OREKIT: AN OPEN SOURCE LIBRARY FOR OPERATIONAL FLIGHT DYNAMICS APPLICATIONS

Conference Paper · April 2010

CITATIONS
77

READS
4,891

3 authors, including:

Luc Maisonobe
Thales Group
19 PUBLICATIONS **131** CITATIONS
SEE PROFILE

Pascal Parraud
CS GROUP, Toulouse France
9 PUBLICATIONS **98** CITATIONS
SEE PROFILE

Orekit : an Open-source Library for Operational Flight Dynamics Applications

Luc Maisonobe(1) (Luc.Maisonobe@c-s.fr)

Véronique Pommier-Maurussane(2) (Veronique.Pommier@c-s.fr)

(1), (2)

CS Communication & Systèmes

Parc de la Plaine - 5 Rue Brindejonc des Moulinais - BP 15872

31506 Toulouse Cedex 5

France

1 OPEN-SOURCE FOR SPACE FLIGHT DYNAMICS LIBRARY

1.1 Historical context

In 2002, the following observation was made by CS: several request for proposal (RFP) had come out that requested complete flight dynamics systems, either for low earth orbit or for geostationary orbit. Among the whole lot of products available on space flight dynamics market, several could pretend to fit, but they were either fulfilling only part of the requirements, or they were monolithic suites that offered much more possibilities than necessary. Moreover, proven systems were old.
On CS side, lots of experience had been acquired along the years and the many fulfillments performed, but the company did not own the corresponding intellectual property. All the work done was on customers request and IP was sold along with the product. The licenses costs proved to be far too expensive and put customers off.
From that reflexion came out the idea to develop a CS-owned space-flight dynamics library, with CS internal funds, that could be used as a building block for flight dynamics systems and would allow CS to answer AUTONOMOUSLY to future RFP, thanks to this new asset.
At the same time, CS took advantage of the opportunity to use state of the art technology, concepts and physical models for that project.

All that work took place between 2002 and 2008 with an ups and downs progression as CS is a commercial company, so it was not always easy to find financial and human resources. Things sped up in 2006 with a long internship. The resulting product was a low level library which services ranged from fast development enabling up to data physical models and highly elaborated algorithms. V1.5 of the product was used for ATV/ISS rendez-vous monitoring.

In the end, Orekit's assets allowed to propose it as a standalone product.

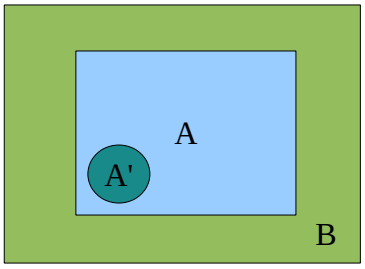
1.2 Strategic choices and business model

In 2007 CS had good feedbacks about Orekit: users were enthusiastic about it, suitability to technical needs was good, it had been embedded in a complete system and flight-proven operationally on ATV monitoring. But it was still a marketing failure as proprietary model seemed to be destructive, putting some clients off and slowing down its development.
For that reason, CS decided in 2008 to change commercial strategy and distribute Orekit as an open-source product.
That decision was supposed to keep the product alive, think it in a larger “ecosystem” and allowed CS to propose an innovative alternative to its competitors proprietary products.

1.2.1 License choice

Publishing an open-source library for space flight dynamics implies some specific choices to make concerning license terms.

Several different open-source license types coexist, which have different consequences when a component is reused.
Consider an original open-source product A on which a developer both adds modifications A' and external code B to deliver a complete application:



- If A is published under the term of a « Strong copyleft » license: A' AND B must be distributed under the same license. It is an diffusive license for open-source field.
- If A is published under the term of a « Weak copyleft » license: only A' must be distributed under the same license as A, B may be distributed under different license terms.
- If A is published under the term of a « Business friendly » license: neither A' nor B have to be distributed under the original license terms of A.

Examples of strong copyleft licenses are GPL and CeCILL, examples of weak copyleft licenses are LGPL, EPL and CeCILL-C, examples of business-friendly licenses are BSD, MIT and Apache.

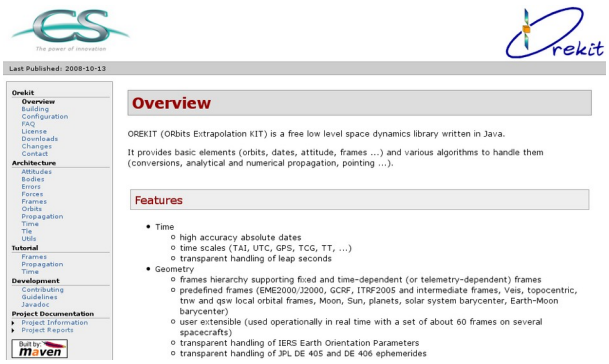
License choice was made after the following analysis:

- **Bad example of the JAT** (Java Astrodynamics Toolkit) which is a free project started in 2002 by students and was developed in a technical context similar to Orekit's. The license chosen was a **strong copyleft license**, which has limited their audience to academics and individuals, and lead them to change the license terms in 2008 to overcome this drawback.
- **Orekit context:**
  - it addresses a niche market with agencies and industrial customers only, both having strong requirements about intellectual property preservation;
  - it introduces a technical breakthrough in a critical field so users should not be rushed;
  - business models chosen is based on services surrounding the product.

All of this lead CS to choose the Apache V2.0 « business friendly » license. As seen before, this is a permissive license, widely known, that basically states that anybody can take Orekit and do what he wants with it, except saying he has written everything or removing the copyrights in the code. An end-user application that uses Orekit can be published using any license, free or not. That license allows unlimited commercial use:

- No propagation of license terms to the user code
- No constraints
- Fulfills the free software 4 freedoms :
  - Freedom to run the program for,
  - Freedom to study how the program works and adapt it,
  - Freedom to redistribute copies,
  - Freedom to improve the program and release the improvements.

Orekit is then free in both meanings of the word: « free as in free beer, and free as in free speech ».
First free version of Orekit V3.1 was released on 17th of July 2008.



1.2.2 Business model

Open-source is an approach that proved efficient in mainstream software industry. It does not always need a very large community as was once thought, but still needs some involvement. The return on investment increases for all contributors as the project expands and the risks decrease at the same time as more and more people use it. The model is attractive for both public entities, academics, industry and SMEs, bringing something to each one of them. It also increases the yield of public funding.

As said before, the open-source business models are simply based on service and support to users, not on license fees. The models include:

- training, on-call support, guarantee
- consultancy (on OREKIT or on space flight dynamics)
- integration (possibly with adaptations)
- specific developments
  - new functions or models
  - custom changes

2 FROM BASIC TO EXPERT USE OF OREKIT

We'll now see how it is possible to build flight dynamics applications from Orekit.

First it's important to point out that Orekit library offers different levels of use. It is to a large extent adjustable to user needs.
It is designed both to be used for quick development for

simple use case, and to be tuned to specific needs for expert use.

2.1 High level features for easy, fast development

Orekit provides a consistent set of low level objects and algorithms. It features all classical notions needed for space flight dynamics like time, frames, orbital parameters, orbit propagation, attitude, celestial bodies, force models, JPL ephemerids ... For each notion, an extensive support is provided.

We'll stress on the following high level features provided to perform the first goal: being easy to use:

- attitude modes
- frames handling
- propagation & associated outfitter
- time scale
- Earth orientation parameters

Attitude modes

Several predefined attitude modes, are available, the most current ones. The implementation is hierarchical : there are « elementary » laws like:

- Body center / celestial body pointing,
- Target / nadir / LOF-offset pointing,
- LOF-offset, fixed rate, spin-stabilized,

then « complex » laws which are specializations of elementary laws: yaw steering / yaw compensation from ground pointing).

All of these modes are completely interchangeable due to the fact that they implement the same interface, so it is easy to switch from one law to another. Novice users or preliminary studies may select a basic law first and later use a complex one with only minor changes at the law initialization and absolutely no changes in the core processing code.

Frames

Frames are organized as a tree structure, the relationship between elements being the transform that links one to another. A new frame can be built by defining the transform that links it to a parent frame; the combination of the transforms is then automatic and allows to go easily from a frame to any other.

The definition transforms can be either fixed or variable transforms. Variable transform can either be a time dependent law so the transform and frame are automatically updated as function of time, or a telemetry dependent law so they can be updated as function of the received telemetry in real-time (this was used for the ATV monitoring, with a set of about 60 different frames).

A lot of predefined frames are also provided, in particular all of the main Earth model frames.

Orbit Propagation

Orekit offers different kind of predefined orbit propagators, from the most basic like keplerian propagator, to the most accurate numerical propagator.
One important feature supported by all propagators is the ability to handle discrete events detections on the fly during the course of the propagation. The propagator can be set to two different modes : « slave » mode or « master » mode.
In the first case, « slave » mode, the handling of the loop step-by-step is under responsibility of the user. It's what was traditionally done in legacy software: a loop over time with computations performed on line at each step.
In « master » mode, the handling of the time loop is delegated to the propagator itself, and only the processing code at the core of the loop has to be written by user, through what is called a « step handler ». The user only has to propagate all at once without handling himself a loop, the « handle step » method of his step handler will be called automatically by the propagator at each step. This mode simplifies user code and increases the decoupling between the boilerplate time handling code and the complex computations user wishes to be performed at each step, outside his main program. This simplification is a great help when discrete events are added inside the regular time loop, as explained later.
The propagator can handle either fixed step or adapted step. Normalized step handler allows to get intermediate states at fixed step even if a variable step integrator is chosen.

Moreover, event detection can be automatically performed by the propagator, by adding what is called « event detectors ». Once again, a large panel of predefined event detectors is available (orbit events, geometrical events, fov events) and users can define their own events. Once an event detector is added, the propagator will check at each step the occurrence or not of required event(s) and trigger it at the appropriate time. Event occurrence time is not predefined, it is discovered on the fly during the propagation, when some user-defined conditions are met.
Event detection **works in both slave and master mode**, but in slave mode the user does not get the intermediate state when the event is triggered, he only has the final state he asked beforehand.**In master mode intermediary state corresponding to event time is computed at the exact time event happens, no matter step size**.

When an event is triggered, it can be used for controlling the propagation, like for example changing

the force models when a maneuver start or stop event occurs, or even completely stopping the propagation when an altitude threshold is encountered during a reentry simulation for example. This can also be used for simple operational events logging (ground station visibility, eclipses ...).

Earth orientation parameters
All Earth orientation conventions are supported for precession and nutation models. IERS 2003 new conventions and IERS 1996 former conventions are available in the existing Earth frame tree, with or without corrections, to be chosen when getting frame.
One frame from one convention can coexist with another frame from another convention in the same application, as the corresponding data are automatically loaded when constructing the frames, relieving the user from the responsibility to handle this himself, and avoiding confusion risk.
The handling of the huge parameters files and different conventions can be ignored by users as it is done automatically. Of course, expert users can still manage the tricky details, but most users won't. This goes to the point that users who are not flight dynamics experts and do not know what precession, nutation or polhody are can use Orekit's frame and take these phenomenons into account without noticing it: they just pick up an Earth frame and use it.

Time scale
Several time scales are supported (UTC, TAI, GPS, TT, ...) and all defined relatively to TAI scale. A date can be defined in any of the time scales available, then used in any other time scale. The conversion is handled automatically, like for frames relations.
TUC leap seconds are also handled transparently; users who do not know about them don't need to bother about them.

2.2 Fine tuning for expert use

Beyond all these foreground assets that enable easy early stages, Orekit offers great freedom for users to design custom-built applications with expert tuning in many respects.

Force models
Physic models are a crucial point for space flight dynamics field. A large panel of the most currently used models are provided in Orekit, but they can still be modified to suit users' need perfectly, or be extended if some very specific models are required. Some of them can also be customized from simple to complex cases. As an example, drag and radiation pressure can use either spherical shape or central box defined by several flat panels and solar array with automatic alignment, or even a custom shape model.

…….

Events
Event detection is also an important customizable point. Users might often need to detect events specific to their application, so new event handlers can be created according to event detector interface.
Moreover, that functionality is convenient and flexible enough to offer the interesting perspective to be extensible to some “exotic” needs. Beyond existing “classical” events, users could imagine to benefit from Orekit's events handling mechanism to detect any concrete or abstract discrete point during the propagation, that could be plot simply by defining the corresponding g-stop function.

Frames
Users can create new frames adapted to their need by defining the transformation that links it to any “parent” frame in the existing tree. This transformation can be time or telemetry-dependent. This is exactly what was done for ATV monitoring, with more than 60 frames involved and the telemetry of both vehicles used in real-time.
One interesting feature of frames update is it can be done implicitly on one frame to satisfy constraints on other frames. This allows for example tuning precisely the position of the center of gravity of a vehicle taking into account the parallax of the localisation antenna which is defined by its own slightly offset frame.

…….

Time scale
Even if it might not be often needed as Orekit already provide wide time scales panel, new time scales can still be created to comply with specific needs. In that case, user simply has to define between his time and TAI.

Data
In order to simplify integration of Orekit-based application in legacy systems, data loading can be customized with a plug-in mechanism that can connect to existing libraries or databases.

3 ACKNOWLEDGEMENTS

We thank Christine Fernandez-Martin for her constant support to the project and for allowing Orekit to hatch out in public field.
We thank all the contributors to Orekit development since 2006, who showed their interest and involvement in that project. We also thank all the users who help diffusing and improving the product.

4 REFERENCES

1. Time Code Formats, Consultative Committee for Space Data Systems, blue book 3 rd edition 2002 .

2. Sylvie Chauvin, *Stratégies et potentiels de l'open source dans les entreprises et les administrations françaises*, 2009-2011, OpenCIO summit, Open World Forum 2009 .

3. Dennis D. Mc Carthy and Gérard Petit, *IERS Conventions (2003)*, International Earth Rotation and Reference Systems Service .

4. *Free Software Foundation, GNU General Public License*, http://www.gnu.org/licenses/gpl-3.0-standalone.html, June 2007

5. CEA, CNRS, INRIA, CeCILL *Free Software License Agreement*, September 2006, http://www.cecill.info/licences/Licence_CeCILL_V2-en.html .

6. *The BSD License*, http://www.opensource.org/licenses/bsd-license.php

7. Apache License, Version 2.0, January 2004, http://www.apache.org/licenses/LICENSE-2.0 .

8. David A. Vallado, Paul Crawford, Richard Hujsak, and T.S. Kelso, August 2006, *Revisiting Spacetrack Report #3, 2006 AIAA/AAS Astrodynamics Specialist Conference* .

9. Ernst Hairer , Syvert Paul Nørsett , Gerhard Wanner , *Solving Ordinary Differential Equations I. Nonstiff Problems*, Series in Computational Mathematics, Vol. 8, Springer-Verlag 1987, Second revised edition 1993.

10. L.Maisonobe, C. Fernandez-Martin, V. Pommier-Maurussane, *Building a community, an approach for Space Flight Dynamics*, September 2009