Capitalising Astrodynamics Tools over decades

Vincent Martinot (TAS)

AstroNet-II Final Conference
Tossa de Mar (Spain) from 15/VI/2015 to 19/VI/2015
• Introduction

• Evolutions in
  • hardware/operating systems,
  • programming languages,
  • development environment,
  • Computer language background of fresh engineers

• Recommendations for the
  • Development of your tools
  • Maintenance of your tools

• Example of implementation

• Synthesis
• Thales Alenia Space France (Directorate of Observation & Science) has internally developed over the years a series of Astrodynamics Tools
  • They need to be maintained and enhanced over the years to ensure a sustainable capacity in mission analysis

• This represents a real challenge to do so over decades with regards to evolutions in hardware/operating systems, programming languages, development environment, educational background of fresh engineers,…

• There are however simple recommendations for the development and maintenance of your tools that can help you to reduce the effort (and the cost) to try to achieve this goal.

• The elements presented here shall be seen as return on experience (RETEX) over the last 20 years in an industrial context
  • Different environmental constraints may lead to different solutions
- **Hardware**: From central computers to laptops

  - 1990’s: Workstation, Central computer
  - 2000’s: Desktop PC
  - 2010’s: Laptop PC

- **Operating systems**: From VMS to Windows

  - 1990’s: Unix, VMS
  - 2000’s: Windows
  - 2010’s: Windows
Constraint on the work process

- **Central Computer / VMS:**
  - Time sharing of the computer resources: need to estimate the CPU of your run before you submit your job to the central computer

- **Workstation**
  - Not equipped with word processing: need to swap between computers when writing technical notes

- **Desktop PC**
  - Not possible to perform calculations during off-site meetings: need to take actions back to the office even for simple calculations

- **Laptop PC**
  - Constraints not identified so far on the work process (CPU Ok / Storage OK / possible to work when in mission)
• Fortran, an imperative language, was available to develop programs and libraries in Astrodynamics when using a central computer.
• When moving to workstation, both imperative (Fortran, C) and object-oriented languages (C++) were available.
• Matlab started being used for prototyping from early 2000’s (combining calculations and visualisation).

<table>
<thead>
<tr>
<th>Programming languages</th>
<th>From functions to objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990’s</td>
<td>2000’s</td>
</tr>
<tr>
<td>Workstation</td>
<td>Desktop PC</td>
</tr>
<tr>
<td>Central computer</td>
<td>Laptop PC</td>
</tr>
</tbody>
</table>

Imperative

- Fortran
- C

Object oriented

- C++
- Matlab
- Java
• Revision control was basically “implemented” in central computer via a native VMS mechanism: the addition of a numbering in the filename.

• Workstations offered a real revision control system using SCCS and later RCS (with tags of the version number embedded in the object/exe file)

• When working on desktop PC, the revision control system was moved to Clearcase with an access to a repository on a central server (previous revision history not kept because extensive tests between the Unix and PC versions of Astrodynamics programs)

• Clearcase was not compatible with the move to laptops and the need to develop even when not on-site
  • Deployment of Git with a local repository to manage evolution when off-site and a resynchronization process (on user initiative) with a central repository to share the evolutions with the other developers
Computer language background of fresh engineers

• In the engineering universities and high school in France
  • Up to the end of the 90’s, imperative languages only were taught (except in computer science sections)
  • Object-oriented was introduced in the 2000’s only on a conceptual level (development of very simple programs to illustrate the object-oriented programming)
  • Matlab started being taught at about the same time

• Nowadays
  • Java is the only object-oriented language that is taught
  • Matlab is usually the only imperative language which is mastered by the students
  • Fortran is an optional module in the engineering universities

• Java & Matlab are proposed to more experienced engineers in the industry to reduce the gaps of computer languages w.r.t. fresh engineers
Recommendation for the development of your tool (1/2)

• **Which language to choose to develop new tools?**
  • It is difficult/impossible to guess what will be the language that will be the reference for Astrodynamics for the next 20 years…
  • It really depends on the basic astrodynamics libraries available to you

• **If you have a Fortran heritage in Astrodynamics libraries, no need to switch to Java to follow the trend…**
  • models (e.g. atmospheric) are still made available in Fortran
  • Fresh engineers can more easily learn Fortran than experienced engineers Java…

• **If you do not have, select Orekit (Java) as the building block on which to build your new developments**
  • You are sure to have consolidated methods for frame transformation, force estimation, propagation, parameters conversion
  • You will benefit from the further development of the Orekit community
Recommendation for the development of your tool (2/2)

- **Maximize your chance of being compatible with the forthcoming operating systems**
  - Separate the interface (more sensitive to the evolution of the graphics primitives) from the calculations (to be submitted directly to the processor)
    - It simplifies the validation process
    - If your interface is not running anymore, you still have a way to run your calculations!
  - Exchange the information between both via a TEXT file (input for the calculations)
  - Output the results in a TEXT file (structured if possible)
    - A structured text file allows you to develop API and scripting

- **Do not rely on operating system commands**
  - E.g. embed a sort into your program if you need to sort your output (do not call the one that may be available at operating system level)
- **Avoid using a too smart algorithm & code simply**
  - What you found smart to do once may require hours years later to be understood (even by you 😊).
  - This maximizes the chance that you are not the only one to make it evolve
  - It may not be the most efficient in terms of computing time but computer calculation power goes on improving anyway

- **Put information directly in the code**
  - Use this information e.g. in the header of the code to generate a developer manual (for a program) or a synthesis of the calling sequences for a library

- **Develop an interface**
  - Reduces the need for a user manual (less documents to maintain)

- **Split complex calculations into smaller programs**
  - easier to maintain individually but more to maintain…
Recommendation for the maintenance of your tool (2/2)

- Develop management tool to compile your programs (with GUI 😊)

- Advantages
  - Gives you an overview of the programs you have to maintain
  - Allows an easier deployment of your programs (the end user compiles on its computer)

- Implemented by
  - Using makefile (and not project) to compile
  - Retrieving dynamically the list of available programs
Example of implementation

Interplanetary missions

- **INT (Interplanetary Navigation Tool)**
  - In-house tool gathering TAS heritage on interplanetary Mission Analysis

- **Main features**
  - INT is organized in modules performing a dedicated calculation
  - Interplanetary Trajectory design modules are gathered in a single theme (Cruise)
  - A main window allows to select the module you want to run
  - Ancillary calculations and visualization tools are available to help the design of interplanetary trajectories
  - Implemented in:
    - Fortran for the code
    - Tcl/Tk/Tix for the GUI
Example of implementation
Scripting capability (1/2)

- API in Matlab and perl are available and give a scripting capability

- Modules can therefore be linked to:
  - study the sensitivity of the output data generated by one module to a set of its input parameters
  - allow a step-by-step trajectory design (typically the refinement of the results given by a first guess using various schemes e.g. DSM on 1st leg or on 2nd leg)
  - be able to incorporate system constraints in the design (e.g. minimum distance to the Sun, Sun aspect angle at impact, swing-by minimum altitude, arrival velocity,...)
#!/usr/bin/perl

use strict;
use INT_Connect;

my $module='lamsol';
&INT_Connect::NamelistCopyTemplate($module);
&INT_Connect::NamelistSubstitute($module,'EPOCH','IDATE','3');
&INT_Connect::NamelistSubstitute($module,'EPOCH2','IDATE2','3');
&INT_Connect::NamelistSubstitute($module,'RUN','CDNP','"3\"');
&INT_Connect::NamelistSubstitute($module,'RUN','GARR','" & -1310\"');
&INT_Connect::NamelistSubstitute($module,'WIN','IWIN','1');
&INT_Connect::NamelistSubstitute($module,'WIN','WINDEP','60');
&INT_Connect::NamelistSubstitute($module,'WIN','WINARR','60');
&INT_Connect::NamelistSubstitute($module,'RUN','NCO','-1');
&INT_Connect::NamelistSubstitute($module,'FINCOND','MAS_ME','5.75e10');
&INT_Connect::NamelistSubstitute($module,'INICOND','ILNMODEL','20');

my $fichier='chemint_plot2D.dat.3years';
open (INPUT, "chemint_plot2D.dat.3years") || die "Problème pour ouvrir $fichier: $!";

my $departure_date; my $arrival_date;
my $res;
Example of implementation Around the Earth

- Similar principles are used to develop and maintain a set of programs for the design of mission around the Earth (Space Mechanics Tool = SMT)

- A single hosting structure for the different modules
  - Code based on a common library in Fortran
  - GUI using the same procedure in Tcl/Tk/Tix

- Graphical results are generated using Matlab (via an utility in runtime to avoid the need for licence)
• Capitalization of the tools developed along the years needs to be considered in the general context of sustainability of engineering capacities

• A few simple rules can help the developed code to be run albeit the change of hardware/software/operating systems

• Choice of the programming language for your code depends on the libraries (completeness, regular updates, available documentation, etc) which you can build upon

• The change will be costly therefore think twice before making a change

• Other returns of experience on this subject are welcome!