Coordination of Software Development in COBIK and Laboratory for Multiphase Processes

Revision 2, June 2011.
(Revision 0: November 2010)

Igor Grešovnik

Contents:

1 INTRODUCTION & BACKGROUND

This is the cover document for coordination of activities related to software development in Prof. Božidar Šarler's groups at COBIK & University of Nova Gorica.

In COBIK (Centre of Excellence for Biosensors, Instrumentation and Process Control) we hve a roughly 3 years long project with the aim of developing software for numerical simulation and optimization of arc-discharge reactor for production of fullerenes and carbon nanotubes. A Ph.D. student will develop a numerical model based on collected literature, while I will be in charge of developing software for optimization and inverse identification of model parameters (adopting and refining some ideas developed in IOptLib [4]).

In the Laboratory for Multiphase Processes, extensive knowledge has been accumulated in the field of numerical simulation with meshless techniques, with years of experience in modeling of physical phenomena related to solidification of metals and collaboration with steel and aluminum production industry. However, software was developed on ad hoc basis, development was mainly done by Ph.D. students each of which developed and used his/her own code. Such approach turned effective in the past, but poses a number of limitations for further development. As problems to be solved in the future are becoming more complex , interdisciplinary and interwound, the need for more systematic and correctly managed software development bocomes evident.

It was decided that I will initiate and lead development of a common simulation framework that will be used in the Laboratory for Multiphase Processes, and also shared by COBIK for development of numerical models of fullerene production. The development of this framework will significantly improve efficiency of software development in the group, it will provide a platform for permanent inclusion of research achievements, and will simplify introduction of new employees into working proces. The framework will be designed profeccionally, it will be scalable, extensible and modular. Initial momentum will be provided by new Ph.D. students while current work (of Gregor Kosec, Robert Vertnik, Umut Hanoglu) will be slowly added later to enable transition from existing codes to a common simulation code. The framework will be intended for both academic work (including Ph.D. theses) and for development of industrial applications, which is a particular challange in code design. Joint use by UNG and COBIK will be beneficial for both institutions and will cause additional synergetic effects.

The present document was created to detail the ideas of how software development will be organized, to argue and clear important details such as choice of platforms, etc.

2 CURRENT STATE, ACTIVITIES & PLANS

2.1 Current Activities

2.1.1 Test case for NAFEMS heat conduction test in C#

Task leader: Robert

- Check for numerical and graphical libraries that were used in FORTRAN codes (or possible substitutes) – Robert
- Development of a small simulation code – Robert
    - Help with input files reading and input forms – Igor
- Setting up the test and checking results – Robert
- Overseeing the development – Igor

2.1.2 Choice of basic platforms for software development

Task leader: Igor

- Starting activities – Igor, Robert
- Search for libraries availability on different platforms – Robert, Igor
- Setting up basic requirements - Igor
- Evaluation of platforms with respect to requirements – Igor, Robert, …
- Inclusion of group members for remarks, comments, suggestions – Igor
    - Collection of information about what individual group members are working on, which are their requirements in terms of software, what are main interactions with others. Information is gathered in individual meetings. - Igor

2.1.3 Short course on C#

Reading:

- Harness the Features of C# to Power Your Scientific Computing Projects

Task leader: Igor

- Information about intended course, scheduling
- Preparation of material
- Performing the course
    - Delivering individual topics
    - Suggesting exercises & additional reading
    - Identification of possible issues, help to overcome them

2.2 Plan

2.2.1 Main Tasks and Assignments

2.2.1.1 Construction of general framework – Robert, Igor, Gregor Kosec

A complete simple example (NAFEMS heat con1duction test) is first coded in C# - Robert
On this example, basic structure will be created – Robert, Igor, Gregor

Basic design requirements:

- Easy switching between 2D and 3D
- Easy coupling with other numerical codes, e.g. thermal code with external mechanical code to obtain plastic heat generation
- Enable micro-macro modeling
- Enable re-meshing
- Enable multiple joint domains with different material properties and possibly with different physical laws, but with shared boundaries and therefore shared simulation point co-ordinates
- Extensible material properties
    - Enable definition of material properties through external calculations, consider various arrangement, e.g. accumulation through time stepping procedure, cases where storage of history variables is required
- Enable non-local time and domain effects (limited or unlimited domain), e.g. relaxation
- Flexible definition of material properties and clear & efficient rules to access them

2.2.1.2 Linear Solvers

Somebody keeps tracks of linear solvers suitable for inclusion in the framework and accessible under suitable conditions. Also implements inclusion of the solvers, including design of how libraries are included, implementation of wrappers, and inclusion in the standard class library of the framework.

Main requirement for solvers one should consider:

- Efficiency
  - Efficiency of system of equations assembly
  - Efficiency of the solver itself
- Integration suitability
  - Prices
  - Compatibility of licenses
  - Platforms for which the specific solver is natively available

For large scale industrial problems, solvers have almost always the predominant impact on the overall CPU efficiency of the simulation code. Usually, not using a solver with sparse storage also has adverse impact on memory usage. Somebody should therefore maintain a good overview of what is the current state of solvers market, and should be skilled in integrating a variety of solvers into the simulation framework.

Licenses should be carefully examined before integrating a particular solver into the framework. Some licenses (in particular some open source licenses) will not be compatible with our framework because of the restrictions they impose. A typical example is the GPL license, which de facto requires that if some software is linked with the respective libraries (the term used by the free software community is “derived from”), its free open source must be provided under the GPL compatible license, which in effect bans many possible business models for generating revenue by your software. In the case of proprietary licenses, the license cost may be a limiting factor what regards usability of the solver. Some licenses require payment only for development versions, while compiled code linked with your application can be freely distributed to the users of your software.

Solvers are typical example of functionality for which we will probably have to consider linking of code written for different platforms. This is because many solvers are available only in lower level native programming languages (such as C, FORTRAN or even partially in assembler) due to their performance critical character.

Within the software framework, there should be a unique API (application programming interface) for interacting with the solver. All built in solvers should therefore be wrapped into such an interface, such that usage of the solver is uniform to developers.

Many top-end solvers are commercial. While freeware equivalents exist, they may be much less efficient. By defining a common API for all solvers, it will be easy to switch between different solvers (it must also be possible to do this dynamically at the application level). In this way, we can use expensive commercial solvers on high performance systems used to run industrial simulations or in commercial installations of simulation software at customers. For development on local machines, freeware substitutes can be used.

2.2.1.2.1 Graphics

Suitable graphic tools should be gathered or implemented that can be used to represent all possible kinds of results that can be generated by the code.

Graphics is used for viewing and quick verification of results, for inclusion in reports and articles, and for presentations. It is important for developers to have easily accessed tools for presentation of results, which enable first verification of their code, and it is also important to be able to produce quality, readable and good looking representation of results for inclusion in papers and presentations. Graphical representation of results will usually create the first impression of our work to potential partners and customers.

There are two main approaches to graphical representation of data. One can either use an external graphical engine (such as GnuPlot, Mathematica or Matlab) and export graphics in format that is understood by such software, or can show and export graphics by using library routines linked to the code that produces and manipulates the data.

The first approach may seem easier and quicker from developer’s perspective because the external engine provides many high level functionality such as decorating graphics with titles, labels, gridlines, coordinate marks, etc, or user interaction capability for zooming in and out, rotating,, exporting in different formats, etc.

On the other hand, use of external engines is less flexible because high level functionality is not so easily extensible, it requires cumbersome preparation of output (which may include generation of scripts), and it usually takes some user interaction in order to properly transfer the presented data to the graphical software. From this point of view, developing and using a graphics layer based on good general purpose graphical libraries may be a better choice, and should be a long term solution for graphical processing. A good graphical library that can be well integrated with GUI module can enable, after a small initial investment, much faster generation of results as the approach with external engine.

Building general graphic utilities for the simulation framework is extremely important on long run, but is quite low on priority list as compared with other functionality. On the other hand, developers will miss such functionality a lot until it is provided. The main problem with graphics is that it spans several levels in software hierarchy, and valid implementation requires a lot of programming knowledge and experience, which we will lack badly before the development team is well-trained. The solution envisaged is that people can develop different temporary ways of graphical presentation of results, but try to implement these in such a way that others in the team can use them and also contribute to them. What we can do in the very beginning is to make research of possible candidates for graphical libraries and implement some basic stuff based on these libraries.

Requirements that should be considered well when making decision about the base graphical libraries of the framework are the following:

- Ability of generation of different output formats.
    - At least one general vector format should be included.
- Good use of graphic hardware (for efficiency reasons).
- Good graphic capabilities (quality output without defects, smoothing, interpolation, definition of lights, transparency, shading, etc.).
- Availability of general high level functionality such as presentation of data based either on meshes or clouds of points, calculation of surfaces from volumetric data, generation of contour plots, cross sections, proper rendering of intersections, etc.
- Good presentation of different mathematical objects such as meshes, vectors, contours, streamlines, etc.
- Possibility of generating animations programmatically (either built-in or achievable through generation of frames that can be used by external libraries to show and export movies).
- Implementation of various decoration utilities such as titles, labels, grids, boxes, value marks, etc.
- Good integration with GUI development (e.g. graphical windows enable capturing events, transforming views and light positions is straight forward), possibility of integration with graphical GUI builders, etc.
- Availability for different platforms (we should e.g. libraries that are available for .NET but not for MONO, and vice versa)
- License limitations
    - Should not ban any of intended uses of the platform, including commercial or open source sistributions
    - Cost; it is very desirable that the library is freely available. Somewhat acceptable alternatives are development licenses where you pay per developer but can distribute the products without additional costs.

Payable libraries may provide more functionality, but would make the framework much less attractive for open source distributions. Possible solutions may include using two presentation layers where one is free and less capable, and the other is payable.

2.2.1.2.1.1 2D Graphics

2D graphics for simulations can be based on the same lirary as 3D graphics, or it can be based on a separate library. The advantage of using the same library is more unified development and less effort necessary for introducing new developers. On the other hand, a specialized 2D graphic library may be easier and more efficient to use. We can therefore begin with developing 2D graphics on a separate library and later re-implement functionality and integrate it into the common 2D/3D system.

Beside 2D graphics for presentation of simulation results, we will also need charting abilities for plotting various dependencies. This may be implemented as a separate module based on a separate specialized library because such libraries can have very specialized and elaborated features for this purpose (an example of this is the Zedgraph library).

2.2.1.2.1.2 3D Graphics

We currently don’t have a 3D simulator, but 3D graphics can be generated from 2D slice models and most likely we will need to develop fully 3D simulation tools in the future, too.

Since developing 3D graphics can be a relatively demanding task, we can do this in smaller steps each time we need something, and develop a more general module later after gaining extensive experience.

For a final solution there are different options. We could utilize some external engine such as GID that is used for mesh generation and presentation of results in finite element. This could be a quick and easy solution bringing some other benefits (such as meshing). However, I would argue for use of a graphic library because it can be better integrated in out system and this approach would allow more freedom and flexibility.

There are some easy-to-use free libraries available, such as DISLIN. The alternative are more basic libraries, which are more difficult to use, but on the other hand provide much more power. A very attractive candidate in this category would be the VTK library (Visual Toolkit), which has a lot of powerful features, can produce very attractive and clear output when used properly, and incorporates built-in user interaction utilities. It would definitely be a good long-term choice because it would probably meet any requirement that we could have in the future, which can outweigh the smoehow larger effort needed to use it. The library is wide spread in very demanding medical applications.

2.2.1.2.1.3 Integration of Graphics

One part of graphic modules development is development of well structured 2D and 3D libraries, which developers can use very efficiently to present anything they want.

The other part is integration of graphic capabilities created in this way into the simulation framework. This includes definition of user interfaces such that the user can interact with simulation software and its pre- and post-processing capabilities without having contact with the code.

Typically, user interfaces will be in form of GUI, but there can also be a more flexible user interface, e.g an user interface built around an interpreter.

Graphics will typically be integrated with simulation code in the top-most software layer. Integration in lower layers is also possible, e.g. in the case where real-time graphics is required. However, this must be implemented in such a way that concrete implementation of graphics is completely separated form the code (by using abstraction properly) and can be hooked on the code on demand. This hooking should be preferable implemented in such a way that any other processing could be hooked on instead of graphics (or in addition to graphics), e.g. procedures for exchange of data with coupled simulation codes.

2.2.1.3 Geometry Definition, Presentation and Mesh generation

Goal is to establish a system for definition of complex geometries, use of geometric definitions within simulations (e.g. for contact detection and calculation of contact terms).

Wish list:

1. Geometric definition used in the framework should be compatible with standard CAD formats.

2. Meshing tools should work with abstract geometrical definitions, either directly or indirectly (with intermediate transcription of geometry into mesher-native formats.).
3. Reverse should also be possible – to generate abstract geometrical models from meshes (either non-deformed or deformed).
4. It should be possible to manipulate geometrical definitions from the code.
5. Presentation module should be able to show results or meshes superimposed to geometrical definitions (preferably with ability to assign define transparency and other optical properties to graphical representation of geometry).

Definition of geometry of simulated objects is linked with representation of continuum geometry used within the numerical models, therefore good integration of both is of primary interest. At least in the beginning it is not feasible to develop a full scale CAD representation of geometry, therefore the emphasis will be on seamless integration of existent tools (such as CAD systems and meshing tools). Choice of the right software to rely on is very important for this task, and tools for importing, exporting, and interaction of native geometrical representation of such systems will be considered. This also means that internal geometry representation will be built, with efficiency and compatibility issues always kept in mind and with knowledge of that continuously updated.

2.2.1.4 Definition of test cases

Since the beginning we should maintain a set of test cases. When the system evolves, test cases (i.e. input formats, formats of results etc.) will be changed as we go.

Test cases are maintained in order to enable the following:

- Enable testing of correctness of code
- Testing of efficiency (e.g. when studying a new solver)
- Testing that nothing is corrupted when new functionality is added or existing implementations are modified
- Following of stability of the code with respect to input/output formats, etc. (we will strive for invariability backward compatibility of input formats, a much as this is feasible without affecting efficiency and good design of the code).
- Quick demonstration of software capabilities for potential customers and partners
- Quick and plastic introduction of newcomers into the code (code structure as well as pure usage).

There will be a growing set of test cases. Maintenance of some can be dropped in order to reduce maintenance costs, especially when their coverage is replaced by more elaborated tests.

We will maintain some test cases particularly for education purposes. This will enable somebody to learn about just specific portions of the complex system almost as effectively as if he/she studied a simple code including just the minimal subset of functionality.

2.2.1.5 Industrial Applications

2.3 Motivation and Justification

2.3.1 Statement of Objectives

Primary goal is to establish systematic and quality development of simulation and optimization software based on modern software architecture and managed according to generally recognized good practices of software development. This will include:

- Design and systematic development of code base
    - Basic libraries
        - Modular development
        - Logical hierarchy
        - Identification and inclusion of standard set of external libraries (paying attention to possible license conflicts, quality and usability of libraries, complementary libraries to avoid overlapping, etc.)
    - Common simulation framework
    - Common optimization framework
    - Applications
- Establishment of procedures and tools that will enable team work and quality assurance
    - Definition of coding standards (examples are [1] and [2], but ours will be quite different)
    - Systematic software design in order to achieve:
        - Modularity
        - Extensibility
        - Readibility fo code
        - Efficiency of work
        - Minimal doubling of work
    - Setting up revision control system (probably Subversion)
    - Definition of testing procedures
    - Itroduction of Peer review
    - Division of assignment within the group (e.g. testing, taking care of software and hardvare)
    - Arranging optimal knowledge covering and division of specialties (with partial overlapping to ensure availability of sufficient development potential for any upcoming task)
- Development of key products that can be marketed:
    - General purpose libraries
        - Especially free & open source
    - General purpose simulation framework and general purpose optimization framework (can be stand-alone, but are also pre-integrated)

        - Academic and demo versions for popularization of our work
        - Commercial releases when mature enough
        - Architecture: common framework with modules for particular disciplines (e.g. thermal, linear mechanical, plastic, etc.) and techniques (e.g. domain decomposition, micro-macro, discrete elements, etc.)
- Development of tailored applications by demand (this is what is done now for the most of the time):
    - Based on common framework in order to enable rapid development
    - Coding standard released in order to meet deadlines
    - Introduction of standard project management techniques with time
    - Gradual division of work to development and short-term project work

2.3.1.1 Considerations

Establishment of systematic development of code according to high level quality standards will take quite a lot of effort at the beginning. This must be considered a long term investment that will pay off in the future, especially by:

- Increasing efficiency of development (especially by prevention of doubling of work, continuous accumulation of useful code, good documentation, etc.)
- Increasing quality of code
    - Established quality assurance procedures will probably be more and more often required by customers, therefore this is necessary to keep us in the business
- Popularizing the group by releasing recognizable products
- Ability of rapid delivery of customized services and products on demand, which is very important for industry (which usually has tight deadlines)

The most difficult will be transition period where a lot of decisions will have to be made that are not so obvious. Introduction of more systematic software development should not disturb the ongoing work too much, but it should still be quick enough if we want to achieve the desired effects.

Applications that are already working in production environment or are close to completion will be left as they are, at least for quite some time. Support for these applications will be provided in the same way as before. Only when enough “meet” is accumulated, some old applications may be replaced, especially those that have good prospects for the future.

2.4 Overview of Numerical libraries

We need the following:

- Matrix operations (practically everything is needed for optimization)
- Sparse matrix operations (decomposition, eigenvalues, iterative solvers)
- Special functions
- Fourier transformations

- Numerical integration, interpolation, splines,
- Ordinary differential equations,
- Smooth optimization

2.4.1 Numerical Libraries Available for C#

2.4.1.1 Math.Net , Math.NET Numerics

Already used by IGLib. LGPL license for the numerical part.
Read here about sparse matrix support.
Features of Iridium (numerical part):

- Extensive full matrix support, including LU & decomposition, eigenvalues, SVD decomposition, basic operations such as matrix summation & multiplication, norms, etc.
- Complex type with many operations
- Non-uniform probability distributions, multivariate distributions, sample generation
- Polynomial interpolation, splines
- Numerical integration
- Fourier transformations
- Special functions, constants
- Combinatorics, polynomials
- Fully managed, object oriented style, extended by IGLib

Math.Net numerics: Math.Net Iridium and dnAnalytics have merged into this library.

2.4.1.2 dnAnalytics – merged with Math.NET Numerics

2.4.1.3 DotNumerics

C# implementations of Kapack, Blas and Eispack. Also includes some methods for differential equations and unconstrained optimization.

2.4.1.4 ILNumerics

The library does not currently support sparse matrices, but this is promised for the future (see FAQ).

Features:

- Linear algebra: LU, QR, SVD, and Cholesky decompositions, eigenvalues, eigenvectors
- BLAS, LAPACK
- Object oriented design

2.4.1.5 Alglib

Multi-language library, includes various numerical routines, from sparse sources.

2.4.1.6 IMSL

Commercial numerical library, it is available for C#.

2.4.1.7 Extreme Optimization

Commercial, for .NET, also includes sparse matrices.

2.4.1.8 NMath

Commercial .NET library, also includes sprase matrices.

2.4.1.9 Mapack.NET

Here is the original web site. It seems this library is not developed any more because in versions found on the internet it is stated that the library is for .NET 1.0. It implements basic full matrix operations in pure C#, the library is simple to use.

Statement:

Mapack is a .NET class library for basic linear algebra computations that supports a large number of matrix operations and properties.

It supports the following matrix operations and properties: Multiplication, Addition, Subtraction, Determinant, Norm1, Norm2, Frobenius Norm, Infinity Norm, Rank, Condition, Trace, Cholesky, LU and QR decomposition Single Value Decomposition, Least Squares solver, Equation System solver and Eigenproblem solver.

The algorithms were adapted from Lapack and the Java Matrix Package.

The Mapack.zip download package contains both the Library and the C# source code.

2.4.2 Numerical Libraries Used within the Group

LAPACK (Linear Algebra Package).

2.4.3 Graphical Libraries Available for C#

2.4.3.1 ActiViz

Pricing.

A good long term choice for graphical library would probably be VTK. Currently there is only a commercial C# implementation called ActiViz available. Single developer license costs 2900 $. The library is also available free of charge for personal use, but in this case it can not be redistributed and it prints a watermark on each graph. Possible solution is that we by one full license for compiling of commercial applications, while personal licenses are used on individual machines.

2.4.3.2 Microsoft WPF (Windows Presentation Foundation)

Included in .NET, but WPF livraries are not included in Mono framework (it is a question whether they will ever be)!

2.4.3.3 EyeShot (Commercial with trial version)

Recommended by Tomaž Tekavec. Sais it's a good library.

2.4.3.4 Microsoft XNA

This is a Microsoft platform for game development in .NET. It probably doesn't have the right license in order to use it in our code development, and it can also not be used on multiple platforms. However, game engines can be one option for base libraries for computer graphics in simulation framework, therefore it could be beneficial to look at it a bit.

You can start with examples, and you can obtain some nice examples from Visual Studio's Extension Manager (under Online Gallery/Templates/XNA Game Studio). You can open the Extension manager through the main menu/Tools/Extension Manager.

2.4.3.5 DISLIN

Library used by Robert. It seems that DISLIN is available for C#. Library is free for non-commercial use, for commercial use it has affordable prices.

2.4.3.6 ILNumerics

Also includes graphical library, but the problem is that it is available only under GPL license, which limit possibility of use in commercial applications. It is also the question whether the library is strong enough for our needs.

2.4.3.7 SharpGl

C# wrappers for open GL, probably not strong enough for us.

2.4.3.8 VisIt

VisIt is interactive software for scientific visualization. Maybe numerical software could be integrated with it as a plugin.

Free open source (BSD license)

2.4.3.9 ParaView

Open-source, multi-platform data analysis and visualization application, BSD license, works as client-server.

2.4.4 Graphical Libraries Used within the Group

2.5 Internal Codes

This section contains data about computer codes that have been developed or are being developed within the group.

2.5.1 Code for Continuous Casting Simulation in Štore Steelwork (Robert Vertnik)

3 ESTABLISHMENT OF SYSTEMATIC SOFTWARE DEVELOPMENT

3.1 Plans

- Connect current codes with interfaces
    - Integration mainly by data exchange through files
- Uniform development of code
- Establishment of **revision control system** - Subversion server
- Establishment of issue tracking & ticketing system
- VPN for access from anywhere

3.2 Choice of Development Platform

We will strive to concentrate all software development on one or two development platforms. According to the nature of our work, the main criteria for choice of these platforms are:

- Efficiency, especially in terms of CPU usage
- Availability and price
- Well elaborated language concepts suitable for development of complex applications
- Availability of numerical libraries
- Availability of suitable representation layer
    - Graphical libraries with good 3D support, suitable for representation of scientific results
    - Possibility of good integration with GUI
- Availability of basic utilities
    - Input/output
    - GUI building
    - Database connectivity
    - Web communication
- Availability of other libraries
- Possibility of deployment of stand-alone applications (independent of expensive packages)
- Portability
- Support (documentation, examples, etc.)
- Prospects for the future
- Popularity

Figure 1 shows main groups of programming languages.

Due to extensive base of readily available mathematical, numerical and graphical tools, *Mathematica* or *Matlab* could be used as basic platforms. However, there are some disadvantages related to these systems, in particular dependence on relatively expensive commercial package, not very comfortable programming environment (language syntax is not designed for rapid development, debugging is quite difficult), slowness in comparison to other languages, bad support for GUI building and system tasks such as input/output or web communication, difficult integration with other environments, etc. For these reasons, it would be better to use such systems for specialized tasks when sensible, and build interfaces with other software in such cases.

The most perspective candidates for the development platform seem to be C++, Java and C#.

Advantages of C++ are speed, rather good object oriented language design, and wide availability of numerical and graphical libraries. Because of wide availability of compilers for all platforms, portability of products is relatively good. Still there are some subtle differences between different C++ compilers and even between implementations of the same compiler on different platforms, which can be rather annoying when porting applications. Portability is particularly problematic in the area of GUI and other system dependent things such as database connectivity, web communication, etc.

Alternatives are managed Java and .NET (with C#) execution environments. As compared to C++, the main difference is that Java and C# do not have pointers. Because of this, coding is much easier, especially for unskilled programmers, and the whole range of possible programming errors (many of them very persistent and difficult to discover) vanishes on this account. On the other hand, the programmer does not have complete control over dynamic memory deallocation (since this is automatically performed by runtime environment’s garbage collector), which can be accompanied by performance penalty in some cases, especially where clever memory handling can exploit system architecture in order to achieve peak efficiency. Yet on the other hand, such intended optimizations require very high programming skills and are often not exploited anyway.

One great advantage of Java and C# frameworks is that they rigorously standardize a very large code base across all platforms on which they are available (including GUI, input/output, web communication, database access, etc.). In particular, Java comes with a large standard set of free developing tools and wide standard codebase incorporated in the framework. Hwever, the .NET framework with C# is more elaborated and enables more efficient development. The Java programming languages has some deficiencies that are inherited from the past (since Java was the first widely used platform of this kind, while .NET development has started later and incorporated many lessons learned from Java). For example, Java does not know calling by reference, generic types in Java are much more limited and do not ensure type safety to the extent as C# generics do, GUI building tools are much more elaborated in C# as in Java and the same is true for many other specialized areas (e.g. thread synchronization support, which is very important for numerical applications). Building large applications is much simpler in C# than in Java. It also seems that C# has currently much better development potential than Java, and it should be the preferred choice.

What concerns the development environment and portability, Java has some advantages over C#, but these do not prevail. The situation with C# is as follows. On Windows OS, the best development environment for C# is *Microsoft Visual Studio*. Its *Express edition* is available free of charge and it has all functionality most of developers will ever need. It lacks support for some specialized tasks, e.g. for building web services. The solution is then that developers who would deal with these things are provided with payable professional versions of Visual studio while others use free versions.

Cross platform open source implementation of .NET and C# development environment exists and is called *Mono*. Mono does not include everything that Microsoft .NET implementation (available only on Windows and Mac) does but most of the libraries we would need are available (including Windows Forms for GUI). Apart from what is missing in Mono, both implementations are compatible, there this should not be too much of a problem when we would need to port C# applications to other platforms.

The .NET framework provides an extensive systematically arranged code base and C# is a highly elaborated, simple to use and well designed object oriented language with many advanced features (such as generics, strong type checking, array bounds checking, detection of access to uninitialized variables, garbage collection, suitability for deployment in distributed environments, extensive internationalization support, well designed exception handling, reflection). Because of this, *C# is proposed as development platform for control applications* including optimization shell.

For *simulation core development platform*, the choice will be made *between C# and C++*. The advantage of using C# would be that majority of the development is performed on a single platform. But for C# to be acceptable, two basic arguments should be verified, namely the speed of C# code in comparison with C++ and the availability of important libraries such as numerical (especially those for sparse matrices) and graphical (suitable for use with scientific and technical computing).

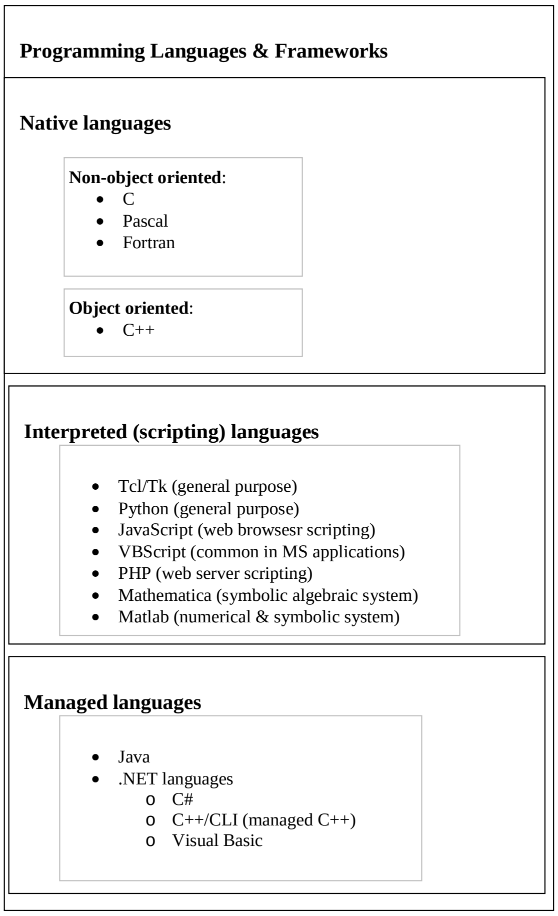
Currently it seems that numerical libraries will not be problematic.

For graphical libraries the situation is not well explored yet. It seems that DISLIN that was used by Robert is available for C#. A good long term choice for graphical library would probably be VTK. Currently there is only a commercial C# implementation called ActiViz (actually these are wrappers around the C++ library) available. Single developer license costs 2900 $. The library is also available free of charge for personal use, but in this case it can not be redistributed and it prints a watermark on each graph. Possible solution is that we by one full license for compiling of commercial applications, while personal licenses are used on individual machines.

Another concern in the case that C# is chosen may be *how to port the existent software* that was created *in C++*. The long term procedure would be to manually translate all the code form C++ to C# (we could also check whether automatic translators are available). This is not too difficult since C# syntax is similar to that of C++. The main problems would arise from pointers, incompatible libraries, lack of multiple inheritance in C#, etc. Another possibility would be to use managed C++ available on the .NET platform. In this case only parts of code where pointers are used should be corrected. However, this solution is available only on Windows with .NET, because Mono does not support managed C++.

See also:

- Harness the Features of C# to Power Your Scientific Computing Projects



Programming Languages & Frameworks

Native languages

Non-object oriented:
- C
- Pascal
- Fortran

Object oriented:
- C++

Interpreted (scripting) languages

- Tcl/Tk (general purpose)
- Python (general purpose)
- JavaScript (web browsesr scripting)
- VBScript (common in MS applications)
- PHP (web server scripting)
- Mathematica (symbolic algebraic system)
- Matlab (numerical & symbolic system)

Managed languages

- Java
- .NET languages
  - C#
  - C++/CLI (managed C++)
  - Visual Basic

Figure 1: Main groups of programming languages with some common representatives.

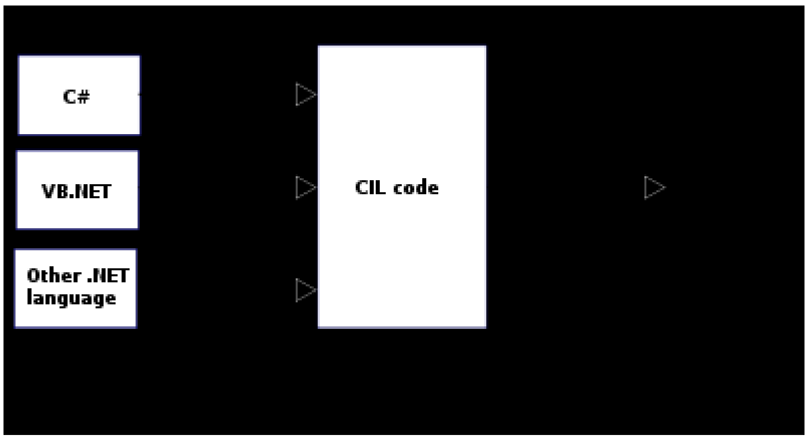


Figure 2: All .NET languages are translated to the common intermediate language.

4 OPTIMIZATION TOOLS

4.1 Basic Optimization Scheme

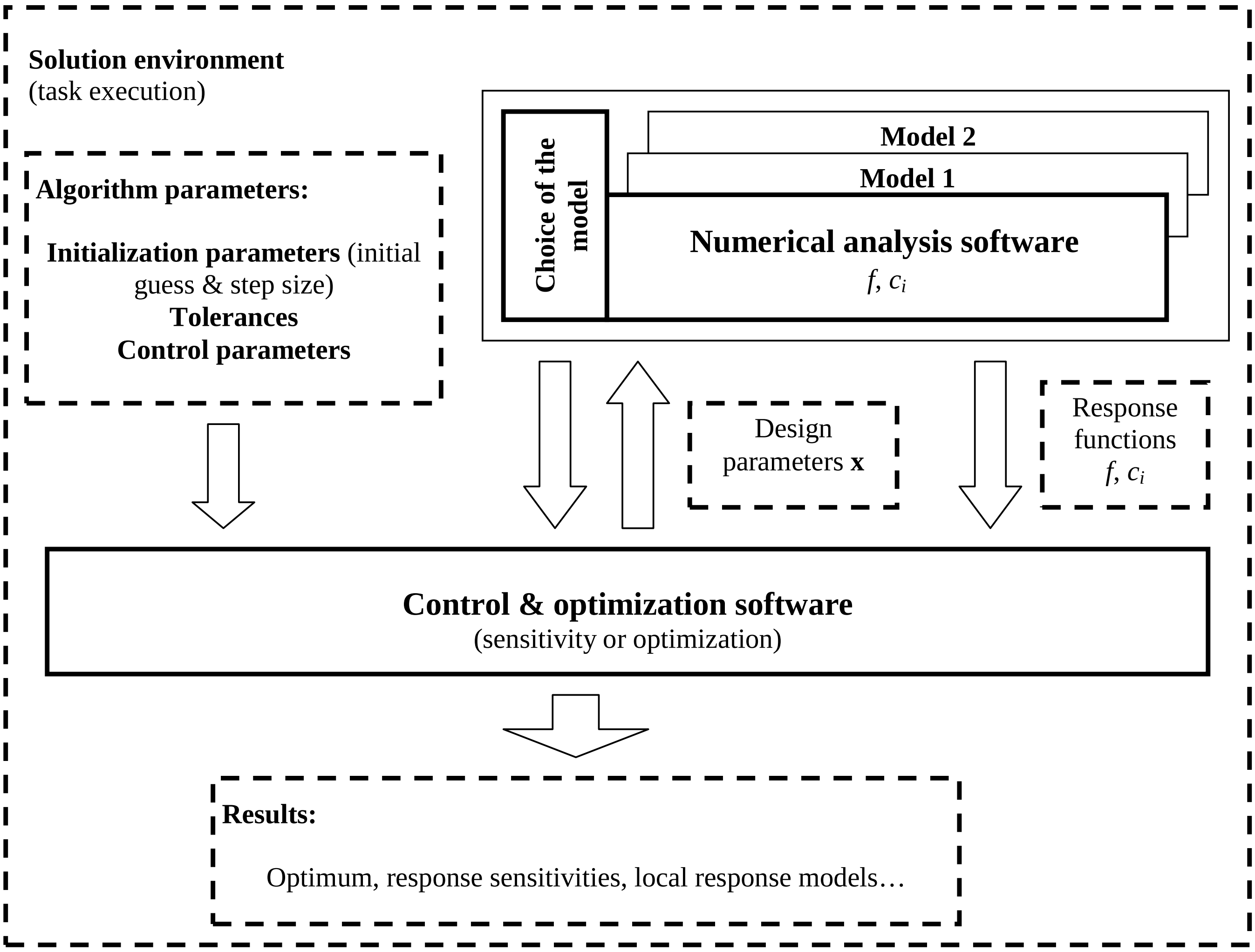

Solution environment
(task execution)
Algorithm parameters:
Initialization parameters (initial guess & step size)
Tolerances
Control parameters
Choice of the model
Model 2
Model 1
Numerical analysis software
f, c_i
Design parameters x
Response functions
f, c_i
Control & optimization software
(sensitivity or optimization)
Results:
Optimum, response sensitivities, local response models…

Figure 3: Solution environment scheme.

minimise	$f(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{R}^n$	(1) a)
subject to	$c_i(\mathbf{x}) \le 0, \ i \in I$	b)
and	$c_j(\mathbf{x}) = 0, \ j \in E$,	c)
where	$l_k \le x_k \le u_k, \ k = 1, 2, ..., n$.	d)

Figure 4: Example: statement of an optimization problem.

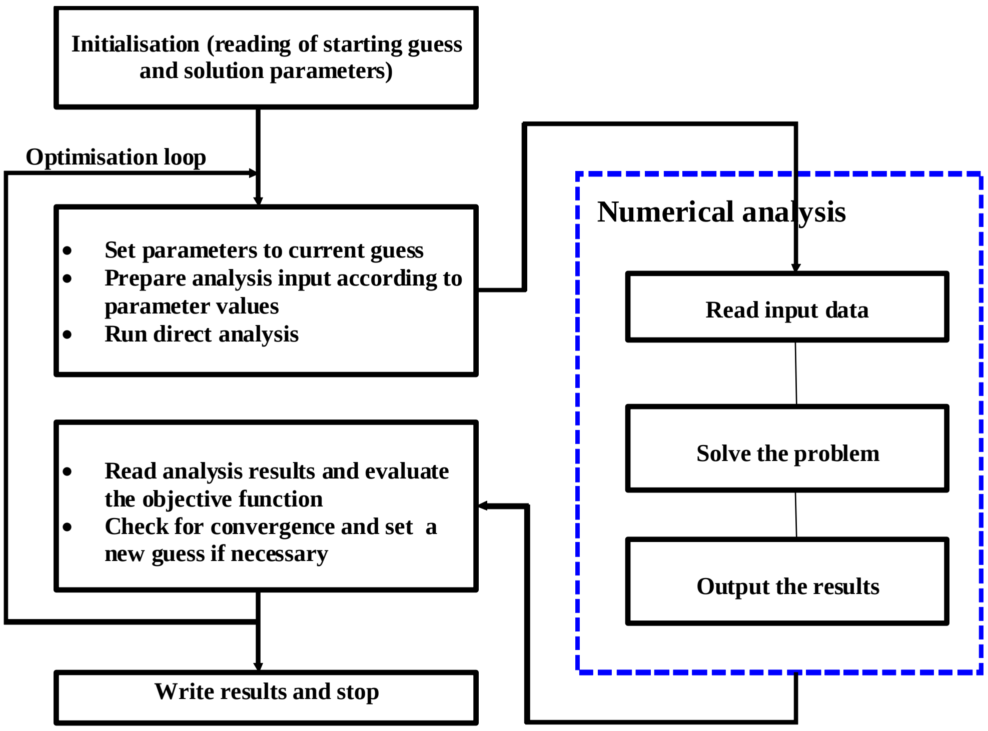

Initialisation (reading of starting guess and solution parameters)
Optimisation loop
Set parameters to current guess
Prepare analysis input according to parameter values
Run direct analysis
Read analysis results and evaluate the objective function
Check for convergence and set a new guess if necessary
Write results and stop
Numerical analysis
Read input data
Solve the problem
Output the results

Figure 5: Solution scheme for optimization problems.

1. Take current optimization parameters
2. Prepare numerical model according to parameters
3. Run numerical simulation of the process
4. Extract the relevant quantities from simulation results
5. From measured data
    - Read result file
    - Extract relevant data
6. Calculate the response functions and eventually their gradients (in our case the discrepancy function *f*)
7. Store the response functions in output arguments and return

Figure 6: Numerical analysis: flow in the case of parameter identification.

4.1.1 Notes on Nomenclature

When talking about optimization with people without extensive background in the field, missunderstandings are very common. Section 4.1 should provide some basic overview necessary for clear communication between optimization, numerical analysis and industrial experts. It is also good to fix some standard expressions which are often used in the context of optimization.

Function $f(\mathbf{x})$ in Equation (1) a) that is minimzed in an optimization problem is called ***objective function***. Vextor $\mathbf{x}$ is a vector of ***optimization parameters***. Optimization problems can be stated in such a way that there are more than one objective functions. In this case we have multiobjective optimization and in general there is not a unique solution to such a problem, but we obtain a whole multidimensional space of solution among which we can choose (which may be impractical, especially when the dimension of solution space is more than 2 or 3). Other names are sometimes used for objective function, such as *merit function*, *cost function* (which may sound more appropriate when optimization problem is stated as minimization rather than maximization problem), *discrepancy function* (especially in the contect of inverse problems where a measure fo discrepancy between experimental measurements and those approximated by a numerical model is minimized).

Functions $c_i(\mathbf{x})$ and $c_j(\mathbf{x})$ are ***constraint functions***. Equations (1) b) through (1) d) (of which first two involve constraint functions) are called ***constraints***. Equations (1) b) are called ***inequality constraints***, equations (1) d) are called ***equality constraints*** and equations (1) e) are called ***bound constraints***. Bound constraints could be stated as normal equality constraints with simple constraint functionbs, but usually they are stated separately because they are easier to deal with for optimization algorithms and because evaluation of the corresponding constraint functions does not require solution of the direct problem.

The set of all points $\mathbf{x}$ in the parameter space that satisfy all constraints is called the *feasible set* or ***feasible region***. Any such point is called a *feasible point*.

The objective and constraint functions (i.e. $f(\mathbf{x})$, $c_i(\mathbf{x})$ and $c_j(\mathbf{x})$) are collectively called ***response functions*** (or simply ***respnse***) of the optimization problem.

The *objective function* is defined in accordance with what one want to achieve when stating and solving the optimization problem (e.g. minimal energy consumption with constraint that time of the considered operation function must remain under certain limit; or minimal discrepancy between results of numerical model and experimental results, a common goal in inverse problems and model calibration). In practical cases there can be more than one goals, often conflicting (e.g. we can also seek for as small consumption of energy as possible and at the same time as short operation time as possible). In such a case, the objective function will usually be defined as weighted sum of terms that measure achievement of individual goals, or more conveniently some nonlinear functions of such terms. For example, industrial problems can usually be stated in terms of a common goal that is overal cost or benefit. If it is known how energy consumption or operation time affect the cost then it is easy to compose an objective function from individual terms. When this is not so obvious in advance, some parts of the goal statement can be moved to constraints, and this can be iteratively varied until the solution obtained is as meaningful for our practical situation as possible.

Constraints can have two distinct purposes. In some cases constraints are a logical part of the definition of the optimization problem and are related to goals we want to achieve. For example, we want to minimize energy consumprtion in a forming process, but don't want plastic deformation (or some complex measure of material damage) to exceed some specified limit anywhere in the formed part. The last goal is most logically stated as constraint. Sometimes goals that could be stated as terms in the objective function are moved to constraints in order to avoid the question of weighting of objectives or multi-objective formulations.

Another purpose for stating constraints is to avoid solutions that violate some physical laws (e.g. material with Poisson’s ratio greather than 0.5) or solutions that are infeasible for some natural reasons (e.g. geometric constraints) or solutions for which numerical calculation of response functions would be unstable. In these cases it is sometimes necessary to ensure that none of the points in parameter space where response functions are evaluated (by the optimization algorithm) are violating certain constraints. Optimization algorithms that are adapded to this requirement are usually called ***feasible methods*** (e.g. ”feasible sequential quadratic programming”).

A single evaluation of all the response functions at the specified value of optimization parameters is called ***direct analysis***. A module that performs such calculation is also often called like that (or more precisely the ***direct analysis module*** or *direct analysis program*). This can be as simpel as a couple of lines of code that use some analytical expressions that define the objective and constraint functions. It can contain evaluation of some global approximation of response measured on a real-life system (e.g. by a neural network). In our case, the *direct analysis* will usually involve a complete numerical simulation of the system in question at the specified values of optimization parameters.

Optimization parameters at which direct analysis is performed form the ***analysis input***. The values of response functions calculated by the direct analysis at specific optimization parameters form ***analysis output***. However, analysis output can in some cases consists not only of values of the response functions, but also of their gradients with respect to optimization parameters. Second derivatives are also provided by the direct analysis in some cases, although this is seldom the case.

When the direct analysis involves a numerical simulation or some other approximation of the response (e.g. by neural networks), input for these components does not directly coincide with optimization parametes (the ”*analysis input*”), as well as output does not coincide with the *analysis output*. In order do utilize a numerical simulation for use in solution of optimization problems, proper mapping must be implemented between the analysis input (or optimization parameters) and input of numerical simulation, as well as between simulation output and the analysis output (i.e. response functions and possibly their gradients). This mapping is called ***parameterization***.

At the input side, parameterization can be as simple as arranging values of optimization parameters to specific places of input file for numerical simulation. This is the case e.g. when optimization parameters represent material properties, which are directly read as simulation input. In other cases parameterization is more complex, e.g. when a portion of optimization parameters defines geometry of object involved in numerical simulation. Such parameters are commonly referred to as ***shape parameters***. In this case, parameterization involves generation of mesh (used in simulation) consistent with *shape parameters*.

At the output side, parameterizatio usually consists of some form of ***post-processing*** of simulation results and calculation of some meaningful output parameters that are arranged in expression that define how response functions are evaluated. For example, if the objective functions contains external work applied to the system of interest, then forces dot multiplied by displacements must be integrated over time and object boundary. Parameterization is therefore usually closely related to the numerical method used in simulation and must therefore be at least paritially performed by the simulation software. In some cases, analysis output also contains gradients of the response functions. In such cases the numerical simulation (or other kind of approximation, e.g. a trained neural network) must be specially adapted to generate this information. Such adaptations of simulation software are commonly categorized as ***sensitivity analysis***. It seems unlikely that we would build sensitivity analysis into our models in a short-term period (e.g. within the next three years).

Optimization algorithms are used to solve optimization problems. The optimization algorithm usually takes some user-specified initial guess and successively changes optimization parameters in an systematic manner and runs direct analysis at these parameters in order to calculate the response. Since the procedure must be automatic for almost all practical purposes, any numerical simulation (or other procedure that used in approximation of response functions, such as a neural network) must be able to be run in non-interactive manner, e.g. as program that is run via command-line and terminates after simulation is finished.

The terms *direct analysis*, *direct analysis input* and *direct analysis output* are usedsomehow ambiguously when describing software architecture. In the strictest sense, the term *direct analysis* is used for a function that is called by the optimization algorithm to perform evaluation of optimization respnse at some specific point in the space of optimization parameters. This is more strctly called the ***direct analysis function***. Different algorithms in different optimization libraries usually require different forms of direct analysis functions (with different signatures, i.e. number and types of arguments). Some algorithms, for example, call separate functions for evaluation of each individual component of the response (i.e. the objective function, individual constraint functions, and eventually the gradients thereoff). In our optimization environment we will define one or two standard forms of *analysis function*, and will implement adapters (wrappers) for algorithms that in their original form require different analysis functions. In this way we will be able to combine any definition of the direct analysis with any built-in algorithm suitable for the particular problem.

In a less strict way, the term *direct analysis* is also used for a stand-alone computer program that is able to read analysis input form files, map it to input for numerical analysis, run numerical simulation (or other kind of response approximation), post-process results, and write analysis output to a file. We will define standard analysis input and output file formats for this purpose, and implement *direct analysis functions* within the optimization system that will wrap this kind of analysis functions.

Definition fo the optimization problem refers to the definition of how to calculate the response functions. This is essentially the definition of the direct analysis.

Optimzation environment is a software environment that is used to define optimization problems and to run optimization algorithms that calculate numerical solutions to these problems. Typically, optimization environment enables combination of different definitions of the direct analysis with different solution algorithms.

4.2 Data Exchange Between Optimization and Direct Analysis

Analysis request (analysis input file):

```
{ { p1, p2, … }, { reqcalcobj, reqcalcconstr, reqcalcgradobj,
            reqcalcgradconstr }, cd }
```

Legend:

p1, *p1*, *p3* – optimization parameters at which analysis was performed
Flags that tell whether something has actually been calculated (0 – yes, 1- no):

- *reqcalcobj* – flag for the objective function
- *reqcalcconstr* – flag for constraint functions
- *reqcalcgradobj* – gradient of the objective function
- *reqcalcgradconstr* – gradients of constraint functions

cd – a free parameter that can be used to transfer additional information to the direct analysis. In principle *cd* can be anything embedded in curly brackets (*{..}*) If only the eventual embedded curly brackets are properly closed. Most commonly it will not be used at all and therefore empty brackets ("*{}*") will be put in place of *cd*. Otherwise, interpretation of what stands in curly bracket is entirely in the domain of the analysis program, therefore the documentation of the analysis program should provide information on how to compose *cd*.

Analysis results (analysis output file):

```
{
  { p1, p2 ... },
  {
    calcobj, obj,
    calcconstr, { constr1, constr2, ... },
    calcgradobj, { dobjdp1, dobjdp2, ... },
    calcgradconstr,
    {
      { dconstr1dp1, dconstr1dp2, ... },
      { dconstr2dp1, dconstr2dp2, ... },
      ...
    },
    errorcode
  },
  { reqcalcobj, reqcalcconstr, reqcalcgradobj, reqcalcgradconstr }
  < , { ind1, ind2, ... }, { coef1, coef2, ... }, defdata >
}
```

Legend:

- *calcobj* – flag for the objective function
- *calcconstr* – flag for constraint functions
- *calcgradobj* – gradient of the objective function
- *calcgradconstr* – gradients of constraint functions

obj – value of the objective functions
constr1, *constr2*, … - values of the constraint functions
dobjdp1, *dobjdp2*, ... – derivatives of the objective function with respect to individual parameters (components of the objective function gradient)

dconstr1dp1, …, *dconstr2dp1*, *dconstr2dp2* – derivatives of individual constraint functions with respect to individual optimization parameters – components of gradients of the constraint functions (e.g. *dconstr2dp3* is the derivative of the second constraint function with respect to the third parameter)

errorcode – integer error code of analysis – 0 for no error, usually a negative number for errors, values are function specific

reqcalcob , *reqcalcconstr*, *reqcalcgradobj* and *reqcalcgradconstr* are request flags for calculation of the various values, as have been passed to the analysis function. The same as with parameter values, these flags are requested only for verification. In vast majority of cases these flags will not be used by the optimization program, and they can simply be set to 1.

Analysis results (analysis output file) for multi-objective case:

```
{
  { p1, p2 ... },
  {
    calcobj, {obj1, obj2, ... },
    calcconstr, { constr1, constr2, ... },
    calcgradobj,
    {
      { dobj1dp1, dobj1dp2, ... },
      { dobj2dp1, dobj2dp2, ... },
      ...
    },
    calcgradconstr,
    {
      { dconstr1dp1, dconstr1dp2, ... },
      { dconstr2dp1, dconstr2dp2, ... },
      ...
    },
    errorcode
  },
  { reqcalcobj, reqcalcconstr, reqcalcgradobj, reqcalcgradconstr }
  < , { ind1, ind2, ... }, { coef1, coef2, ... }, defdata >
}
```

Examples of analysis ouptut files:

```
{ {1.11, 2.22}, { 1, 6.1605, 1, {-0.165, -2.44} , 1, {2.22, 4.44}, 1, { {-
            1.5, 0.}, {0., -2.} }, 0 }, { 1, 1, 1, 1}, {}, {}, "3" } }

{ {1.11, 2.22}, { 1, 6.1605, 1, {-0.165, -2.44} , 0, { }, 0, {  }, -1 }, { 1,
            1, 1, 1}, {33, 45}, {2.5, 3.33 38.1}, "3" } }
```

Alternative format: XML (analysis output):

```
<!-- Analysis output file, created by analysis wrapper. -->
<data type="analysispoint" mode="analysis_output" ind="1">
  <ret type="counter">0</ret>
  <reqcalcobj type="counter">1</reqcalcobj>
  <reqcalcconstr type="counter">1</reqcalcconstr>
  <reqcalcgradobj type="counter">1</reqcalcgradobj>
  <reqcalcgradconstr type="counter">1</reqcalcgradconstr>
  <calcobj type="counter">1</calcobj>
```

```
    <calcconstr type="counter">1</calcconstr>
    <calcgradobj type="counter">1</calcgradobj>
    <calcgradconstr type="counter">1</calcgradconstr>
    <param type="vector" dim="2">
      <vector_el type="scalar" ind="1">1.6</vector_el>
      <vector_el type="scalar" ind="2">1</vector_el>
    </param>
    <obj type="scalar">0.20088905308774715</obj>
    <constr type="table" eltype="scalar" dim="2">
      <table_el type="scalar" ind="1">0.0</table_el>
      <table_el type="scalar" ind="2">0.0</table_el>
    </constr>
    <gradobj type="vector" dim="2">
      <vector_el type="scalar" ind="1">0.24138</vector_el>
      <vector_el type="scalar" ind="2">0.0172418</vector_el>
    </gradobj>
    <gradconstr type="table" eltype="vector" dim="2">
      <table_el type="vector" dim="2" ind="1">
        <vector_el type="scalar" ind="1">-1.1</vector_el>
        <vector_el type="scalar" ind="2">2.1</vector_el>
      </table_el>
      <table_el type="vector" dim="2" ind="2">
        <vector_el type="scalar" ind="1">0</vector_el>
        <vector_el type="scalar" ind="2">-1</vector_el>
      </table_el>
    </gradconstr>
    <!-- Optional definition data: -->
    <cd type="string">Definition data</cd>
  </data>
```

Figure 7: Examples of data exchange file formats.

4.3 Optimization Shell – Things to be Done First

These things should be tone first, within say a one year period, dependent on other activities:

- **I/O Toolbox**
  - Parser
  - Reading/writing analysis data
  - Reading optimization data for standard algorithms
- **Analysis file client and server**
  - Standard exchange files and formats
  - Basic components, e.g. synchronization with file system
  - Integration with internal components
- **Optimization file client and server**
- **Standardization of internal components**
  - Interfaces for analysis functions, result storage, optimization, etc.

- **Toolbox for response inspection**
    - Derivatives, smoothnes, optimality conditions, etc.
    - Response surrogate techniques
- **Algorithm kit**
    - Gradient based
    - Robust
    - ...?
- **Basic graphics**
- **Rough application outline**

5 USE OF IGLIB AS BASE LIBRARY

5.1 About IGLib.NET

Kaj je IGLib

Pred nekaj leti sem se odločil, da bom na novo zgradil framework za optimizacijo. Ta naj bi med drugim nadomestil tudi večino funkcionalnosti sistema *Inverse* [8], vendar bi bil zgrajen na drugačnih, bolj sodobnih konceptih z uporabo izkušenj pridobljenih v letih dela na optimizaciji, numeričnih simulacijah in tehničnem softveru. Ko je bil narejen koncept *Inverse*-a, je bila tehnologija na tem področju na čisto drugem nivoju. Zaradi okoliščin se mi je zdelo bolj smotrno, kot prilagajati obstoječi framework, postaviti novega od začetka.
Še bolj kot pri *Inverse*-u sem se želel lotiti zadeve sistematično in pri tem uporabiti dolgoletne izkušje na različnih področjih. *IGLib* ([5]-[7]) sem poimenoval osnovno knjižnico, na kateri bi bilo zgrajeno ogrodje. Knjižnice si nisem zamislil le kot podlago za novi framework, ampak bolj splošno kot podlago za sistematičen razvoj tehničnih aplikacij. Njen razvoj sem vezal na naslednja načela:

- IGLib bo prosta in odprta knjižnica brez omejitev za uporabo. Na ta način ne bo nobenih ovir za pridobivanje partnerjev za razvoj, za mednarodno in interdisciplinarno sodelovanje in za komercializacijo morebitih produktov razvitih na osnovi knjižnice. Če bo kdaj pozneje v razvoj knjižnice vključenih več partnerskih ustanov, bodo od takšnega sodelovanja vse imele korist, ker jim bo takoj dostopno, kar so v okviru knjižnice razvili drugi.

- Moja začetna motivacija za razvoj knjižnice je uporaba le te na svojih projektih. Energija, ki jo vlagam v sistematično načrtovanje arhitekture, se mi povrne pri bolj efektivnem razvoju končnih produktov.

- Knjižnica bo odvisna od številnih drugih knjižnic za različne stvari (npr. linearno algebro, procesiranje signalov, risanje grafov itd.), pogoj za vse takšne zunanje knjižnice, ki so njen del, pa je kompatibilnost licenc. Če bo pri razvoju neke aplikacije prav prišla kakšna knjižnica, katere licenca ni kompatibilna z IGLib (in bi zaradi tega prišlo do omejitev glede uporave knjižnice), se vsi deli odvisni od te knjižnice zapakirajo v drugo knjižnico, ki pa je seveda lahko odvisna tudi od IGLib.

- Knjižnica temelji na ogrodju .NET in je napisana v jeziku C#. Ker je .NET lastniško okolje Microsofta, je dolgoročno namen ohranjati kompatibilnost z ogrodjem Mono, ki je prosta odprtokodna (zaenkrat delna) implementacija .NET-a. S tem bo zagotovljena prenosljivost tudi na druga sisteme, npr. Linux.
- Velik poudarek je na sistematičnem razvoju in dobrem planiranju knjižnice. V zvezi s tem je postavljenih neakaj načelnih pravil.
    - Ko se pojavi potreba po novem orodju ali funkcionalnosti, ki je dovolj splošne narave, da bi spadala v knjižnico, se pri implementaciji najprej razmisli, kako bi bilo to narejeno v okviru širšega modula knjižnice, ki pokriva področje, kamor to spada. Potem se navadno implementira samo specifičen del, ampak na način, da je to razširljivo v konsistenten splošen modul z dobro zastavljeno arhitekturo
    - Kadar preveč sistematično vključevanje funkcionalnosti ni smotrno (recimo, kadar bi to zahtevalo preveč energije ali bi odvračalo pozornost od treutnega cilja), se pri implementaciji uporabi kompromis in se označi, da gre za del kode, ki še ni zrel za vključitev v knjižnico. V knjižnici bo več ločenih nivojev in koda bo prehajala od najbolj osnovnega (grobi osnutek za testiranje konceptov) v zreli del, kjer se bo težilo k čimvečji stalnosti.
    - V zrelem delu knjižice so postavljeni kriteriji za dokumentacijo kode, izgradnjo testnih primerov za testiranje funkcionalnosti in druga pravila za zagotavljanje kvalitete kode.
    - Spodbuja se kolaborativni pristop, kjer imajo vsi vpleteni korist od sodelovanja.
    - Spodbuja se mnogoterost idej. Če ima več ljudi različno vizijo o tem, kako bi implementirali isto funkcionalnost, lahko vsak naredi svoj modul in ga vključi v knjižico. Sčasoma se bo pokazalo, katera ideja bo pridobila več uporabnikov in razvijalcev.
- V določenem obdobju bom imel sam kontrolo nad načrtovanjem knjižnice, dokler ne bo knjižnica dovolj razvita, da bo možno vodenje razvoja tudi v širših ovirih.

Nekaj primerov stvari, ki so že vključene v knjižnico ali so v razvoju: osnovna linearna algebra (brez razpršenih matrik), vmesniki za definicijo funkcij, osnovni moduli za razvoj optimizacijskih algoritmov, 2D diagrami, celovit sistem za javljanje napak in ostala sporočila, interaktivni ukazni kalkulator, interpreterski moduli, ki omogočajo uporabniku sestavo operacij, osnova za izgradnjo vmesnikov preko datotečnega sistema in podobno.

Ker imam dobro izdelano sliko o tem, kaj bi rad od optimizacijskega ogrodja, imam zaenkrat to za rdečo nit razvoja, hkrati pa imam pri načrtovanju vedno v mislih tudi šuiršo uporabnost.

Kako bi vključil IGLib v tvoje okolje in kakšne so prednosti

Knjižnico bi vključil na podoben način, kot jo vključujem v ostale svoje projekte, torej kot bazično knjižnico, kjer se na urejen in načrten način akumulira funkcionalnost splošne narave, ki je potrebna pri sprotnem razvoju končnih produktov. Tisti del, ki je knjižnica, odtane odprt in obdrži zgoraj navedene lastnosti. Stvari, ki so končni produkti ali vsebujejo algoritme in druge stvari, ki so pomembne za skupino, bodo zapakirane v samostojne module in aplikacije, ki ne bodo prosto dostopne.
Prednosti uporabe knjižnice so različne. Meni in pozneje tudi ostalim bo olajšala delo, ker vsebuje veliko že narejenih stvari in se bo nabor teh stvari širil na sistematičen način. Knjižnica ima homogeno zgradbo in temelji na enotnih in dodelanih konceptih, ki se bodo še dopolnjevali.

Koncept knjižnice dolgoročno omogoča motiviranje strokovnjakov, da se pridružijo uporabi in razvoju knjižnice. To prinese vzajemno korist vsem, ki uporabljajo knjižnico, ker se poveča razvojni potencial, več uporabnikov odkrije tudi več napak, prispevanje k razvoju odprtih knjižnic pa je tudi dobra referenca za tiste, ki prispevajo. Organizacija razvoja okrog takšne knjižnice predstalvja tudi dobro osnovo za razvoj timskega dela v razvojni skupini, omogoči lažjo povezljivost rezultatov zaradi dodatnega nivoja standardizacije, ki se spontano uporablja in razvija, ter prispeva k hitrejšemu napredovanju razvojnega potenciala članov skupine.

Če se kdaj pozneje odloči, da bi bilo bolj smotrno knjižnico zapreti, se to vedno lahko naredi tako, da se obdrži trenutno stanje knjižnice, ki se potem razvija naprej ločeno od originala (ki bo ostala odprta knjižica) in se dajo vse spremembe pod drugo licenco. Licenca knjižnice namreč ne postavlja kakšnih omejitev glede komercialne uporabe ali predelave ali licenciranja izpeljanih produktov.

Povezava z mojim delom na splošno

Kot sva se že pogovarjala, si bom pri usklajevanju razvoja softvera prizadeval za čimvečje poenotenje razvoja in tesno sodelovanje članov skupine pri tem. To pa ne pomeni, da vidim dolgoročno IGLib kot osnovo za ves softverski razvoj. Možno je, da bodo tudi dolgoročno določeni deli razvoja softvera osnovani na drugih platformah kot IGLib, npr. lahko se izkaže, da bo simulacijski del najbolje osnovati na nativnem C++. V splošnem ni zelo narobe, če se kot osnova za razvoj uporabljajo dve ali tri platforme, če za to obstajajo tehtni razlogi. Takšne odločitve so dolgoročne in nekaterih ne bo možno sprejeti takoj. Bi pa na .NET in IGLib zasnoval razvoj povezovalne platforme, na kateri bomo integrirali razvite produkte, to bo vključevalo tudi optimizacijsko lupino. V začetni fazi bo povezovanje različnih že narejenih delov sistema potekalo preko vmesnikov med njimi, ki bodo v glavnem temeljili na izmenjavi podatkov preko datotečnega sistema in sistemskih ukazov za poganjanje aplikacij. Tak način povezovanja ni vedno najbolj optimalen, kar se hitrosti tiče, je pa najbolj pregleden in obvladljiv, kar bo v našem primeru prevladujočega pomena.

Spodaj navajam še vsebino licence za knjižnico. Ta se bo morda v prihosnosti še dopolnjevala, cilj pa je ohraniti vsebino zelo kratko in razumljivo.

==== Vsebina licence za knjižnico:

Opombe:
Načrtoval sem, da bo IGLib nekoč pod odprtokodno licenco (glej spodaj pod possible future license), vendar moram najprej zagotoviti ustrezno stanje knjižnice in zadostno kontrolo nad njenim razvojem. To med drugm pomeni zadosten obseg knjižnice in zadosten razvojni potencial okrog knjižnice, ki bo pod mojim vodstvom, da bom lahko zagotovil, da bo razvoj potekal v pravo smer (to med drugim pomeni, da bo knjižnica na dolgi rok ustrezala svojemu namenu in bo dobro služila interesentom, med katerimi bi bil tudi Laboratorij za večfazne procese). Do takrat bom obdržal malo bolj restriktivno licenco za knjižnico, ki bo hkrati omogočila potrebno razpolaganje interesentom in zagotavljala, da ohranim kontrolo nad razvojem knjižnice. Spodaj je predlog te licence.
Za točen tekst licence glej navodila in dokumentacijo ([6], [7]).

5.2 **IGLib** ***License Agreement***

This is a license agreement for the *IGLib* utility library ("*the software*") and its documentation, which are owned and copyrighted by Igor Grešovnik, Jamova 80, Ljubljana. The software subject to this license agreement includes all files that are contained in the software directory (i.e. the root directory where this license file is located, and all its subdirectories).

The software and its documentation are developed and copyrighted by *Igor Grešovnik*, Ljubljana, Slovenia ("*the author*"), except for individual parts of the software for which separate different copyright notices are provided. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The author hereby grants the limited rights to use the software to the following licensees ("*the licensees*"):

- Group of Professor Božidar Šarler within the Laboratory for Multiphase Processes of the University of Nova Gorica; responsible person: Božidar Šarler.
- Group of Professor Božidar Šarler within the Laboratory for Supervisory Systems of the Centre of excellence for Biosensors, Instrumentation and Process Control; responsible person: Božidar Šarler.

The present license agreement is the agreement between the author and the licensees that defines the terms and conditions for use of the software. By using the software, licensees accept the terms of this license agreement. The responsible persons stated above shall be responsible for execution of the terms of this license agreement by the individual licensees.

5.2.1 Grant of Rights

The author hereby grants, and licensees hereby accept, subject to the terms and conditions of this Agreement, a nonexclusive, nontransferable and nonassignable license to use the software in in order to create Derivative Products.

Licensees can use, license, sell, and distribute their products derived from the software without any limitations, except that the source code of the software may only be used by licensees and may not be shipped together with derived products or distributed by licensees in any other way, unless a prior written consent is provided by the author. Licensees may not give to third parties any technical details or documentation of the software, unless a prior written consent is provided by the author.

Licensees are obliged to retain this license agreement and all copyright notices in all copies of the software. In any derived products, licencees shall acknowledge use of the software with a notice that is easily accessible to the users of these derived products.

The author and the licensees hereby agree that they will jointly develop the software with the purpose of its improvement and extension in order to fit their needs. Such development will be performed under guidance and with consent of the author. All modifications of the software will be copyrighted by the author and will be subjected to the terms of the present license agreement.

5.2.2 Statement of Intention and Obligations

The intention and common interest of the author and the licensees is to develop and continuously improve a good base library for development of their applications, and will jointly pursue after efficient and high level development work in order to produce good and useful software from which all of them will benefit. In long term, the author intends to broaden the circle of developers and users of the software and may eventually release the software under a free open source license in order to attract a broader community of collaborative developers and users. The intention of the author and licensees is to maintain longer term collaboration on the development and use of the software.

Within the period in which licensees and the author will jointly work on the library, any contributors will be respoinsible for maintaining integrity and good quality of the library. They will refrain from any actions that might harm the usability, quality or good reputation of the library.

Within the period in which the software is used as base library for development of derived product at licensees' institutions, main contributors to the libraries derived from the software will be granted similar rights as stated in the current license agreement. This means that the main contributors will be able to use the developed libraries to which they contribute over this period, under similar terms as stated in this license agreement, to derive their own products from these libraries. In particulat, they will be allowed to use, license, sell, and distribute such derived products without any limitations, except that the source code of the derived products may not be distributed. However, this right will be granted only for libraries and applications that do not contain any trade secrets or vital knowhow that is used for commercial purposes (and which the involved institutions - the licensees - do not want to reveal publically). Such non-disclosable contents will be separated from basic technical libraries and put into specialized units (applications and high-lever libraries). Contributors who are granted rights from the current paragraph will be selected by the author of the software.

In addition to the rights stated in the previous paragraph, the author of the software, Igor Grešovnik, will retain the right to spawn his own continuous development thread for any of the derived libraries mentioned in the previous paragraph, and to use, develop and copyright such a newly created library without any limitations. In the case that such a fork event occurs, the author must assign a new name to his forked version of the library, and may only include in this version the code of the original library that was created before the fork event occurred, unless agreed otherwise by the copyright holder of the original library.

Licensees will pursue the goal that the products derived from the software are as open and as widely disseminated as possible, especially when creation of such derived products is partially or fully supported by public funding.

5.2.3 Disclaimer

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE AND ITS DOCUMENTATION ARE PROVIDED ON AN „AS IS“ BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

5.3 Possible future License Agreement for IGLib

This software and its documentation are developed and copyrighted by *Igor Grešovnik*, Ljubljana, Slovenia, except for individual parts of the software for which separate different copyright notices are provided. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE AND ITS DOCUMENTATION ARE PROVIDED ON AN „AS IS“ BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

5.4 IGLib README

==== Content of the file Readme.html:

IGLib.NET (Investigative Generic Library) is a set of utility libraries that are particularly suited for development of technical applications.

The system has been designed and developed by *Igor Grešovnik*, who set up its foundations in 2006 and is leading its development. In longer term, the library is intended for distribution as free open source under a BSD-like license. However, the library will not be open for public immediately because the author wants to achieve a certain level of maturity first and stabilize library development within a smaller group of dedicated developers. Hopefully the library will be released as free open source library by the end of 2013. Before this happens, groups and individuals can join development (and usage) of the library by individual agreement with the principal author. The author is open to such arrangements but would like to retain a good control over development until a certain level of maturity is reached. He believes that such position will be beneficial for future users and developers.

Since 2011, the library is used by the Laboratory for Multiphase Processes of the University of Nova Gorica, and by the Laboratory for Advanced Materials Systems of the Centre of Excellence for Biosensors, Instrumentation and Process Control, where it is used as base library for development of applications in the field of neural networks and optimization. These grous use the code under a customized license agreement and also contribute to library development in a limited extent.

IGLib contains some basic utilities like those for parsing of strings, a couple of utilities for building GUI, a numerical library, and other components. It aims at providing a well designed base library for developnent of complex numerical and other technical applications. Parts of IGLib have also been used in other areas such as a system for managing a histological laboratory or large scale invoicing support system.

Historically, the initial motivation for development of the library arose from the needs to have a good base library for development of complex optimization software, but the library was planned in a much broader sense since the very beginning of its existence. For more information, check the library home page at

http://www2.arnes.si/~ljc3m2/igor/iglib/,

or check code documentation at

http://dl.dropbox.com/u/12702901/code_documentation/generated/iglib/html/index.html.

External Libraries

This library depends on a number of external free open source libraries. Authors of the code are grateful to all developers that invested their work to develop these libraries and who made them open and accessible to the public.

The following external libraries are used:

- Math.Net, an excellent scientific library written entirely in C#. Iridium and Neodym libraries are used from this project.
- ZedGraph, a flexible charting library for .NET.
- NPlot, an easy to use 2D plotting library.
- Activiz, C# wrappers for the VTK 3D graphics library.

Please visit the home pages of these great libraries (just follow the links above) and consider whether you can support their development in some way.

Authors' index

Below is the list of authors' synonims used in the code:

- Igor - Igor Grešovnik, Črneče 147, Ljubljana, Slovenia (gresovnik (at) gmail (dot) com)
- Tako78 - Tadej Kodelja, Slovenia (tadej (dot) kodelja (at) gmail.com)
- Vertnik - Robert Vertnik, Slovenia (robert (dot) vertnik (at) gmail (dot) com)
- Katarina - Katarina Mramor, Slovenia (kmramor (at) gmail (dot) com)

6 GENERAL MANAGEMENT OF SIMULATION FRAMEWORK - SUGGESTIONS

6.1.1.1 From Mail to Božidar & Robert, Nov 26 2010

Spodaj sem poskusil zgoščeno in okvirno predstaviti zamisel o tem, kako uredimo razvoj, pravice in dostop do razvitega simulacijskega softvera. O teh stvareh bi morali v doglednem času sprejeti neke osnovne odločitve, bomo pa verjetno rabili nekaj časa, da izmenjamo in uskladimo poglede.

Softver bo logično sestavljen iz več nivojev: osnovne knjižnice, višjenivojske knjižnice, simulacijsko ogrodje, splošne simulacijske aplikacije in kustomizirane industrijske aplikacije.

Precej časa bodo verjetno vse, kar bomo naredili za industrijo, kustomizirane industrijske aplikacije. Lahko, da bomo kdaj tudi prodajali splošne licence (podobno, kot so npr. licence za Fluent), vendar si v naslednjih treh letih težko predstavljam to možnost.

Kar se tiče kustomiziranih industrijskih aplikacij, bodo to zaprte kode. V vsakem primeru posebej se bomo morli z naročnikom dogvoriti, ali hoče imeti ekskluzivo glede uporabe in na katerih delih softvera. V večini primerov naročniku ne bomo prodali softvera (to bi pomenilo, da izgubimo vse pravice na tem softveru), ampak licence za uporabo.

Tudi pri kustomiziranih aplikacijah bo lahko samo del kode tak, ki bo resnično vezan na specifično aplikacijo in za katerega lahko naročnik zahteva ekskluzivne pravice. Če v kakšnem primeru ne bo tako, bo softver seveda bistveno dražji za naročnika, v tem trenutku nimamo niti kapacitet, da bi lahko kaj takšnega naredili. Da bomo lahko ostajali na nivoju, je za nas nujno, da imamo velik del softvera, ki ne pade v kakšne ekskluzivne pogodbe.

Dele kustomiziranih aplikacij, ki jih bomo razvili za kakšno konkretno naročilo, bomo v večini primerov tako ali tako morali zapreti. V našem interesu je vedno, da čim manjši del softvera pade v to kategorijo, da bomo lahko iste stvari uporabili čim večkrat. Najbolj idealna situacija pri industrijskih naročilih je, če moramo zapreti samo mali del softvera, ki se res tiče le zelo specifičnih stvari za dano naročilo (npr. konkretnih strojev ali procesov, ki jih simuliramo).

Pri osnovnih knjižnicah in tudi pri osnovnem simulacijskem ogrodju bi na vsak način poskusil ohraniti čimvečjo odprtost. To nam bo med drugim omogočilo tudi uporabo veliko odprtokodnega softvera, ki je že narejen in s katerim rešimo del svojih problemov. S takšno odprtostjo lahko veliko pridobimo pri prepoznavnosti naše skupine, predvsem v akademskem okolju. Na ta način lahko tudi motiviramo druge, da uporabljajo naš softver in se morda tudi priključijo razvoju. Že samo uporaba softvera v čim širšem obsegu je koristna, ker bomo tako dobili povratne informacije o tem, kaj ne dela v redu in kaj bi lahko bilo bolje zastavljeno.

Pri koristnem povezovanju navzven vidim dva možna načina, kako se to lahko zgodi. Prvi način je, da ljudje, ki bodo od nas šli delat v drugo okolje in bodo navajeni na uporabo našega softvera (ga bodo tudi razvijali), prenesejo ta softver v svoje novo okolje. Potem se lahko dogovorimo, da prispevajo k našemu razvoju, ali pa da ga samo uporabljajo in nam sofinancirajo razvoj. Če bo vse OK, bomo mi imeli dovolj močan razvojni potencial, da jim bo to bolj v interesu, kot pa začeti novo vejo razvoja in sami razvijati softver naprej. Za primere, ko bi vseeno hoteli začeti svojo vejo, se moramo dogovoriti, kateri del softvera lahko za to uporabijo.

Drugi način povezovanja je, da nekdo drug pride do nas z interesov, da bi uporabil naš softver kot osnovo za svoje stvari. Ko bomo enkrat imeli dobro osnovo in če bomo odprli del softvera, se bo verjetno našel tudi kdo, ki bo zainteresiran za kaj takšnega.

V vsakem primeru se mi zdi koristno prizadevati si, da še drugi uporabljajo naš softver, ker s tem pridobivamo ugled in reference ter dokazujemo, da je naš softver kvaliteten (s tem je podobno kot s citati pri člankih). Tako pridobimo tudi koristne povratne informacije in dodatno kontrolo kakovosti (če je več oči, ki gleda softver, se najde in odpravi tudi več pomanjkljivosti), tudi v primeru, da drugi softver samo uporabljajo.

Pri odprtih kodah se navadno vzpostavi več interesnih skupin ljudi: takšni, ki dejansko prispevajo tudi pri razvoju, aktivni uporabniki, ki ti dajejo zelo koristne povratne informacije, in pasivni uporabniki, ki samo uporabijo softver. Navadno je zelo tažko in v večini primerov neefektivno ločiti med temi skupinami in npr. dati dostop do kode samo prvi skupini, ker potem to ni odprta koda in izgubi svojo funkcijo pri privabljanju potencialov, pa tudi marketinška funkcija takšne kode zbledi. V glavnem imamo v praksi dve možnosti - ali kodo čisto odpremo, ali pa jo zapremo in se dogovarjamo z zunanjimi skupinami za skupni razvoj na podlagi bilaterarnih pogodb, kjer so (navadno precej komplicirano) določene vzajemne obveznosti in pravice. Govorim seveda o delu kode, ki pa mora biti funkcionalno zaključena celota. Po mojem mnenju je odločitev za odprto kodo v našem primeru zdaleč najboljša, ker bomo vsaj delno še vedno delovali v akademskem okolju in bomo na ta način lahko izrabili veliko priložnosti, ki se v tem okolju ponujajo (pridobivali bomo ugled in še prišli do zastonj razvojnih kapacitet). Da del kode odpremo, da mi zdi tudi z moralnega vidika korektno, saj bomo razvoj v veliki meri pokrivali iz javnih sredstev. Jaz bi tudi ljudem, ki bodo delali na razvoju kode, dal pravico, da svoj del kode npr. po končanem doktoratu vzamejo in z njim prosto razpolagajo, ker bo to zelo dobra motivacija za to, da vlagajo svoj trud v razvoj kode. To jim seveda nič ne bo koristilo, če bodo lahko vzeli samo tisto, kar bodo sami napisali, ampak mora biti sem vključena funkcionalno zaključena celota, da bodo lahko svoje stvari dejansko tudi uporabili.

Osebno vidim le dva argumenta proti odprtosti kode: da lahko pride do situacije, ko bomo težko vzdrževali nadzor nad razvojem, in da lahko nekdo poceni pride do tega, kar smo s trudom razvili, in ali postane naša konkurenca ali pa zaradi tega ne bi od nas kupil storitev, ki jih sicer bi.

Zaradi nadzora bi izvedel odpiranje kode postopma. Licence in te stvari lahko uredimo takoj, praktičen dostop do odprtega dela kode (upload na strežnike itd.) pa lahko uredimo pozneje, ko bo koda v dovolj zrelem stanju in bomo imeli dovolj razvojnih potencialov. Vedno imamo mi možnost voditi razvoj (tudi, če izdamo kodo pod odprto licenco), težave lahko imamo samo, če bi bil prevelik naval ljud, ki bi se hoteli v to vključiti. Tudi za ugled ni dobro, če delamo reklamo za kodo, ki je še zelo nedodelana.

Probleme s konkurenčnostjo bi reševal na ta način, da ne damo vsega, kar sodi v osnovno simulacijsko kodo, takoj v odprti del, ampak pri najbolj vrhunskih stvareh (ki lahko za nas pomenijo pomembno konkurenčno prednost in ki še niso lahko dostopne drugje) to naredimo s časovnim zamikom. Na začetku bi takšne module dali v zaprti del kode in se seveda hkrati pohvalili, da imamo za ta in ta problem odličen algoritem, ki ga lahko stranke dobijo preko komercialnega naročila (ravno za takšno oglaševanje nam lahko v prihodnosti odlično služi portal, preko katerega bomo distribuirali odprto kodo). Tu bomo morali iskati dober kompromis med tem, da je odprti del kode vseeno dovolj funkcionalen, profesionalno narejen in stabilen, in med tem, da imamo v zaprtem delu dovolj adutov, da nam drugi težko konkurirajo. V vsakem primeru bo naš največji adut, če uspemo narediti dober tim in organizirati delo, kot je treba, ker nas bo v tem primeru vsak težko dohajal.

Celotna slika bi bila po mojem predlogu takšna:

Imamo dobro dizajnirano odprtokodno ogrodje z vzpostavljenim odličnim notranjim razvojnim potencialom. Vse doktorate vprežemo v to, da to ogrodje izboljšujemo in učinkovito akumuliramo orodja potrebna za dobro simulacijsko kodo (grafika, definicija vhodnih podatkov, solverji in druge numerične knjižnice, definicija kompleksnih geometrij, dober remeshing, adaptivni inkrementalni algoritmi, različni numerični triki itd.). Bolj ko bo to okolje dodelano, lažje bomo pritegnili dodatne razvojne potenciale (doktorandi iz tujine in postdoktorandi, ki si bodo želeli delati v naši skupini, ter zunanje raziskovalne skupine, ki se bodo želele vključiti v razvoj) in večji bo naš ugled, ki bo koristen pri pridobivanju poslov.

Poleg odprtega okolja imamo plast modulov in algoritmov, ki po vsebini sicer spadajo v osnovni nivo, vendar so zelo zmogljivi glede na trenutno stanje razvoja tako da niso zlahka dostopni na trgu. Ta del je zaprt, v večini primerov za dovolj dolgo omejeno obdobje (dokler to ne postane nekaj običajnega in splošno dostopnega). Te algoritme vključujemo v nekatere aplikacije po naročilu in jih po možnosti delimo s partnerskimi skupinami, ki v zameno ponudijo kaj drugega.

Nad tem imamo plast orodij, ki so bolj vezana na industrijske primere in jih nočemo mešati v odprto okolje, ker so preveč specifična. Te stvari so v glavnem zaprte za zunanjo uporabo, včasih jih lahko damo v kakšno odprto knjižnico zaradi reklamnih namenov, v vsakem primeru pa ta del kode poskušamo obdržati izven modulov, ki padejo pod kakšne ekskluzivne licence, da jih lahko brez težav uporabimo za več naročnikov.

Zadnjo plast predstavljajo kustomizirane aplikacije po naročilu komercialnih strank ali takšne, ki nastanejo v okviru skupnih projektov z industrijo. Ta nivo mora biti v vsakem primeru zaprt za zunanji dostop, saj bo pogosto vseboval informacije, ki lahko predstavljajo poslovne skrivnosti industrijskih partnerjev. V nekaterih primerih tudi znotraj skupine ne bodo imeli vsi dostopa do takšnih delov kode.

References:

[1] Doug Lea: *Draft Java Coding Standard*. Electronic document, available at http://g.oswego.edu/dl/html/javaCodingStd.html.

[2] *C# Coding Standards & Best Practices*. Electronic document, available at http://www.dotnetspider.com/tutorials/CodingStandards.doc.

[3] Igor Grešovnik: *Programmers' guidelines for Development of Software within COBIK & Laboratory for Multiphase Processes*. Treatise, COBIK, 2012.

[4] Igor Grešovnik: *IoptLib*, electronic document at http://www2.arnes.si/~ljc3m2/igor/ioptlib/.

[5] Igor Grešovnik: *IGLib.NET*, electronic document at http://www2.arnes.si/~ljc3m2/igor/iglib/index.html.

[6] Igor Grešovnik: *IGLib Documentation*, electronic document at http://dl.dropbox.com/u/12702901/code_documentation/generated/iglib/index.html .

[7] Igor Grešovnik: *IGLib.NET Code Documentation*, electronic document at http://dl.dropbox.com/u/12702901/code_documentation/generated/iglib/html/index.html .

[8] Igor Grešovnik: *Optimization program Inverse*, electronic document at http://www2.arnes.si/~ljc3m2/inverse/index.html.

Sandbox (this is not a part of this document)