The Economics of OptimJ

OptimJ is a radically new optimization modeling language designed as a Java extension with Eclipse integration, in contrast with all existing modeling languages that are home-brewed domain-specific languages.

OptimJ for your optimization projects means tight IT integration, much faster time to market, less maintenance, and better runtime efficiency.

OptimJ technical overview

OptimJ™ is an extension of the Java™ programming language with language support for writing optimization models and powerful abstractions for bulk data processing. The language is supported by programming tools under the Eclipse™ 3.2 environment.

- OptimJ is a programming language :
    - OptimJ is an extension of Java 5
    - OptimJ operates directly on Java objects and can be combined with any other Java classes
    - The whole Java library is directly available from OptimJ
    - OptimJ is interoperable with standard Java-based programming tools such as team collaboration, unit testing or interface design.
- OptimJ is a modeling language :
    - All concepts found in modeling languages such as AIMMS™, AMPL™, GAMS™, MOSEL™, MPL™, OPL™, etc., are expressible in OptimJ.
    - OptimJ can target any optimization engine offering a C or Java API.
- OptimJ is part of a product line of numerous domain-specific Java language extensions, a novel approach of "*integration at the language level*" made possible by Ateji proprietary technology.

Simpler and faster development process

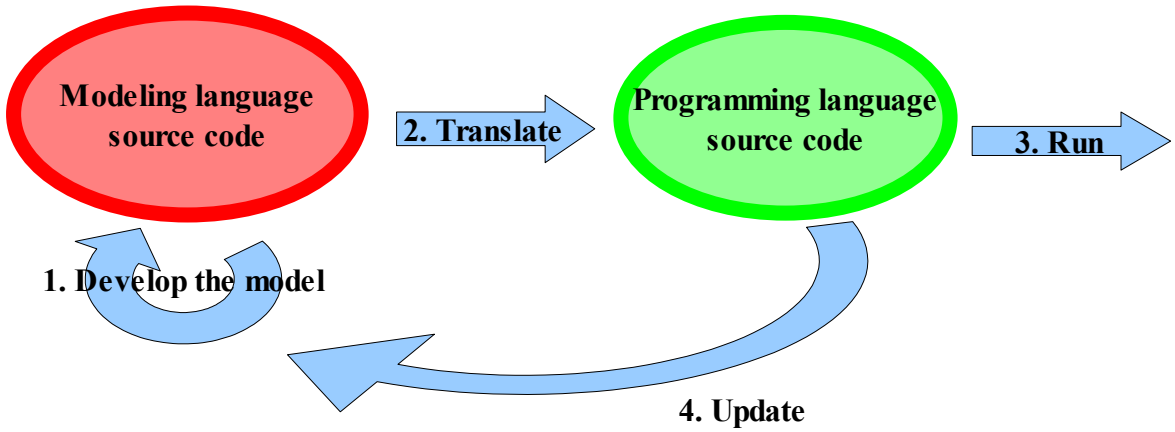
The usual approach is in two steps: prototype, then deploy

Most users of modeling languages use them only for prototyping a model. When they are satisfied with the model, they then translate it by hand into a mainstream programming language making direct calls to the solver API.

A modeling language is required for expressing, manipulating and above all understanding models (coding any realistic model from scratch with low-level API calls is simply not an option). But today's modeling languages have importants drawbacks:

- Efficiency: an extra runtime layer is required that has an important impact on performance.
- Platform compatibility: the resulting code must be executed in a specific environment, typically into a different process.
- Data feeding and solution reporting is problematic because of differences in data types and data formats, and require additional code for handling communication between different process. This interface code takes time to develop, is error-prone and slow at execution.

This leads to a development process in two phases, as depicted below.



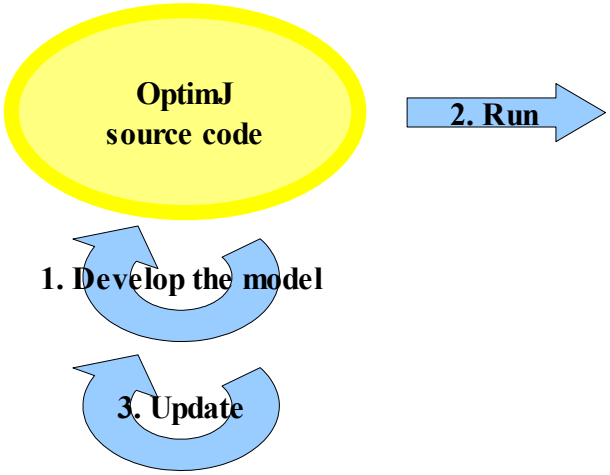
The hand translation part is complex and requires a few days work for an developer that must be expert in both the modeling language and the programming language.

A hand translation almost invariably introduces hard to find errors; so you'd better provision some additional time for debugging.

Updating the resulting application is problematic. Because there is no going back from the programming language source to the modeling language source, making changes means start again from scratch and redo part of the translation. Chances are you will stick with an out-of-date application rather than engage in a long update process.

With OptimJ, you prototype and deploy at once

Being an extension of Java, OptimJ is both a modeling language and programming. Compare the diagram below with the previous one:.



In a sense, OptimJ is doing the translation for you. The code generated by the OptimJ compiler makes direct calls to the solver API: it is as efficient as what you would have written by hand, sometimes even more because the compiler is able to perform some high-level code optimizations that are difficult to do by hand.

Data feeding and reporting is immediate and does not require additional code, because optimization models directly work with your application data.

No more hand translating models into code means shorter development times, less errors, and easier update.

Enhanced developer productivity

While major modeling languages come equipped with a house-built development environment, or no development environment at all, OptimJ is integrated in the state-of-the-art and widely spread Eclipse environment. This has an enormous impact on developer productivity.

Standard libraries and tools

Why learn yet another way of accessing databases, reading Excel files and designing graphical interfaces? And how comfortable would you feel relying on an optimization engine vendor to maintain these libraries and catch-up with the most recent advances in software technology?

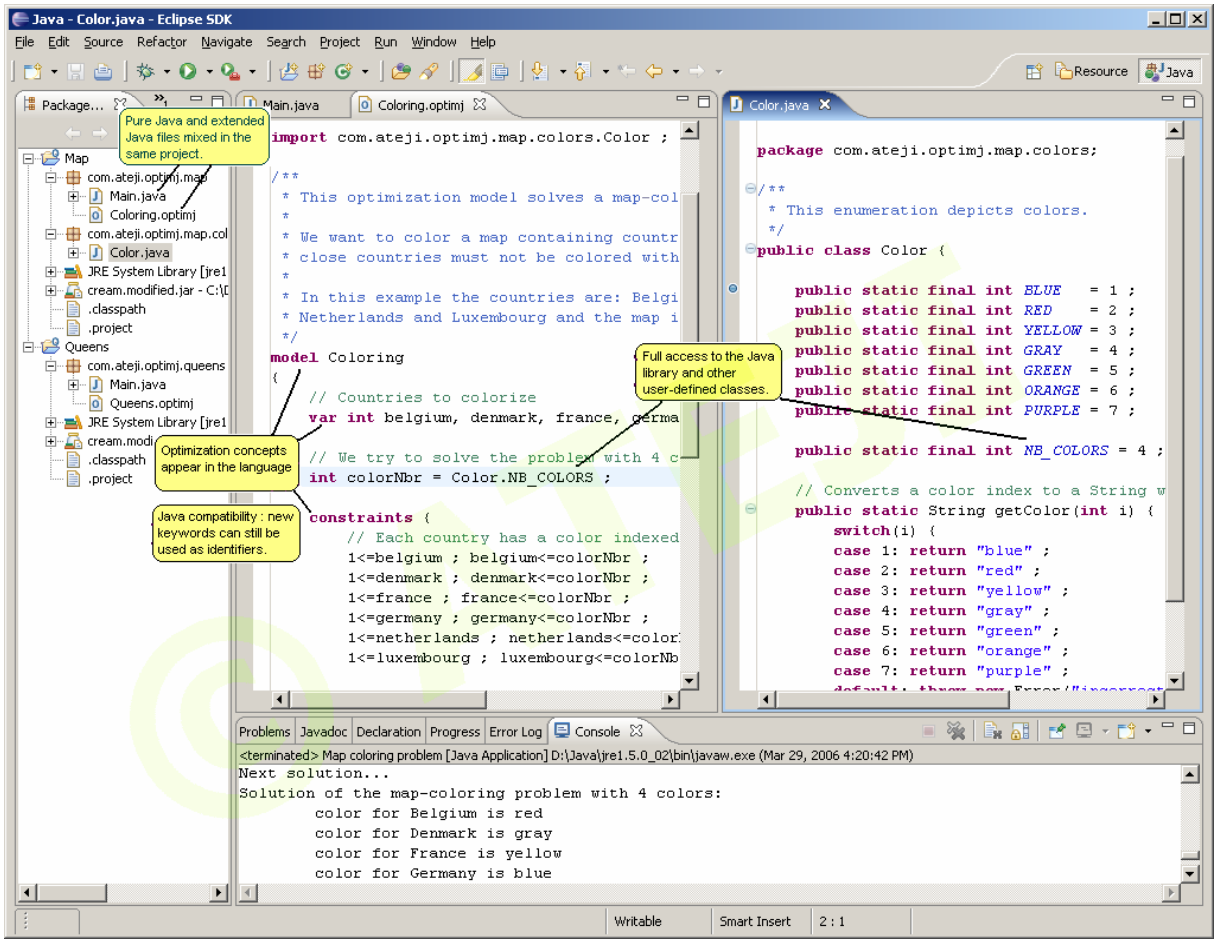
When modeling with OptimJ, you have direct access to the whole set of standard Java libraries and accompanying tools. GUI designers, XML parsers, JDBC drivers, Hibernate, Swing, Javadoc, CVS, JUnit, you name it, all can be used with your optimization model.

Direct interaction with the IT department

In any large organization, optimization projects must ultimately be embedded into a larger IT environment. With OptimJ, your optimization team delivers standard Java class files that work directly on the application data. Developers from different teams share and synchronize their work on a common CVS repository and collaborate on integration testing.

State-of-the-art programming support

The OptimJ support in Eclipse offers a language-aware editor with rich features such as syntax coloring, navigation, search, integrated debugging, etc., that are today the basic tools of any professional software developer.



Bulk data processing

In addition to algebraic modeling features, the OptimJ language also provides powerful and concise abstractions for bulk data processing. They prove very handy in the data preprocessing phase that is often required before stating an optimization model. Common examples are transforming couples of arrays into sets of couples or vice-versa, aggregating data and checking integrity constraints.

OptimJ features such as set comprehensions, list comprehensions and tuples enable writing in one single and intuitive line complex bulk data transformation processes that would require dozens of involved loops in Java.

Code reuse with Object-Oriented Modeling

Object-oriented languages are commonplace today in the software development landscape, and no reasonable programmer would imagine doing without them, but today's modeling languages are still lagging behind. OptimJ is a true object-oriented language, inheriting its object-oriented features from Java. This Object-Oriented Modeling approach enables structuration and reuse of both models and algorithms.

A given model is easily adapted to a different application domain by abstracting away data representation via standard OO techniques such as interfaces or proxy classes. Optimization algorithms such as column generation need only be written once and can be reused by abstracting away the specific models and the notion of what constitutes a pattern in a particular problem.

Fast project start-up time

All recent university graduates today in IT-related areas have a hands-on experience of Java and Eclipse. But how many of them know the vendor-specific modeling language that comes with an optimization engine and master its home-built software library and development environment?

Talent availability

Since OptimJ is an extension of Java with support under Eclipse, the required language learning is limited to the extended part, and the development environment is familiar. There is no need to learn yet another way to write 1+2 and yet another command to compile your code. Java experts can write, compile and run their first simple optimization model in fifteen minutes.

But OptimJ is also a modeling language, representative of the algebraic modeling languages family, and an expert in modeling will feel at ease with OptimJ in a matter of hours: the same concepts are present at the language level and syntax is familiar.

Team work

The secret of a successful software development project is efficient team work, such as having a database specialist and a web design expert working in close cooperation. Common software development tools and common languages are very important in this respect because they provide the basic communication medium.

The same is true for optimization projects and OptimJ provides this communication medium. Optimization experts feel at ease with the optimization part of the language, while programming experts master the Java part of the language.

They can work together and interact easily because they are using the same common language, the same development environment, the same building tools, and share their work on a common central repository.

Fast installation and set-up

OptimJ is delivered as a standard Eclipse plug-in and is installed in a few minutes using the Eclipse plugin manager.

Less maintenance

The later you identify bugs, the more they will cost in maintenance. Many OptimJ features help preventing bugs or identifying them early::

- strong typing (inherited from Java) forbids ill-behaved programs
- navigation features of the development environment help grasping the overall logic of large programs
- debugging features on OptimJ source code help locating problems

In today's software applications, many bugs lie at the interface between languages, hidden in XML configuration files or other middleware connectors. They are particularly difficult to track because language tools such as debuggers or static analysers cannot cross the language barrier.

With OptimJ, integration is done at the language level. This means zero interfacing code and zero configuration files, and much fewer places where hugs can hide.

No performance penalty at runtime

Current modeling languages include a runtime layer that serves purposes such as abstracting away from differences in solver engines or implementation of advanced language features.

A runtime layer can incur a severe runtime performance penalty, typically requiring allocation of additional memory objects for each variable or constraint of your model. Not only is this a time-consuming operation (in some examples, stating a model can take much longer than actually solving it), but it can quickly exhaust all the available memory. Reaching the limits of physical memory because of a runtime layer overhead means that you simply cannot state your model, even though your solver would have been able to solve it.

OptimJ is a full language compiler and does all its work at compile time: there is no additional runtime layer whatsoever. This is why OptimJ is able to generate code with performances similar to what you would obtain by hand-coding directly against a low-level solver API.

Cost-effective licencing scheme

OptimJ is reasonably priced compared to all major modeling languages, and is licensed as a programming tool. No license is required for deployment.

No matter how many projects your developers are working on, no matter how many customer sites are using your code, you will need only one license per developer. Your investment in OptimJ quickly pays off over the course of a few projects.

Try OptimJ

A free evaluation version of OptimJ is available for download at www.ateji.com, with additional technical information and numerous code samples.

Ateji helps you jump-start your first OptimJ projects by providing consulting and education services as needed. Contact us at info@ateji.com