

JScience, a general scientific API in Java

Silvère Martin-Michellot

BA, BS, MA, MS

Senior computer programmer

silvere@digitalbiosphere.org

www.jscience.org

12 June 2008

Keywords

Numerical computing, scientific computing, Java, science library, framework.

Abstract

This paper describes JScience, a general science API (Application Programmer Interface) written in Java. This API comes in two flavors: one main distribution and another experimental one. Our aim was to cover all knowledge (but database content) up to bachelor graduation level or higher and distribute it for free under GPL license. JScience is the first API ever to do so.

In this paper, I argue that Java offers a different environment from previous languages (C, C++, Fortran) which allows to write a better science API (speed, portability, code quality, features, reusability, extensibility). We hope that this API will define a model for future scientific computing software (OS level service, object oriented, open source, wide spectrum).

Introduction

There is a lot of scientific software available for free on the Internet. Yet, actually, there is two markets that have been poorly addressed by software developers. First, science packages cover almost exclusively mathematics. Some of them cover also some hard sciences (physics, chemistry and biology) but there is very few code in soft sciences. Then, most science packages focus numerical computation, not science software development. What we have is commercial software that helps solve numerical problems but is of no use if we want to make standalone applications.

JScience effort has been made into existence as two separate API. The official distribution, which is small, written in Java 1.5 (generics, boxing...) is targeted towards core mathematics and unit systems makes a nice introduction to the scientific computing world. Its description is already covered in a separate communication. The other distribution is experimental, it has a much wider scope and uses the unit system from the main distribution. This is the distribution I discuss in this paper though much of what is said could apply to the official distribution.

Numerical computation survey

We know for long that Java is a good candidate for scientific computation. It is fast, almost as much as C++ in general and about half the speed of Fortran. It is also object oriented (which is not the case of Fortran) and documented (which is not the case of C++).

This makes Java an ideal choice because it is much less expensive to leverage scientific computing solutions than others competing languages. Yes, your code may execute a bit slower but it will cost you much less to develop and maintain. So, you can actually use the same money to buy faster machines. Moreover, because accuracy is a critical component in scientific computing you are taking much less risks using Java that stack overflow or other kind of hard to trace errors ever appear.

History of JScience

JScience started from a couple of unrelated projects. Mark Hale had developed Jsci from 1999 and ongoing. His focus was on mathematics and physics. Jean-Marie Dautelle had been working on JADE (Java Additions to Default Environment) from 2000 targeting unit systems and mathematics. I was, on my side, working with Digital Biosphere, a 3D virtual environment struggling with geography, physics and astronomy. In year 2004, after having produced some code for Jsci, I offered Jean-Marie Dautelle to merge some code into a new project. He accepted, partly because he had copyright problems with the name JADE. From then on, we try to produce a general scientific API in Java.

The problem we faced is however not new. Many open source developers start building their project from scratch reusing few code from others, wasting time and energy because they are not able to find a good code base accepted fully by the community.

There is therefore a lot of unfinished scientific projects available over the Internet. One of the key ideas of JScience was to reuse this effort and adapt code from others to the new architecture rather than rewrite everything.

JScience, the concept

JScience aims to be a general science API in Java. We want to cover all knowledge (but databases contents that you will have to get elsewhere) up to graduation level or better. We also want it to be cost free and open source (delivered under GPL) because we want people to be able to use it the way they need it. Finally, we want JScience as a framework, that is, a conceptually sound set of tools that work together to provide something efficient.

We target researchers mostly but we hope to gain some interest from corporate projects. We, at JScience, are ourselves a bunch of engineers and researchers, mostly computer enthusiasts with common passion for science.

Our approach is to use Java 1.5, think of objects as first rank citizens (and use less utility classes), use XML support as much as possible and implement international and acclaimed standards whenever they exist. Yet, we admit this is a very new domain and we are therefore leading the way for hopefully better efforts.

The product

Our continuous effort over the last years has produced the widest science API effort ever (with more features than all existing C++ or Fortran scientific API).

We have one website for communication and two sub websites for development using the dev.java

platform provided by Sun Microsystems for open source developers. We maintain a couple mailing lists and bug tracking systems.

Our software packages are:

- Official distribution at version 4.3, around 200 classes distributed as a 2MB zip
- Experimental distribution at version 1.0b, around 4400 classes distributed as a 25 MB zip¹

Though the official package covers only core mathematics and unit systems, the experimental distribution reuses the code from the units systems (but a separate code for mathematics) and covers architecture, astronomy, bibliography, biology, chemistry, computing, earth science, economics, geography, history, law, linguistics, mathematics, medicine, physics, philosophy, politics, psychology and social sciences.

In fact, the only not covered by the experimental distribution is education. The reason is that there are many competing distance education frameworks around that would overlap and do a better job than JScience.

Benefits

The first benefit of using JScience is that it cuts down development costs dramatically. Not only developers have a very wide API to build their code upon, but JScience provides fast algorithms that normal developers would have trouble to design and debug should they do so. The reason is that scientific computing is hard and code is made by field experts. Though it is completely unsound, many developers start their development from scratch and try to redesign these highly technical algorithms.

Moreover, should you find a bug in JScience, or would like a new feature, we develop it for free and include it in the next release making sure that we have the most efficient algorithm.

On the opposite, you can ask that some new scientific module you have developed is added to JScience architecture. This means that your code can be part of the library and that you will only have to focus on your real application code in the future.

Of course, when you use JScience you have all the standard advantages of using Java software: portability, object oriented code, memory management, error propagation, internationalization, embedded XML support and auto generated documentation (a must see feature for a scientific API). This language helps to design elegant code (code quality) that is both easy to maintain and opens the way for implementation of new features without breaking the underlying model (extensibility)².

As it has been said earlier, Java is fast enough for scientific computation, almost as fast as its best competitor languages. Moreover, though we currently lack our own global benchmarking software, we have proven on our website that some parts of our code³ is even faster than the genuine Java

1 We have actually 3 distributions "levels" for the experimental code, each containing the classes from the one from lower level: core mathematics only, hard sciences and soft sciences. This is convenient for the users which can bundle a smaller API with their product. We also maintain a separate distribution without any Java code but only data files.

2 In fact, it would probably be impossible to write and maintain such an API in any other language than Java. The library would be too big in Fortran without object oriented support (aside Fortran 2003) and useless in C++ because it lacks a proper documentation tool. Doxygen can help to provide such a documentation but it is not a standard tool of C++ and generated documentation would therefore be not of much help to the users of this science library.

3 The JScience library is based on the real time Java library Javolution, which is designed by one of the JScience founders (Jean-Marie Dautelle).

code that comes from the Java Runtime Environment (the official implementation of the Java language from Sun Microsystems).

The idea to build a wide science library makes sense because we hope that this library will be someday (or at least parts of it) included in the core Java language distribution. As a matter of fact, the unit system used in JScience is now formally developed as JSR 275 (so called Units specification), the official Java Community Process (JCP) from Sun Microsystems for future extensions of the Java language API.

The people at JScience think that scientific computation should be a service available at OS level along with I/O, GUI or networking API. This is also why JScience tries to stick to all existing standards rather than define new ones. Especially, we try to support XML oriented data.

JScience tries to provide a model close to theoretical science concepts rather than just provide a numerical wrap up without correct foundations. As such, it supports infinite precision computation.

Finally, there is something very interesting happening with JScience. Because we have coded the underlying concepts of most sciences, each time we now add a new module, less and less code has to be produced because we reuse much code from the other modules.

For example, the

- org.jscience.sociology.Person class extends
- org.jscience.biology.human.Human which itself extends
- org.jscience.biology.HistoricalIndividual (an organism with some chronological data) which itself extends
- org.jscience.biology.Individual, itself basically a
- java.util.Set (or org.jscience.mathematics.FiniteSet) of org.jscience.biology.Organs, itself a
- java.util.Set of org.jscience.biology.Tissues, itself a
- java.util.Set of org.jscience.biology.Cells which contain
- org.jscience.biology.DNA or org.jscience.biology.RNA stored in
- org.jscience.biology.Genome as org.jscience.biology.Chains⁴.

Many other classes are used in the same way. This is very efficient because if you now need to access the DNA of the Person, you can get it without the need for a further line of code, because of Java object oriented features and JScience wide scientific spectrum approach.

For our team, it seems that JScience fills a gap in scientific software. If you only need a couple Matrix operations to support some computation in your application, JScience is probably not a good choice. On the opposite, if you need to use some units of measure, to display graphs, to access some standard data, to open some standard file formats and to have ready to use high level science targeted objects, JScience is a must see.

The future of JScience

We hope that in the near future we will get a wider interest from the community. We really look forward more support and recognition from other scientific computing people, especially the ones

⁴ Amusingly, these are in turn available as

- org.jscience.chemistry.Molecules made up of
- org.jscience.chemistry.Atoms treated as
- org.jscience.physics.Particles.

that have used traditional languages since then.

We also expect to grow in audience outside academic or big business companies.

We plan to enhance the API with some more packages. Yet our main effort will be to provide better, faster algorithms for existing features. For example, we plan to have better, explicit parallel support. Of course, we will also make some more documentation and design some new samples.

Possible trends for numerical computing

In my opinion, future progress in numerical computing in general will come from further research in a couple areas where work has already begun:

- Multi-inheritance: single inheritance of objects like in Java is not fully adequate for science.
- Solid foundations through direct translation of scientific concepts (arbitrary precision computation, axiomatics, true mathematics): packages are not numerical recipes but can be real food for thought and mirror the way science itself is designed
- New methods in numerical computing: new algorithms and new approaches (like finite elements for example) can be invented and implemented.
- Meta languages: use of parsers and some new languages (Fortress for example) could be more suited to describe some scientific domains than traditional programming languages.
- Models for social sciences: social sciences are only at the beginning, much improvement can be done by defining core concepts like it is done in physics, biology and chemistry.

Conclusion

JScience has made a reputation of its own in the small but growing world of scientific computing. It is regarded as quality software in the Java and some of its code is used as a base for the JSR 275 official Java language extension for Unit specification.

JScience is the starting point of a new generation of scientific API that makes use of object oriented features as well as garbage collection, just in time compilation and error management (thanks to Java). It is completely open source, fast, accurate and available now.

References

Ashby, J.V. (2004). *A comparison of C, Fortran and Java for Numerical Computing*. DL Technical Reports, DL-TR-2004-003. Available at <http://www.cse.scitech.ac.uk/arc/reports/jaspa.pdf>

Boisvert, R.F., Moreira, J., Philippson, M. & Pozo, R. (2001). *Java and Numerical Computing. Computing in Science and Engineering*. Available at <http://www2.informatik.uni-erlangen.de/Forschung/Publikationen/download/cise-ron.pdf>

Cary, J.R., Shasharina, S.G., Cumming, J.C., Reynders, J.V.W. & Hinkler, P.J. (1996). Comparison of C++ and Fortran 90 for Object-Oriented Scientific Programming. Submitted for publication in Computer Physics Communications, Nov. 1996. Available from Los Alamos National Laboratory as Report No. LA-UR-96-4064. Available at http://www.amath.washington.edu/~lf/software/CompCPP_F90SciOOP.html

Dautelle J.M. (2007). *Introduction to JSR-275: Measures and Units*. Available at <http://www.javaworld.com/javaworld/jw-10-2007/jw-10-jsr275.html>

Emmott, S. & Rison, S. (2006). *Towards 2020 science*. Microsoft Research, Cambridge. Available at <http://research.microsoft.com/towards2020science/downloads.htm>

Emmott, S. & Rison, S. (2006). *Towards 2020 Science: A Draft Roadmap*. Microsoft Research, Cambridge. Available at <http://research.microsoft.com/towards2020science/downloads.htm>

Java Numerics <http://math.nist.gov/javanumerics>

Java Programming Language Official Website <http://java.sun.com>

Javolution Frontend Website <http://www.javolution.org>

JScience Frontend Website <http://www.jscience.org>

JScience Official Backend Website <https://jscience.dev.java.net>

JScience Experimental Backend Website <https://jade.dev.java.net>

Hale, D. (2006). *The Java and C++ platforms for scientific computing*. In *2006 Project Review*, CWP-548. Available at <http://www.mines.edu/~dhalo/papers/Hale06JavaAndCppPlatformsForScientificComputing.pdf>

Hanlon, D. (2002). *Java for Scientific Computing: An Introduction to Java*. UKHEC Technical Note. Available at <http://www.ukhec.ac.uk/publications/guides/javaintro.pdf>

Kay, A., Ingalls, D., Ohshima, Y., Piumarta, I. & Raab, A. (2007). *Steps Toward The Reinvention of Programming ; A Compact And Practical Model of Personal Computing As A Self-Exploratorium*. VPRI Technical Report TR-2007-008. Available at http://www.vpri.org/pdf/steps_TR-2007-008.pdf

Kahan, W. & Darcy, J.D. (1998). *How Java's Floating-Point Hurts Everyone Everywhere*. Workshop on Java for High-Performance Network Computing held at Stanford University, 1998. ACM. Available at <http://http.cs.berkeley.edu/~wkahan/JAVAhurt.pdf>

Moreira, J.E., Midkiff, S.P., Gupta, M., Artigas, P.V., Wu, P. & Almasi, G. (2001). *The NINJA project: Making Java work for high performance numerical computing*. Communications of the ACM, vol. 44(10). Available at <http://citeseer.ist.psu.edu/moreira01ninja.html>

Sips, H.J. & Van Reeuwijk, C. (2001). *Java for Scientific Computation: Prospects and Problems*. In *Third International Conference on Large Scale Scientific Computation (LSSC2001)*, pp. 236-246. LNCS 2179. Available at www.pds.ewi.tudelft.nl/pubs/papers/scicomp01.pdf

Smith, L.A. & Bull, J.M. (2001). *Java for High Performance Computing*. UKHEC Technology Watch Report. Available at <http://www.ukhec.ac.uk/publications/tw/hpcjava.pdf>

Teukolsky, S.A., Vetterling, W.T. & Flannery, B.P. (1992). *Numerical Recipes in C: The Art of Scientific Computing*. WH Press, University of Cambridge, New York.

Teukolsky, S.A., Vetterling, W.T. & Flannery, B.P. (1992). Numerical recipes in FORTRAN: the

art of scientific computing. WH Press, Cambridge University Press, New York.

Teukolsky, S.A., Vetterling, W.T. & Flannery, B.P. (2002). *Numerical recipes in C++: the art of scientific computing*. Second edition. WH Press, Cambridge University Press, New York.

Thiruvathukal, G.K., Breg, F., Boisvert, R., Darcy, J., Fox, G.C., Gannon, D., Hassanzadeh, S., Moreira, J., Philippson, M., Pozo, R. & Snir, M. (1998). *Java Grande Forum Report: Making Java work for high-end computing*. In *Supercomputing '98: International Conference on High Performance Computing and Communications*, Orlando, Florida, November 7-13, 1998. Panel handout. Available at <http://www.javagrande.org/sc98/sc98grande.pdf>

Units and Measure Java Official Extension <https://jsr-275.dev.java.net>

Van Reeuwijk, C., Kuijlman, F. & Sips, H.J. (2003). *Spar: a Set of Extensions to Java for Scientific Computation*. Concurrency and Computation: Practice and Experience 15, pp. 277-299. Available at <http://www.pds.ewi.tudelft.nl/pubs/papers/cpe2003.pdf>