Liger - An Open Source Integrated Optimization Environment

Ioannis Giagkiozis
Department of Automatic Control and Systems Engineering
The University of Sheffield
Sheffield
S1 3JD
i.giagkiozis@sheffield.ac.uk

Robert J. Lygoe
Product Development Europe
Dunton Technical Centre
Ford Motor Co. Ltd
Powertrain Calibration
Calibration CAE & Optimization Methods
blygoe@ford.com

Peter J. Fleming
Department of Automatic Control and Systems Engineering
The University of Sheffield
Sheffield
S1 3JD
p.fleming@sheffield.ac.uk

ABSTRACT

Although there exists a number of optimization frameworks only commercial and closed source software address, to an extent, real-world optimization problems and arguably these software packages are not very easy to use. In this work we introduce an open source integrated optimization environment which is designed to be extensible and have a smooth learning curve so that it can be used by the non-expert in industry. We call this environment, Liger. Liger is an application that is built about a visual programming language, by which optimization work-flows can be created. Additionally, Liger provides a communication layer with external tools, whose functionality can be directly integrated and used with native components. This fosters code reuse and further reduces the required effort on behalf of the practitioner in order to obtain a solution to the optimization problem. Furthermore, there exists a number of available algorithms which are fully configurable, however should the need arise new algorithms can also be created just as easily by reusing what we call operator nodes. Operator nodes perform specific tasks on a set, or a single solution. Lastly as visual exploration of the obtained solutions is essential for decision makers, we also provide state-of-the art visualization capabilities.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments—*Graphical environments, Integrated environments*

Keywords

Multi-objective optimization, industrial optimization, optimization software

1. INTRODUCTION

Recently Michalewicz [20] presented a paper criticizing the evolutionary computation (EC) community for ignoring, what he referred to *actual* real-world problems. His argument revolved about the fact that EC algorithms for optimization will usually have some shortcoming that would make their application in real-world problems infeasible. An example could be that the method does not scale well to the number of decision variables or objectives encountered in practice, or that constraint handling is not as efficient as it could be and also that there is no clear way of bringing all the required elements for the solution of a real-world problem by the practitioner. In our view, this perspective is somewhat exaggerated, however Michalewicz does present a series of reasonable arguments that cannot simply be ignored.

We believe that EC optimization algorithms are in fact applicable in practice, however, it is often the case that an expert is required to apply such methods with success. This difficulty is often reflected when practitioners use inappropriate methods for their problem simply because they find that particular method easy to use. This can result in a number of adverse side effects, from the obvious which is the use of inferior solutions to more subtle effects like the use of single objective optimization algorithms when the problem clearly requires a multi-objective approach. Despite the problems, we have to acknowledge that practitioners operate under tight time budgets, hence sub-optimal choices are some times unavoidable.

In this work, we present an integrated optimization environment which we refer to as Liger[1]. Liger is comprised of several components all of which are orchestrated together to overcome the issues discussed so far. The main component of Liger is a the visual programming language, which enables the practitioner to create what we call optimization work-flows. An optimization work-flow is a sequence of operations that are performed in order. This procedure is used to *solve* an optimization problem. It should be noted that decision making, namely the procedure of selecting a solution output from an optimization algorithm is in fact part of the solution process. To accommodate this we have employed a state-of-the-art visualization library, namely D3 [2]. Another important aspect in real-world problems is the fact that some parts of the code that defines the problem are

[1]The name is borrowed from the animal which is a cross-breed between a lion and a tiger.

usually implemented in a language or tool that is different from the solution platform employed. This can create difficulties and is at best a source of delays. To overcome this barrier Liger has interfaces for a number of programming languages and software packages. In essence these can be used as simply as dragging and dropping the appropriate node in the optimization work-flow. In summary Liger has the following features:

- Collection of Evolutionary algorithms
- Collection of *Classical* optimization algorithms. By classical we mean mostly convex optimization methods, e.g. interior point algorithms.
- Provides an easy to use interface for creation of algorithms
- No lock down policy. Meaning the user should be free to use, or reuse, any part of code created in Liger with little or no additional effort.
- Provision for advanced visualization and data exploration. Data in this context usually refers to the set of alternative solutions.
- Interactive interface during algorithm execution. This is required to accommodate progressive preference articulation algorithms.
- Extendable architecture. No language/software is ever complete, hence there must exist a way that it can be extended naturally and in a straightforward manner.
- Provide a facility for systematic testing of algorithms. There have been a number of attempts towards this direction, see PISA for example.
- A broad picture of Liger is that it is, or at least is being designed to be, an Integrated Optimization Environment (IOE).

The rest of this paper is organised as follows. In Section 2 we elaborate on fundamental definitions in single and multi-objective optimization. In Section 3 we comment on related work and in Section 4 we describe the architecture of Liger. Subsequently in Section 5 we present a case study on a potential use of Liger. Lastly we summarise and conclude this work in Section 6.

2. BACKGROUND

A general definition of a single objective optimization problem is,

$$\begin{gathered} \min_{\mathbf{x}} f(\mathbf{x}) \\ \text{subject to } g_i(\mathbf{x}) \leq 0, i = 1, \ldots, m, \\ \text{and, } h_i(\mathbf{x}) = 0, i = 1, \ldots, s, \\ \mathbf{x} \in S. \end{gathered} \tag{1}$$

The functions $g_i(\cdot)$ are inequality constraints and the functions $h_i(\cdot)$ are equality constraints. Depending on how the functions in (1) are defined the problem described may be convex or nonconvex. Although a detailed description between the two is beyond the scope of this work, a relevant distinction is that convex problems can be solved with relative ease while nonconvex problems can be extremely difficult to solve even when considering a small instance. Namely a problem instance with a small number of inequality and equality constraints and for decision variable vectors ($\mathbf{x}$) of low dimension. Evolutionary algorithms (EAs) have been designed to deal with the latter category, and although there are interior point methods implemented in Liger, our focus in this work is in EAs.

Liger has implementations of single objective EAs, however, we find that often real-world problems have multiple competing objectives. One issue with such problems is that there exists no longer a single optimum, rather an entire set of alternatives that are all *optimal*. A multi-objective problem (MOP) is defined as,

$$\begin{gathered} \min_{\mathbf{x}} \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})) \\ \text{subject to } g_i(\mathbf{x}) \leq 0, i = 1, \ldots, m, \\ \text{and, } h_i(\mathbf{x}) = 0, i = 1, \ldots, s, \\ \mathbf{x} \in S. \end{gathered} \tag{2}$$

The objective function in a multi-objective problem is comprised of k scalar objective functions. The set S is the domain of definition of the decision variables comprising the decision vector, $\mathbf{x}$. For problems defined by (2) the optimal solutions are referred to as the Pareto front or the non-dominated set. Briefly, these solutions are the set of objective vectors for which there exists no other objective vector whose elements are all simultaneously *better* in comparison. Better in the context of minimization means smaller. For the sake of brevity we omit any formal definitions of Pareto optimality, the interested reader is referred to [21] for a comprehensive introduction.

3. RELATED WORK

The number of available software libraries for evolutionary algorithms is steadily increasing with some libraries being the result of almost 15 years of research, see Evolving Objects [13] for example. Currently however, the task of selecting an appropriate optimization algorithm, even for experts in the field, can be a rather demanding task. This issue, in our view, can be to some extent alleviated if there existed a tool that aggregated and extended already existing libraries, provided a graphical programming language and was straightforward to extend. As briefly alluded to in the introduction, such a tool can be termed as an integrated optimization environment. The application that we present in this work, although relatively in the early development stages, poses the features that can classify it in this category.

In this section we briefly review a set of available open source libraries that implement and provide the necessary tooling to create new optimization algorithms. It should be noted however, that it is not our intention to provide a comprehensive comparison of these libraries. Also note that, although we present the number of implemented algorithms that are available within the library as an indicator in Table 1, this does not imply that the library in question cannot be used to create a larger number of optimization techniques. This feature, in our view, is important for the end user as it reduces significantly the required time to apply the particular library on a problem. In other words, the number of algorithms present in the library can be seen as an *ease of use* indicator. Following we provide a commen-

Name	Language	License	GA	ES	DE	PSO	EDA	MO	NA	GUI	Reference
ECJ	Java	AFLv3	✓	✓	✓	✓	×	✓	8	✓	[22]
Opt4J	Java	LGPL	✓	×	✓	✓	×	✓	5	✓	[17]
JGAP	Java	GPL	✓	×	×	×	×	×	2	✓	[19]
EvA2	Java	LGPL	✓	✓	✓	✓	×	×	8	×	[14]
jMetal	Java	LGPL	✓	✓	✓	✓	×	✓	15	✓	[5]
EO	C++	LGPL	✓	✓	✓	✓	✓	×	4	✓	[13]
MOMH Lib++	C++	LGPL	✓	×	×	×	×	✓	9	✓	[12]
Open Beagle	C++	LGPL	✓	✓	×	×	×	✓	11	×	[9]
ParadisEO	C++	CeCILL	✓	✓	✓	✓	✓	✓	12	×	[16]
Shark	C++	GPL	×	✓	×	×	×	✓	15	×	[11]
PISA	C	Mixed	✓	×	×	×	×	✓	13	×	[1]
Heuristic Lab	C#	GPL	✓	✓	×	✓	×	✓	14	✓	[24]

Table 1: Software libraries for population-based optimization methods. Licensing is not discussed in the text as it is beyond the scope of this work. However, the licenses under which the libraries are distributed are listed for convenience. The labels have the following meaning: GA, genetic algorithms, ES, evolution strategies, DE, differential evolution, PSO, particle swarm optimization, EDA, estimation of distribution algorithms, MO, multi-objective, NA, number of algorithms and GUI, graphical user interface. A, ✓, signifies that the library supports the corresponding feature and a, ×, that this functionality has not been implemented.

tary on the libraries listed in Table 1 and elaborate on the reasons that led us to include some of them in Liger.

Although every library listed in Table 1 has its merits, jMetal [5], PISA [1] and ParadisEO [16] seem to be most widely employed by the community. In particular, arguably due to its popularity, a small subset of jMetal has also been ported to C++ and *.NET*, however these forks are not so actively maintained as the Java version hence their features are somewhat limited in comparison. ParadisEO, is an extension of the so called *evolving objects* (EO) library [13]. EO is a C++ template library, which as their authors mention is better suited for an advanced user. However, the capabilities of the library are extensive and it has no other dependencies apart from the standard template library in C++. A potential issue with EO and in extension with ParadisEO could be the way memory is managed, namely there exists an object which is granted ownership of every created operator. Once this object becomes out of scope, due to the method or function where it is contained has returned for example, all the operators and results are lost. This however could be overcome in a number of ways whose enumeration is beyond the scope of this work.

PISA [1] is a modular framework which has been primarily written in C. The architecture of PISA is modular and communication between different *modules* is accomplished using text files. In extension this means that the various modules can be written in any language, at least in principle. This feature can also be seen as a weakness for PISA since hard disk input output (IO) operations are relatively slower to operations performed using the free store (RAM). Consequently, the benefit of language independence is gained at the cost of slower execution times as polling is used for synchronizing the various modules.

Another very interesting software is Heuristic Lab [24]. Heuristic Lab (HL) is at a higher level compared to the rest of the libraries seen in Table 1 in terms of its implementation. Namely, HL has a modular architecture build about a plugin-based system, which allows for extensions to be added to the tool. Although the ideas and philosophy upon which HL is based are sound in terms of software engineering, HL appears to be rarely used in practice[2]. A problem with HL is that the selected language is C#. C# is a language created by Microsoft and in extension well supported only on Windows operating systems. Also, C# has an intermediate step (bytecode) in its compilation phase which often has a significant negative impact on performance. Our argument is that HL is non-portable and arguably many practitioners in industry employ Linux for their experiments that require high performance computing, for example clusters.

It can be seen (Table 1) that a number of libraries have been implemented in Java. Arguably the authors of ECJ [22], Opt4J [17], JGAP [19], EVA2 [14] and jMetal [5] aim at cross-platform code. It is true that Java is one of the most employed languages with an array of devices having a Java virtual machine implementation. Nevertheless, it is difficult to argue that Java is the best language for computationally intensive tasks. In comparison it is much easier, in terms of development expertise, to produce code that executes faster in C++, however there is a perception, and to some extent rightly so, that C++ code is relatively less portable. This however is not true for the C++ based libraries seen in Table 1, which can be used on all of the 3 major platforms, namely MAC OS, Linux and Windows, and perhaps others.

4. LIGER - ARCHITECTURE

Liger is based on a small set of open source libraries and applications. The general architecture, namely the underlying framework used to extend Liger is based on a subset of QtCreator, the integrated development environment created by Nokia in support of the Qt library. The main reason for this choice has been expediency, as a large number of low level utilities are already in place in QtCreator, and, have already undergone several debugging iterations. For this reason the Liger shares several features with the architecture of QtCreator. The main plugin, upon which all other extensions depend on, is the UI Core, see Fig. (1). The Core plugin provides the necessary interfaces for fundamental tasks

[2]To the authors' best knowledge.

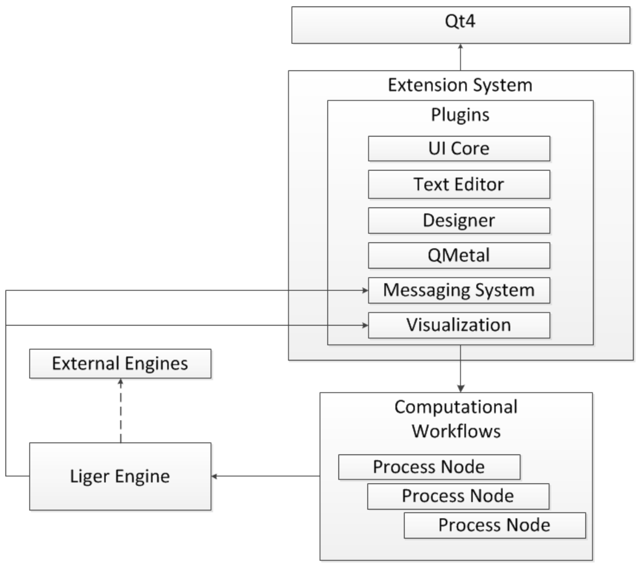

Figure 1: Liger architecture.

such as communication, file input-output operations, handling of settings as well as classes implementing parts of the user interface and also a manager for the Liger designer, see Fig. (2). Note that the designer in Liger shares only the name of the designer in QtCreator. The main plugins that comprise Liger are:

- Designer
- Liger engine
- External interfaces library
- Visualization.

These 4 plugins are further discussed in the following sections. The QMetal plugin, is a wrapper for an extended version of the jMetalCpp library which is a port of the jMetal library to C++. Although the jMetalCpp library is fairly minimal, its design principles are in line with the design patterns used in Liger. Nevertheless, as the number of features provided by jMetalCpp are only a few and focused towards evolutionary computation, Liger could not rely on the provided version by the authors (see Table 1). In what follows we further elaborate on the main plugins that comprise Liger.

4.1 Designer

A part of the Designer structure is seen in the class diagram in Fig. (4). The fundamental computational unit in Liger is the *process node*. Every process node extends and the *IProcessNode* class. This interface class provides the necessary functionality for communication with the Liger engine. A process node is comprised of zero or more input ports, zero or more output ports, a set of connections to the Liger engine and a virtual computation method. The computationally intensive part of a process node which is invoked by the *evaluate()* method can be executed in a thread, out of process or even in a cluster. At present, execution of the aforementioned method is restricted to a different thread from the main user interface. In single threaded CPUs the main thread is reused. In contrast with simulink and XCos in Scilab [3], Liger propagates a *rich* data structure. This is to simplify the configuration of every node. This is not limiting to the least and is also efficient as the data structure, namely the *IData* class, contains only pointers to the process nodes that own the data.

4.2 Liger Engine

The Liger engine is the part that coordinates the order of execution within an optimization work-flow. This plugin bears some resemblance to the Simulink engine. The optimization work-flow in Liger starts at the *Master Start Node* (MSN) and ends at the *Master End Node* (MEN). These two nodes are unique per design and simply provide a frame of reference for the start and end of an optimization loop. When initialised, the engine passes control to the MSN which in turn signals the nodes to which it is connected. This process is reactive in the sense that once a node receives all input signals it notifies the engine that it is ready to proceed to the evaluation stage. Subsequently, the engine invokes the *evaluate()* method of the node and once complete the node emits the result to the node or nodes connected to its output ports. This process is repeated until the termination condition, which is implemented in the MEN node, is reached. A problem with this paradigm is that for a node to become *ready* for evaluation, it must receive all inputs, which may never happen if an input is an output of a node that has not yet been evaluated. This scenario describes what is commonly referred to as feedback loop. A solution to this is the implementation of an internal loop mechanism using a *Start Node* and an *End Node*, which can emulate, to a degree, a feedback loop.

Another mechanism that is implemented in the Liger engine is the identification of the external tools necessary to execute the optimization flow. For example, if the practitioner has added a MATLAB node in Liger, then this will require access to the MATLAB engine. Liger opens a connection with the corresponding engine once it detects that there exists a node that will require such a connection and, naturally, this connection is closed when all nodes that necessitated this connection are deleted from the optimization work-flow.

4.3 External Tools

To foster code reuse we have setup a mechanism to allow for external tools to be connected with Liger. This of course is subject to an available API[3] on behalf of the application in question. For instance, MATLAB provides an API that allows access to its interpreter and Simulink. At this stage we have created a plugin named, MATLAB, which provides the necessary tooling to execute .m files in their native environment, query the results and use these directly in Liger. The ease of use for the industrial non-expert user is illustrated in Section 5.

4.4 QMetal

From the libraries listed in Table 1 we have singled out, EO [13], ParadisEO [16] and the C++ version of jMetal [5]. We selected EO and ParadisEO because, in our view, these libraries provide an unrivaled number of features. However, as mentioned in Section 3, the memory management model employed in EO and ParadisEO is posing difficulties as the

[3]Application Programming Interface

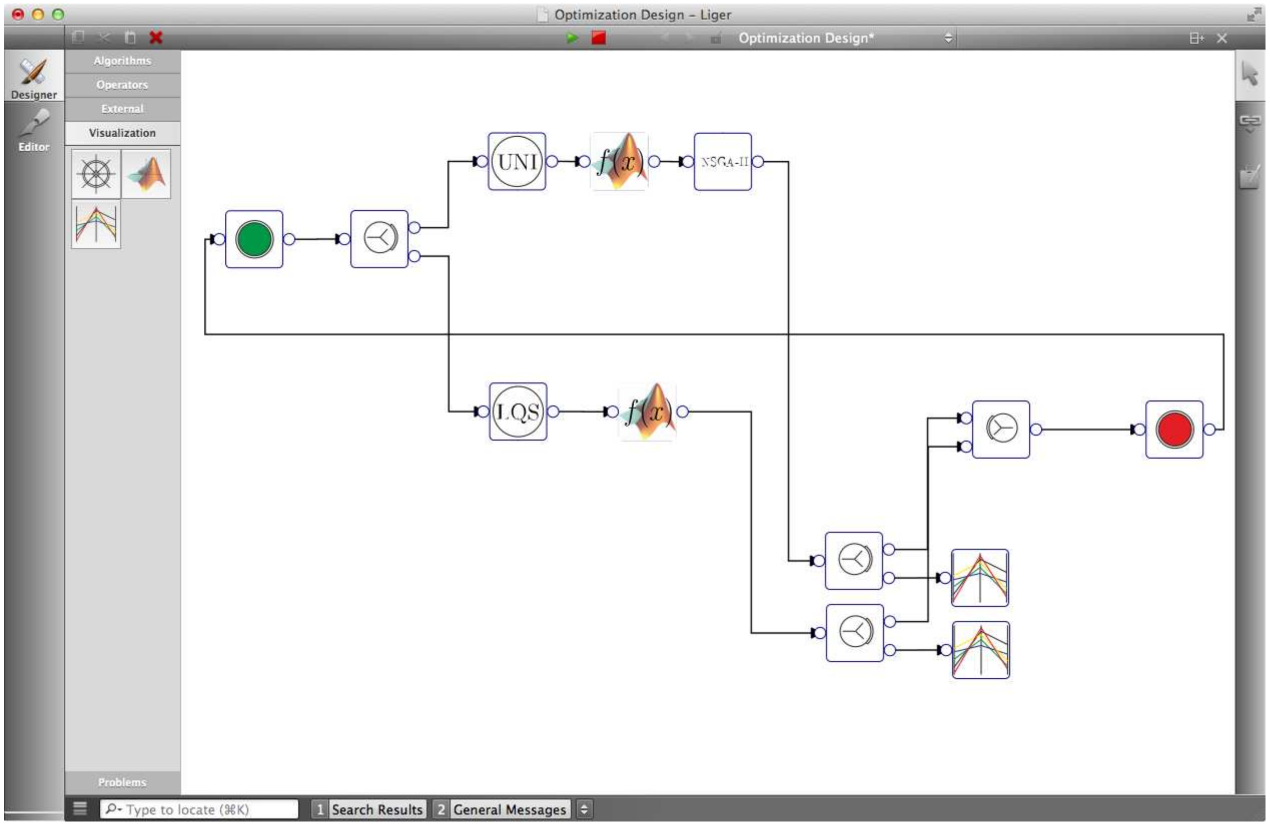

Figure 2: A view of the Designer environment in Liger. Note that the leftmost node (green disk) and the rightmost node (red disk) represent the *Master Start* and *Master End* nodes respectively.

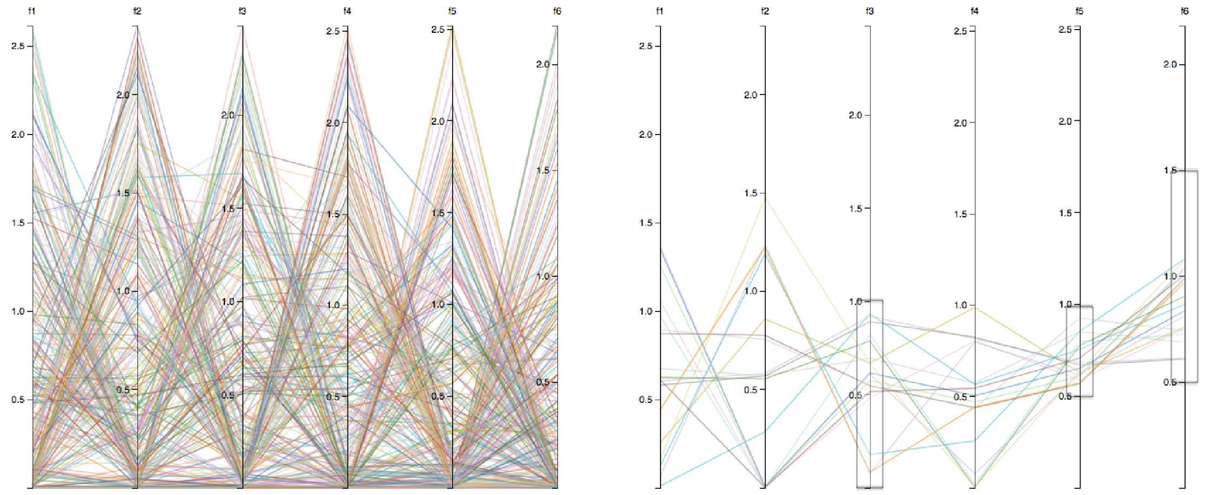

Figure 3: Parallel coordinates plot node in Liger. An approximation of the Pareto front obtained in Liger using NSGAII [4] for the DTLZ2 test problem [10] (see Fig. (2)). The plot to the left depicts the set of all alternatives produced by the algorithm while the plot on the right depicts the selected set of solutions by the decision maker.

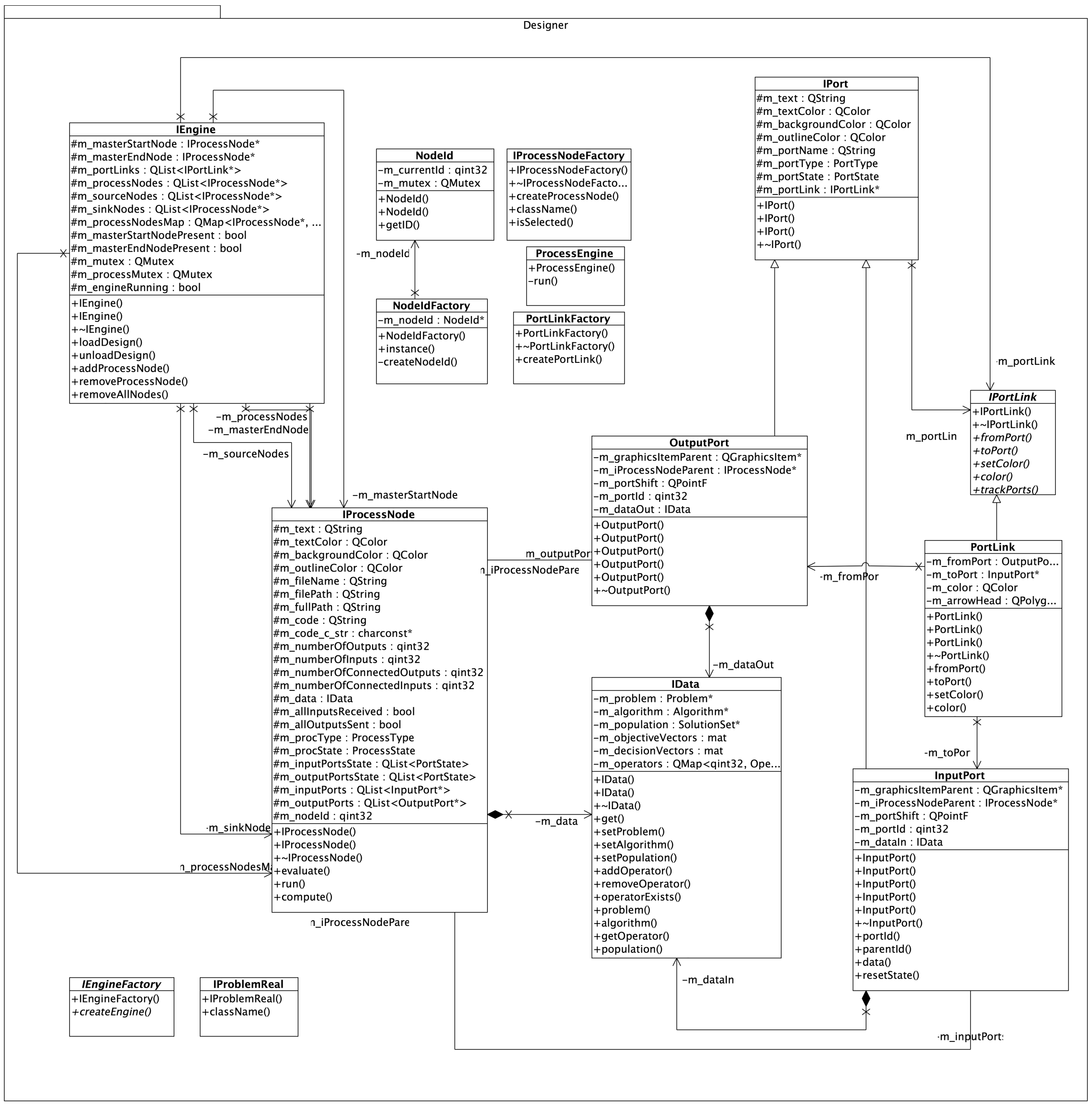

Figure 4: Liger architecture.

process nodes in Liger have ownership of the memory they allocate when creating new operators, algorithms, visualization elements or any other task. The reason behind this choice is to allow for scalability and minimize memory leaks. However, ParadisEO, which inherits this feature from EO, has a state object which manages every object created by the library. This clash has almost been resolved to some extent and modifications are to be submitted to the ParadisEO developers.

The second library of choice, namely the C++ version of jMetal, was selected on the basis that it is build on sound principles using design patterns [7] extensively. These choices enhance code readability and reuse to a significant degree. However, as the C++ version of jMetal is somewhat outdated, we have simply used a modified version for low level interfaces. Nevertheless the architecture of this plugin is based on jMetal hence the name has been selected to reflect this choice.

Presently this plugin provides nodes for the WFG1-9, DTLZ1-7, ZDT1-6 and LZ09 1 to 9 test problems. In addition the following algorithms are available:

- Differential Evolution [23]
- Generalized Differential Evolution 3 [15]
- NSGA-II [4]
- Particle Swarm Optimization [6]

Although this list is not large, it should be noted that all the operators that the above mentioned algorithms employ are also available as independent nodes, which can be used for algorithm creation within Liger. Nevertheless, this list will be extended to at least include MOEA/D [25], RM-MEDA [26], by the *alpha* release of Liger, see Section 6.

4.5 Visualization

The presentation and exploration of results is arguably very important. For instance, we find that the time spent on refining quality plots is a significant portion of time spent preparing a report. Although this time cannot always be eliminated, part of this process can be streamlined by the underlying software. Additionally, there are algorithm frameworks that require user interaction during the optimization process, for example [8]. Although such interaction can be accomplished using other tools, if this functionality is not present in Liger, this would pose difficulties in using our IOE. Another problem is that a high level library for visualization is not available in C++[4], an issue that would lead to excessive development time for the implementation of such a functionality in Liger.

These issues are resolved using a combination of the *QWebkit* class, which is a port of the Webkit library to Qt and D3 which is a javascript based visualization library [2]. D3 creates SVG (Scalable Vector Graphics) and allows for user interaction. Additionally there is ample documentation and examples for this library, and its performance exceptional as well [2]. Naturally, in terms of performance it cannot be compared to a native implementation in C++, however the achieved savings in development time the wealth of documentation for the D3 library and the support of the Qt library to render SVGs to many alternative formats, outweigh the performance degradation.

The visualization elements in Liger are implemented simply as another node, hence they also inherit from the *IProcessNode* class, and follow the same rules of evaluation as other nodes. The difference is only in the output which is visual. To implement a visualization node, there are mainly 3 options:

- Send the data to an external software. This is subject to availability of such interface within Liger. For example this capability exists for MATLAB and we intend to extend this to other software packages. Nevertheless this may not always be an option.

- Implement a plot using a C++ library, however this has the shortcomings portrayed above.

- Alternatively, create a plot using javascript and D3 [2] and then use the interface provided in Liger to establish a communication path so that data can be transmitted back and forth from the C++ code and javascript.

Of course these options may not be necessary as there is an increasing library of available visualization nodes with scatter plots, parallel coordinate plots (see Fig. (3)), bar charts and histograms.

5. CASE STUDY

In this section we demonstrate an optimization work-flow using Liger. The selected problem is a 6-objective instance of the DTLZ2 test problem. In a future work we plan to demonstrate a work-flow on a 6-objective diesel engine calibration problem [18].

Let us assume that the decision maker has an optimization problem whose objective function is implemented in MATLAB. The only requirement that an objective function implemented in MATLAB has to fulfill so that it can be used directly in Liger is that when invoked with no arguments it returns the number of decision variables and the number of objectives. When the function is called with a single argument, namely the population of candidate solutions, the only output must be an array whose rows are the different objective vectors. If the function in question does not satisfy these criteria, it is straightforward to create a wrapper using no more than 4 lines of code.

At this stage the practitioner would like to explore the solutions obtained by an optimization algorithm and compare the results with a latin hypercube sampling search approach. These alternative methods have been implemented and can be seen in Fig. (2). The top flow is using the NSGA-II [4] algorithm, while the bottom is using a grid of points which has been generated by the LQS node subsequently these points are evaluated and then displayed. Subsequently the decision maker can activate the parallel coordinate nodes and explore the solutions, Fig. (3). Note that we present only the solutions obtained by the top optimization work-flow due to space limitations.

During the entire procedure described above the practitioner was only required to write 4 lines of code, which in fact are available as a template in Liger for convenience. The same simplicity governs nodes that handle constraints, which can be bound constraints, equality constraints or inequality constraints.

6. CONCLUSIONS AND FUTURE RESEARCH

Building software that is useful to its creator as well as to a wider community of users is mostly an art and most likely will remain so for the foreseeable future. Liger is an integrated optimization environment whose main focus is to enable non-expert practitioners in industry to use state-of-the-art optimization algorithms. In the case that no available algorithm matches perfectly the problem, then there exists the option of creating customised algorithms tailored for the problem simply with a few mouse clicks. Furthermore, as new optimization algorithms become available the architecture of Liger is perfectly suited to accommodate such additions in an equally straightforward fashion.

As mentioned in the introduction, no software is ever complete and Liger is no exception to this rule. We continuously improve Liger and have several features planned for implementation, some of which are a plugin for the Liger engine so that it can execute process nodes on the Cloud and on a grid. Furthermore we consider code generation for Liger, initially in C++. The idea is to output the optimization work-flow created, that potentially implements some algorithm or experiment and produce code that has as few as possible dependencies to external libraries. Although we have already some simple algorithms that can be produced in such a way, this is a massive undertaking and would require extensions for several libraries used within Liger.

Liger is made available as an open source software under the LGPLv3 software license and an *unstable* release is due at the end of June 2013 and an *alpha* version of the software will be released on September 2013.

[4]To our knowledge.

7. ACKNOWLEDGMENTS

The authors would like to acknowledge the financial support of Ford and The University of Sheffield and collaborators in the Technology Strategy Board Programme: CO_2 Reduction through Emissions Optimisation (CREO Project).

8. REFERENCES

[1] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA — A platform and programming language independent interface for search algorithms. In *Evolutionary Multi-Criterion Optimization (EMO 2003)*, Lecture Notes in Computer Science, pages 494 – 508, Berlin, 2003. Springer.

[2] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[3] S. L. V. Campbell, J.-P. Chancelier, and R. Nikoukhah. *Modeling and simulation in Scilab/Scicos*. Springer Science+ Business Media, 2006.

[4] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[5] J. J. Durillo and A. J. Nebro. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760 – 771, 2011.

[6] R. Eberhart and J. Kennedy. A New Optimizer Using Particle Swarm Theory. In *International Symposium on Micro Machine and Human Science*, pages 39–43. IEEE, 1995.

[7] G. Erich, H. Richard, J. Ralph, and V. John. Design patterns: elements of reusable object-oriented software. *Reading: Addison Wesley Publishing Company*, 1995.

[8] C. Fonseca and P. Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In *Conference on Genetic Algorithms*, volume 423, pages 416–423, 1993.

[9] C. Gagné and M. Parizeau. Genericity in evolutionary computation software tools: Principles and case study. *International Journal on Artificial Intelligence Tools*, 15(2):173–194, 2006.

[10] S. Huband, P. Hingston, L. Barone, and L. While. A Review of Multiobjective Test Problems and A Scalable Test Problem Toolkit. *IEEE Transactions on Evolutionary Computation*, 10(5):477–506, 2006.

[11] C. Igel, V. Heidrich-Meisner, and T. Glasmachers. Shark. *Journal of Machine Learning Research*, 9:993–996, 2008.

[12] A. Jaszkiewicz and G. Dabrowski. Momh multiple-objective metaheuristics. `http://home.gna.org/momh/index.html`, 2012.

[13] M. Keijzer, J. Merelo, G. Romero, and M. Schoenauer. Evolving Objects: A General Purpose Evolutionary Computation Library. In *Artificial Evolution*, volume 2310 of *Lecture Notes in Computer Science*, pages 231–242. Springer Berlin Heidelberg, 2002.

[14] M. Kronfeld, H. Planatscher, and A. Zell. The EvA2 Optimization Framework. In C. Blum and R. Battiti, editors, *Learning and Intelligent Optimization Conference, Special Session on Software for Optimization (LION-SWOP)*, number 6073 in Lecture Notes in Computer Science, LNCS, pages 247–250, Venice, Italy, Jan. 2010. Springer Verlag.

[15] S. Kukkonen and J. Lampinen. GDE3: The Third Evolution Step of Generalized Differential Evolution. In *IEEE Congress on Evolutionary Computation*, volume 1, pages 443–450. IEEE, 2005.

[16] A. Liefooghe, M. Basseur, L. Jourdan, and E.-G. Talbi. Paradiseo-moeo: A framework for evolutionary multi-objective optimization. In *Evolutionary multi-criterion optimization*, pages 386–400. Springer, 2007.

[17] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich. Opt4j: a modular framework for meta-heuristic optimization. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1723–1730. ACM, 2011.

[18] R. J. Lygoe, M. Cary, and P. J. Fleming. A many-objective optimisation decision-making process applied to automotive diesel engine calibration. In *Simulated Evolution and Learning*, pages 638–646. Springer, 2010.

[19] K. Meffert and N. Rotstan. JGAP: Java genetic algorithms package, 2010.

[20] Z. Michalewicz. Quo vadis, evolutionary computation? *Advances in Computational Intelligence*, pages 98–121, 2012.

[21] K. Miettinen. *Nonlinear Multiobjective Optimization*, volume 12. Springer, 1999.

[22] L. Panait, G. Balan, S. Paus, Z. Skolicki, E. Popovici, K. Sullivan, J. Harrison, J. Bassett, R. Hubley, A. Chircop, et al. ECJ: A java-based evolutionary computation research system. 2010.

[23] R. Storn and K. Price. Differential Evolution - A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces. *International Computer Science Institute-Publications TR*, 1995.

[24] S. Wagner, A. Beham, G. Kronberger, M. Kommenda, E. Pitzer, M. Kofler, S. Vonolfen, S. Winkler, V. Dorfer, and M. Affenzeller. Heuristiclab 3.3: A unified approach to metaheuristic optimization. In *Actas del séptimo congreso español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'2010)*, page 8, 2010.

[25] Q. Zhang and H. Li. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

[26] Q. Zhang, A. Zhou, and Y. Jin. RM-MEDA: A Regularity Model-Based Multiobjective Estimation of Distribution Algorithm. *IEEE Transactions on Evolutionary Computation*, 12(1):41–63, 2008.