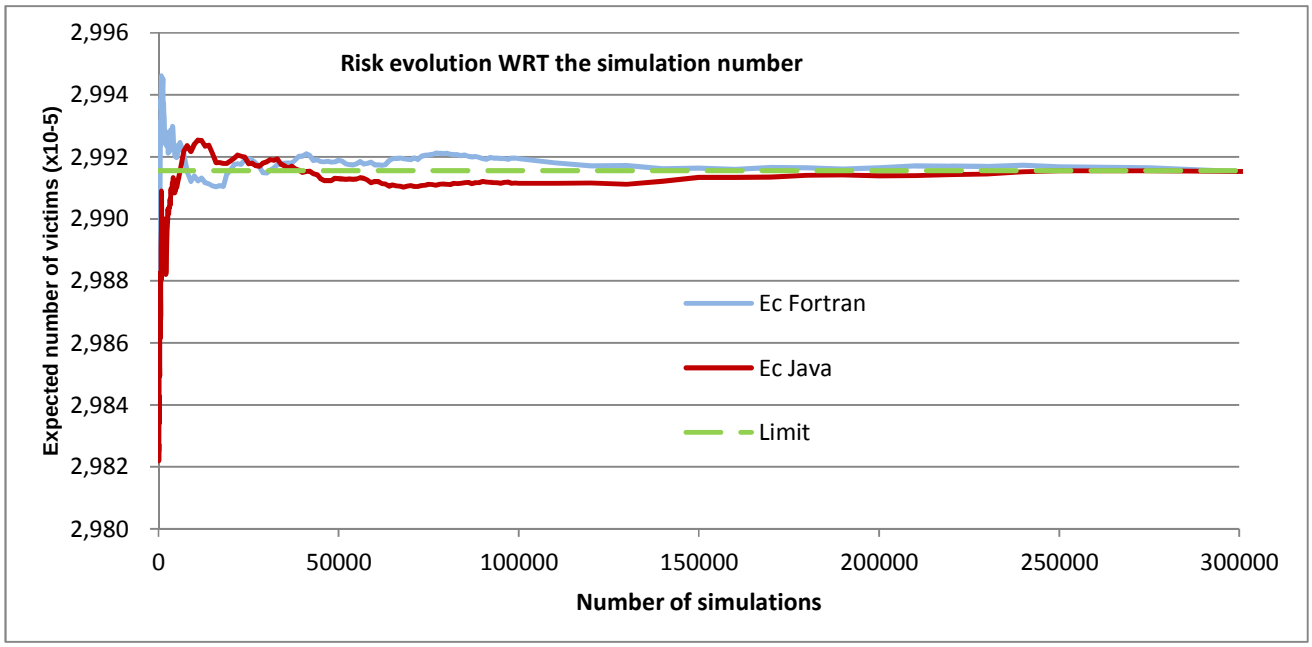

Figure 7 : Risk evolution for one to 300'000 simulations

With even more simulations, one curious behavior can be observed. One would expect the risk to converge with a very high number of simulations, but due to the order of magnitude of the risk (generally about 10^{-6} to 10^{-4}), the machine precision is reached and small variations of the risk are still observed. Figure 8 shows what happens for more than 300'000 simulations, on the same configuration as above.

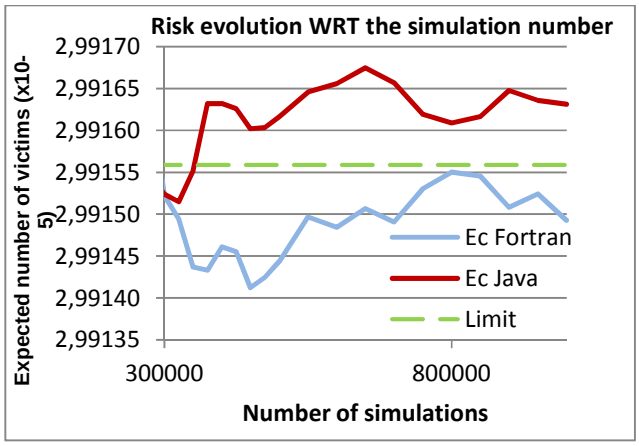

Figure 8 : Risk evolution for up to 1 million simulations

This behavior has to be taken into account for future validations in the form of a threshold on the results. The observation shows that the relative deviation between Fortran and Java on this case is of the order of 10^{-5}, hence future validations on the risk values will use a 10^{-4} relative threshold.

5. CONCLUSION

The Java version of ELECTRA is one of the first software to use the new CNES set of Java tools, with Patrius and Genius. As such, it is in a way a test platform of this new set of tools.

The use of Java, Eclipse and Maven lead to software that is easier to develop, maintain, generate and install, especially for portability. Indeed, at the cost of an overhead in the conception phase, the object oriented language ensures that the development models closely the problem to solve with objects representing real problem elements. This also allows the definition of many extension points as long as interfaces and inheritance schemes are chosen carefully.

These extension points also facilitate the validation, because if every implementations of an interface are individually validated, it is possible to validate the higher level algorithm with only one of these implementations.

All these changes, though, make validation a very delicate business, and one has to conduct detailed validation in some specific cases to ensure that the results are correct. Another point that has to be closely monitored, but was not examined in the development of the Java version of Electra yet, is the performance of the computation.

In the coming months, the development of ELECTRA will have to tackle these issues on the more difficult cases of the launching failure and controlled re-entry. The tool will then be put into service in the frame of the French Space Operations Act.
The experience of the development of ELECTRA Java, could be put in use to avoid pitfalls for the development of other tools, allowing to concentrate on creating more value and consolidate their functionalities.

10. REFERENCES

List and number all bibliographical references at the end of the paper. The references can be numbered in alphabetic order or in order of appearance in the document. When referring to them in the text, type the corresponding reference number in square brackets as shown at the end of this sentence [1].

[1] C. Hourtolle, A. Gaudel-Vacaresse, and A. Blazquez, "ELECTRA : *launch and re-entry analysi tool*"

[2] F. Chemama, B. Lazare, and C. Aussilhou, *"ELECTRA tool launch and re-entry safety analysis"*, ICATT2010.

[3] Patrius, CNES astrodynamics commons library, developed in Java.