The Pointer MDO Framework

Dr. Alexander Van der Velden  Synaps, Inc., velden@synaps-inc.com, President
Dr. David Kokan    Synaps, Inc., Pointer Development
Dr. Olaf Frommann   Synaps, Ing-GmbH, Germany, CEO, o.frommann@synaps-ing.de

Introduction

The purpose of the Pointer MDO framework is to create better designs at lower cost and lower cycle time.

The Pointer MDO framework facilitates better design by allowing a more thorough search of the design space over a wider range of design assumptions (what-if, economic, and political scenarios). Currently, in many companies, the use of time-intensive engineering analyses as well as large design teams has led to unacceptable design cycle times. It is often the case that the economic assumptions under which the design was started are no longer even valid over the design cycle! Nevertheless, this complex effort is necessary to create products with the level of performance, certainty, and safety that can compete in the marketplace. Therefore the purpose of MDO software should be to make the process of engineering design more effective and more efficient while still resulting in a product of the highest quality.

Another function of the MDO framework is to give the user support in defining his (design) problem in a concise fashion by experimenting with many different problem specifications. Only a fully automatic process offers the speed to overcome the problem-definition bottleneck in the learning phase.

The introduction of MDO frameworks today is analogous to the introduction of robots (at GM) in the 1950’s. The robots, although technically sound, initially failed because no infrastructure existed to support robot operation. For instance, the location of parts on the factor floor was changed continuously - not a problem for people, but disastrous for the sensitory-impaired robot.

Every month the robot team was asked to add yet another feature to adapt the robot to the then-current factory infrastructure. In frustration the robot development team left GM and started their own company. They sold their ideas to Toyota in the 1970’s. The Toyota plants were laid out specifically for one-armed robots and the rest is history. It should come as no surprise that design organizations also need to change their modus-operandi in order to best use this new tool.

Fig. 1. One-armed industrial robot.

The time for design automation is now. Currently computers are fast enough to solve any well-posed mathematic problem faster and better than a human being or team of humans. This was clearly proven four years ago when IBM’s ‘Deep Blue’ defeated Kasparov in a chess match. Chess is a well-defined game and, therefore, Kasparov’s brilliance and intuitive talent was no match against a fast computer and an optimization algorithm. In his book The 20th century, Peter Jennings deemed this the most significant event of the last decade, and he may well be right.
The Framework

Synaps Product Line

There are two versions of the Synaps MDO framework:

a) **Pointer.** The purpose of this tool is to provide automatic process execution in a homogeneous (same network, same OS) environment. Processes are executed serially. The framework is based on high level open software standards. Pointer has a very powerful automatic optimization engine and is able to optimize complex nonlinear problems with up to 200 variables.

b) **PointerPro.** The purpose of this tool is to facilitate fully automatic design in a heterogeneous computational environment. Processes are executed in parallel and executed more efficiently. The framework is based on proprietary technology and possesses the same fully automatic design optimization features as Pointer.

Pointer is provided on WinNT, Linux, SGI, and HP. Pointer can be custom compiled to run on most any other flavor of Unix.

**Pointer design criteria**

The software should be able to support the type of expensive design analyses and complex design space searches necessary to improve today’s mature designs.

Table 1: The range of largest non-linear problems that (practically) can be solved fully automatically within one weekend of computational time. (***)

<table>
<thead>
<tr>
<th></th>
<th>Expensive</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variables</td>
<td>1</td>
<td>200</td>
</tr>
<tr>
<td>Function call</td>
<td>3.5 h.</td>
<td>1 min</td>
</tr>
<tr>
<td><strong>Pro</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variables</td>
<td>1</td>
<td>200</td>
</tr>
<tr>
<td>Function call</td>
<td>35 h.</td>
<td>10 min</td>
</tr>
</tbody>
</table>

(**) A weekend is typically the largest computational block (for both student and industry) for practical problem solving. The problem is assumed so complex that 20 calculations are necessary to find the optimum using a 1-D parameter scan. In the case of PointerPro we assume that the analysis time is reduced by an order of magnitude by efficient use of parallel computing (16 CPU’s).

Apart from automatically solving the design problem, it is also of crucial importance that the MDO framework can manage vast amounts of data. This data is required in order to quickly judge whether the mathematical problem statement indeed corresponds to the true design problem and to pick the best design out of many pareto solutions.

For this purpose, Pointer can automatically store up to 200 engineering plots for up to 1000 iterations. This corresponds to about 10 GB of data. For all intents and purposes, Pointer can (and should) handle all the data any engineering team could generate or judge. This feature is not available in other (more expensive) MDO frameworks such as ISight. Without it, the user is left to identify design progress based on abstract parameters only (see parameter plot).
The user should be able to deploy the software with minimum up-front cost. To achieve the highest level of ergonomics, Synaps uses the following list of criteria (in order of preference).

1. Software **automatically** does what the user asks (no manual “steering” necessary).

2. The software **options are intuitive**.

3. The software **only asks questions that the user knows the answer to**. No expert knowledge of optimization is assumed or expected.

4. The user can solve the (design) problem by **using software standards** instead of proprietary software that requires new training.

5. If the user is unable to express his problem, **expert consulting** should be available.

---

![Figure 3](image_url)

**Fig 3.** (Standard Pointer) Front GUI showing the parametric database and one of 200 supporting engineering plots for every design iteration. Also note the unique graphical video playback options of the whole design database (1000 iterations) and custom project graphic.
**Problem Specification Interface**

The problem specification phase is arguably the most difficult part of the MDO process. Pointer users describe engineering problems in terms of an objective, design variables and constraints, or in terms of a fuzzy logic specification.

In our experience almost any problem can be cast in those terms. Multiple objectives can be summed and weighted to a single objective since is impossible to compare apples and pears. However, there needs to be a feedback between this rigid specification and the price paid in terms of the performance of the design.

Three approaches are possible in Pointer:

1) Specify the problem in a rigid way (objective, variables and constraints), run the optimization, and scan the design space of the stored solutions for all pareto solutions (solutions with similar total objective value, but different level of constraint adherence).

2) Change the constraint boundaries and re-optimize. This will give you the sensitivities of the objective (cost function) with respect to the ‘hard’ constraints.

3) Specify your problems in terms of fuzzy logic. A design’s level of merit is defined by applying simple criteria, e.g. desirable, acceptable, or undesirable, together with interdependencies (IF drag is desirable AND weight is acceptable, THEN solution is acceptable). This use of verbal expression makes it easy to exploit the designer’s existing knowledge in order to specify the problem.

The user-friendliness of the problem specification phase is critical. Because of Pointers intuitive GUI (Fig. 3) and multi-block parametric database used to specify the problem, the user can solve problems with up to 1000 parameters, design variables and constraints. Synaps can easily increase this number upon request for those users who need to (and are capable of) defining problems of this size.

The user also can quickly exchange the analyses that are used to compute the objective function and constraint values. In Pointer this is accomplished using the Analysis Scheduler (Fig. 5). Currently all processes in Pointer are scheduled in series. Simple low level programming is required to set up more complicated analysis sequences (conditional branching, iterations, etc). PointerPro has advanced features that allow complex branching and iterative formulations without low level programming. Both Pointer and PointerPro allow the user to graphically delete, replace, and insert new analyses into the process.

Pointer and PointerPro also allow the execution of both legacy (with obsolete software standards) and proprietary codes (no access to source). If the source is available (in C or Fortran), it is possible to compile your application as one big application. However for most applications input and output file parsing is used (Fig. 4, see next section) to communicate with executables. It is also possible to communicate with GUI based proprietary applications such as Microsoft Excel. In our experience, however, such proprietary API links provide an undue penalty on computational performance, accuracy and process robustness. Therefore, they are not (currently) recommended.

The problem specification in Pointer is extremely flexible and powerful. Pointer can run itself or any other engineering optimization tool as an application. This allows for the efficient use of multi-level sub-optimization schemes. It is even possible to run a large sub-optimization on a Linux platform in Europe, even when the Pointer user is behind an NT workstation in the US as we recently demonstrated.

The problem specification process with Pointer is no different than the conventional established iterative design process in which the true goals of any engineering endeavor are discovered and met. The difference is that, with our framework, the iterations are done fully automatically and every problem is solved to the same quality standards. Experience has shown that iterative problem specification using Pointer results in a much faster learning curve for beginning engineers than the process of participating in a traditional design process where design goals are often changed without converging each design.
Fig. 4. The template editor allows the user to define for Pointer how to prepare input files and extract data from output files. In this case the green column is defined to specify the data for the Y-axis of plot number 11.

Analysis Automation and Data Access

Each Pointer project has a parameter database, which is the set of design parameters for a single iteration. After executing a “job” (optimization, parameter scan, etc.) a design database is created, which consists of a set of parameter databases (now having outputs fully consistent with inputs) and up to 200 supporting engineering plots. Pointer has three methods for passing information between itself and the (MDO) analysis codes.

1) Compiler-linked mode. Data is passed by using API calls of the type GET or PUT for integers, real numbers, and arrays. It is also possible to automatically create design iterations support plots by passing the data of local arrays to the design database.

2) ASCII template mode. The user marks parameters, either scalar or vector variables, using a graphical interface and connects them to self-defined parameter names. These parameters are then automatically replaced by the actual values during the scheduling. After each iteration improvement, the parameter database is stored in the design database. It is also possible to extract plot data directly from ASCII output. In this way, up to 200 engineering plots per iteration can be stored in the design database.

3) Mixed compiler and template mode. Templates can be used inside compiled code.

To display the design data, the user can select a video type scan of the whole database of designs. If he would view all possible frames at a speed of 2 per second, it would take more than 24 hours to see every design plot.

Currently, all data is saved in ASCII-files. Since Pointer uses the Perl standard for its database operations, it is possible to use its DBI interface to various commercial databases (see www.perl.com). So with a minimum of effort, the user can move data from the Pointer design database to Oracle or to Sybase for example.

For PointerPro, an interface to write these data into an SQL-based relational database is high on the list of extensions and will be implemented in the near future. Furthermore, there exist plans to couple PointerPro to EDM/PDM systems.

It is also possible to couple Pointer directly to parametric CAD systems such as Solidworks. In this case the design parameters that define the parts are controlled from the Pointer interface. For non-parametric CAD systems (such as CATIA) a customized geometric interface needs to be created that transfers the parametric points into surfaces. The big advantage of such a system is that at the end of the optimization job all the associated CAD drawings of the design can be used for management presentations and further technical analysis.

The PointerPro process chains are defined graphically by the user. The user creates a logical schedule of tasks including loops and parallel distribution of jobs in a completely graphical way. The Schedule is subsequently interpreted by a dedicated client program on a selected host machine. The creation of input files and the parsing of output files are done through the definition of templates on the basis of example files.

In Pointer, only serial processes can be defined. The PointerPro version adds the option of
running analyses in parallel with the Parallel-Wizard communication protocol. This protocol accounts for the machine endianess (location of the least significant byte in data word), so data is transferred in binary format to ensure accuracy. The mounting of disks is unnecessary. In fact, PointerPro/Taskforce was designed with heterogeneous networks of computers in mind. Parallelization is available on the basis of parameter sets that can be calculated in parallel (coarse grain), distribution of subsets like design points or product parts (medium grain) or by using the features of the analysis codes (fine grain).

Both Pointer and PointerPro can be run in batch mode, so that a problem can be defined without manual intervention.

Pointer also has a job script language, which allows the user to automatically submit multiple complex jobs in series. This is especially useful when large databases of optimal designs are created or when the influence of various design assumptions and constraints on the cost function is assessed.

Fig. 5. The serial analysis scheduler in Pointer defines the I/O templates and the serial execution of processes. More complex scheduling can be done with lower level programming.
Fig. 6: PointerPro. Taskforce parallel analysis scheduler. Note the capability to monitor and distribute jobs on several individual machines in a network
Problem solution through optimization

Pointer solves the defined mathematical problem (see problem specification interface) in a fully automatic fashion. The key development driver for Pointer’s fully automatic optimizer was the following:

“Pointer should give the expert user the ability to find a better answer to a wide class of complex design problems in less time than it would take him to solve the problem in a conventional way”

This, of course, includes the user’s effort to learn to work with the optimization software. Dr. Tong of Engineous Software Inc. said:

“Systems that require frequent tuning of search parameters by users are merely transferring the iteration of physical parameters to the iteration of numeric parameters unfamiliar to the designers.”

We completely agree with this point of view, but do not believe that Engineous’s optimization product ISight has achieved this goal. ISight still leaves the user to select from more than ten algorithms and requires the user to control their internal settings.

Pointer automatically controls a hybrid combination of the best current set of optimization algorithms available (Linear Programming, Sequential Quadratic Programming, Downhill-simplex, and Genetic). Many other popular algorithms were tested extensively by Synaps, including variable metric gradient and simulated annealing. These offered overlapping functionality and were therefore unnecessary.

Linear and sequential quadratic programming quickly find the solution for smooth linear and non-linear objective functions. They both assume that the problem is driven by constraints to find the solution more quickly. Downhill-Simplex (Nelder and Mead) is a very powerful DOE (design of experiments) method. It assumes that the solution can be found by using information from the edges of the computational domain. Unlike SQP, it can cope with very rough and nonlinear objective functions. Although downhill-simplex can deal with large discontinuities, Pointer’s genetic algorithm is superior for coping with very pathological topographies. It finds the answer by (partially) random search techniques.

These core optimization techniques are controlled with a proprietary, self-learning, adaptive algorithm. The algorithm optimizes the sequence of the core optimizers, number of iterations and the optimizer algorithm ‘tweaking’ parameters based on solution robustness (probability to get the right answer) and available cycle time.

Pointer does not require any intervention from the user or specialist knowledge about optimization. When the user starts his problem, he only indicates how much time is available and what type of topography he is solving (Linear, smooth, etc). He is completely free to answer “I don’t know” to the latter question.

This is all Pointer needs to know. In Pointer classes we demonstrate that complete novices get better solution with Pointer than experts with by hand iteration in a fraction of the wall clock time. For those who are interested in optimization Pointer provides a means to use the core algorithms individually.

In our experience, Pointer requires T hours to solve a problem:

\[ T = Z \times \text{variables} \times \text{Time-per-design-simulation} \]

Z, the relative complexity of the problem, is typically 50, the first time a highly non-linear engineering problem is solved. The initial investment of so many function calls is essential to understanding the problem posed by the topography of the objective function. Pointer learns from its experience and typically reduces Z to a number below 10 over time. This saves a lot of time when optimizations are repeated.

Pointer also provides an interface for automatic one-dimensional parameter scans. This is particularly useful in determining optimality conditions and sensitivities at the end of the optimization run.

Some MDO methods that are included in competitors packages are not included as options in Pointer because they do not add (sufficient)
functionality, they reduce efficiency, or they lead the user to the wrong conclusions. Specifically:

a) Simple (quadratic) response surfaces. Currently the user can create their own response surfaces by simply fitting any type parametric expression to a dataset. An example is included in the Pointer manual. Since most functional topographies are not of the quadratic type, there is little purpose in offering such an automatic feature.

b) An expert system based on logical statements from the user. This method is not effective in producing better solutions. Solutions created by this method are a priori based on existing knowledge and are therefore not improvements. It is also virtually impossible to guarantee the consistency of such a system.

To date we have never had a user who contacted us who felt they needed those options because Pointer was unable to solve their problem. Pointer’s learning curve is very fast. Our user’s manual is only 100 pages, and our hands-on training course takes only one day. We see these as numbers hard limits for future software development in terms of adding options.

Fig. 7: Optimizing with Pointer is as easy as clicking the ‘I know nothing’ box.

A flexible and powerful architecture.

Pointer uses a FORTRAN computational core and open source extensible TCL/TK graphical user interfaces. All database operations are performed by open source Perl scripts.

PointerPro is based on Java and C. The communication is based upon a lean protocol on top of TCP/IP allowing very high performance even when the amount of data is large. There are no additional packages like CORBA, MPI or PVM used, which makes it simple to maintain the software on different platforms, reduces complexity of installation and the risk of errors due to increased software dependencies.

Due to it’s parallel analysis capabilities, PointerPro is more efficient that Pointer for very large, time-intensive problems. However, whereas Pointer is based on open-source (Perl TCL/TK) software that can be easily extended by the user (user has access to the scripts), PointerPro has a completely proprietary engine.
Fig. 8: The change of each parameter with respect to iterations or other parameter can be monitored real-time.

A few applications

Material design

Research institutions such as Georgia Tech and Sandia National Labs use Pointer to develop new high-tech materials. Pointer is used as the optimization engine to perform model the fitting of experimental data and has proven to be more powerful than the neural net software it replaced.

“We are really pleased with Pointer and as such have been promoting it in conjunction with our modeling effort... I think this bodes well (for your efforts) in the car industry.” Dr. M. Horstemeyer and Dr. J. Lathrop, Senior Material Scientists, Sandia National Laboratory
Aircraft design

Pointer is used extensively by Airbus to design new configurations and do design studies. Applications include aerodynamic, structural and general configuration design.

A3XX super jumbo (photo Airbus industrie)

“The Synaps/Airbus study was the basis for a thorough and successful redesign of the A3XX” Mr. Wissel, head configuration Large Aircraft Division, Airbus Industrie (1999)

Ship design

Navatek II SWATH (photo Navatek)

Pointer is used by companies such as Lockheed-Martin and Navatek ships to design advanced ship hulls that travel at a very high speeds and have long range.

"We used Pointer successfully on a long-range, high-speed hydrofoil study. This involved using a coupled hydrodynamic and structural analysis. Considering the complex interaction among the many design variables and constraints, it would not have been possible for us to achieve the optimum vehicle design without Pointer." Dr. P. Raj and Mr. R.M. Coopersmith, Lockheed Martin Corporation

Mission and control system optimization

Pointer was used by DASA-RIA (Ariane consortium) to perform mission optimization and control gain optimization for rockets and reusable launch vehicles.

As compared with the in-house (semi-automatic trajectory optimization) method, we were able to set up our project in 40% of the usual time and find the best answer in 5% of the time. We also achieved answers that were up to 10% better and never worse. [Conclusions, report Christian Grueber, DASA-RIA, 1999]