# A Grid Framework for Computational Mechanics Applications

MICHAEL M. RESCH, NATALIA CURRLE-LINDE, UWE KÜSTER
Höchstleistungsrechenzentrum Stuttgart (HLRS)
University of Stuttgart
Nobelstrasse 19, 70569 Stuttgart
GERMANY

BENEDETTO RISIO
RECOM Services
Nobelstrasse 15, 70569 Stuttgart
GERMANY

*Abstract*
Currently, numerical simulation using automated parameter studies is a key tool in discovering functional optima in complex systems. In future, such studies of complex systems will be important for the purpose of steering simulations. One example is the optimum design and steering of high power furnaces of power plants. Grid technology makes it possible to carry out sophisticated simulations. However, the large scale of such studies requires organized support for the submission, monitoring, and termination of jobs, as well as mechanisms for the collection of results, and the dynamic generation of new parameter sets in order to intelligently approach an optimum. In this paper we present a Grid framework consisting of GriCoL (Grid Concurrent Language), which we propose as a simple and efficient language for the description of complex Grid applications, along with SEGL (Science Experimental Grid Laboratory), the problem solving environment within which GriCoL works. We apply this framework to a computational mechanics application with industrial applicability, the simulation of high power furnaces at a power plant.

*Key Words:* Grid, GriCoL, SEGL, numerical simulation, dynamic parameter studies, computational mechanics

## 1 Introduction

During the last 20 years the numerical simulation of engineering problems has become a fundamental tool for research and development. In the past, numerical simulations were limited to a few specified parameter settings because expensive computing time did not allow for more. Today, enormous computer resources, which can be provided by the Grid [1], enable the simulation of complete ranges of multi-dimensional parameter spaces in order to predict an operational optimum for a given system. We have used parameterized simulations in many disciplines. Examples are drug design by molecular dynamics, statistical crash simulation of cars, airfoil design by varying airfoil parameters, power plant simulation by varying burners and fuel quality. The mechanism proposed here offers a unified framework for such large-scale optimization problems in design and engineering. The framework levers the resources of the Grid using GriCoL (Grid Concurrent Language), a language for describing complex modeling experiments, utilized within its problem solving environment SEGL (Science Experimental Grid Laboratory).

### 1.1 Existing Tools for Parameter Investigation Studies

There are some efforts in implementing such tools e.g. Nimrod [2], Ilab [3] or SkyFlow [4]. These tools are able to generate parameter sweeps and jobs, running them in a distributed computer environment (Grid) and collecting the data. ILab also allows the calculation of multi-parametric models in independent separate tasks in a complicated workflow for multiple stages. However, none of these tools is able to perform the task dynamically by generating new parameter sets by an automated optimization strategy as is needed for handling complex parameter problems. In addition to the above mentioned environments, tools like Condor [5], UNICORE [6] or AppLeS [7] (Application-Level Scheduler) can be used to launch pre-existing parameter studies using distributed resources. These, however, give no special support for dynamic parameter studies.

Complex parameter studies can be facilitated by allowing the system to dynamically select parameter sets on the basis of previous intermediate results. This dynamic parameterization capability requires an iterative, self-steering approach. Possible strategies for the dynamic selection of parameter sets include genetic

algorithms, gradient-based searches in the parameter space, and linear and nonlinear optimization techniques. An effective tool requires support of the creation of applications of any degree of complexity, including unlimited levels of parameterization, iterative processing, data archiving, logical branching, and the synchronization of parallel branches and processes.

### 1.2 Workflow

Realistic application scenarios become increasingly complex due to the necessary support for multiphysics applications, preprocessing steps, postprocessing filters, visualization, and the iterative search in the parameter space for optimum solutions. Attempting to support this complexity has led to the development of many workflow management systems, such as Kepler [8], Triana [8], Pegasus [8]. A fundamental difference between approaches is the workflow structure. Many systems, such as Pegasus and GridFlow [8], represent their workflows as a Directed Acyclic Graph (DAG) which allows a workflow structure of sequence, parallelism and choice. But they are limited with respect to the power of the model; iteration is not part of the DAG structure. Workflow-based systems such as GSFL [9], and BPEL4WS [9] have solved these problems but are too complicated to be mastered by an average user. With these tools, even for experienced users, it is difficult to describe non-trivial workflow processes involving data and computing resources. Another fundamental difference is the use of either the data flow or control flow programming paradigm. Most existing workflow systems, including Kepler and Triana, use the data flow paradigm. This means that control flow such as iteration is difficult to design and maintain and such systems cannot carry out dynamic, self-steering iterations as in an application such as a parameter study.

The description and execution of complex experiments is not a trivial task and demands considerable effort from the creator of the experiment because existing workflow systems do not offer a sufficiently high organizational level to specify and execute complex modeling experiments. The reason for this can be explained as follows:

• Existing instruments for the description and execution of complex experiments work with specific modules tailored to a corresponding application domain and not with standard adjustable mathematical modules. Therefore, experiment-specific programs must be designed every time a new experiment is created.

• An important characteristic of a language for the description of complex experiments, which requires improvement, is the possibility to generate an executable program of maximum efficiency. This implies that the language must be organized in such a way as to ensure maximal parallelization of all processing, both between and within language elements. In other words, the system must at the inter-element layer provide possibilities of parallel execution on all branches of the experiment as well as pipelined processing of data in the nodes of the experiment program which are connected in sequence.

• A program for a complex experiment requires the description not only of the logic of the experiment but also of the control of complex data flows during the process of its execution. Practically all existing work-flow systems have only primitive mechanisms of data flow control which take little account of the dynamic structure of the Grid.

• Existing systems commonly do not use any universal instruments which provide a centralized information space (a database) for the experiment. This means these systems have problems when describing the interaction between different nodes of the experiment.

• Although workflow systems generally have graphical tools for the description of modeling experiments, these are not sophisticated enough to provide the level of precision or intuitive clarity that the user needs for the description of complex experiments.

Grid Concurrent Language (GriCoL) aims to overcome the above limitations of existing systems. GriCoL was conceived on the principle of a two-level description of an experiment program. This enables simple descriptions of the logic of execution as well as of the data flow of highly complex experiments (consisting of several hundred components). Consequently, GriCoL was envisaged to offer scientists and engineers the possibility to use all types of parallelism in an experiment program. A more detailed description of these and other important features of GriCoL is given in the second section. The third section presents the architecture of SEGL [10], the system within which GriCoL is utilized for designing and executing complex modeling experiments. The fourth section presents the realization of a Computational mechanics application, a power plant simulation by varying burners and fuel quality.

## 2 GriCoL (Grid Concurrent Language)

The GriCoL (Grid Concurrent Language) is a system for designing and executing complex modeling experiment programs in the Grid. GriCoL enables the automated creation, start and monitoring of complex experiment programs and supports its effective execution on the Grid.

The philosophy of GriCoL is based on a known fact: despite the wide variety of complex application tasks across different fields of science and technology, the set of components (e.g. gradient searches, genetic algorithms) within these tasks is very limited. These components, once they have been created in standard

form, can be repeatedly reused for modeling complex systems. If one of the desired components is not available, it is much simpler to implement this component and add it to the language than to generate a new complex application. Also, existing languages for Grid computing suffer from an insufficiently high abstraction level which makes the programming of complex experiments a difficult and painstaking job. Therefore, this paper proposes the following solution: a component-based language for describing complex modeling experiments with a sufficient level of abstraction so that the scientist does not require knowledge of the Grid or of parallel programming.

## 2.1 Common Properties

GriCoL is a universal language for programming complex computer- and data-intensive tasks without being tied to a specific application domain.

GriCoL is a graphical-based language with mixed type and is based on a component-structure model. The main elements of this language are blocks and modules, which have a defined internal structure and interact with each other through a defined set of interfaces. In addition, language elements can have structured dialog windows through which additional operators can be written into these elements. The language is of an entirely parallel nature. It can implement parallel processing of many data sets at all levels, i.e. inside simple language elements (modules); at the level of more complex language structures (blocks) and for the entire experiment.

In general, the possibility of parallel execution of operations in all nodes of the experiment program is unlimited. (It is limited only by the logic of the experiment's execution.)

In order to utilize the capacities of supercomputer applications and to enable interaction with other language elements and structures, we make use of the principle of wrapping of functionality in components. Practically all GriCoL language elements have been designed on this principle.

An additional property of the language is its extensibility through the use of components. With the help of the dialog program Units Library Assistant, it is possible to add new functional modules to the language library. Program codes, which may be generated in any language, for parallel machines (all types of an architecture and operation systems) are wrapped in the standard language capsule. This wrapping principle allows GriCoL to incorporate legacy code.

Another important property of the language is that it is multi-tiered. The multilayer principle (the sub-division into control flow and data flow) as well as the graphical context of the language makes it much easier than with a Grid workflow system for the user to understand and describe the experiment

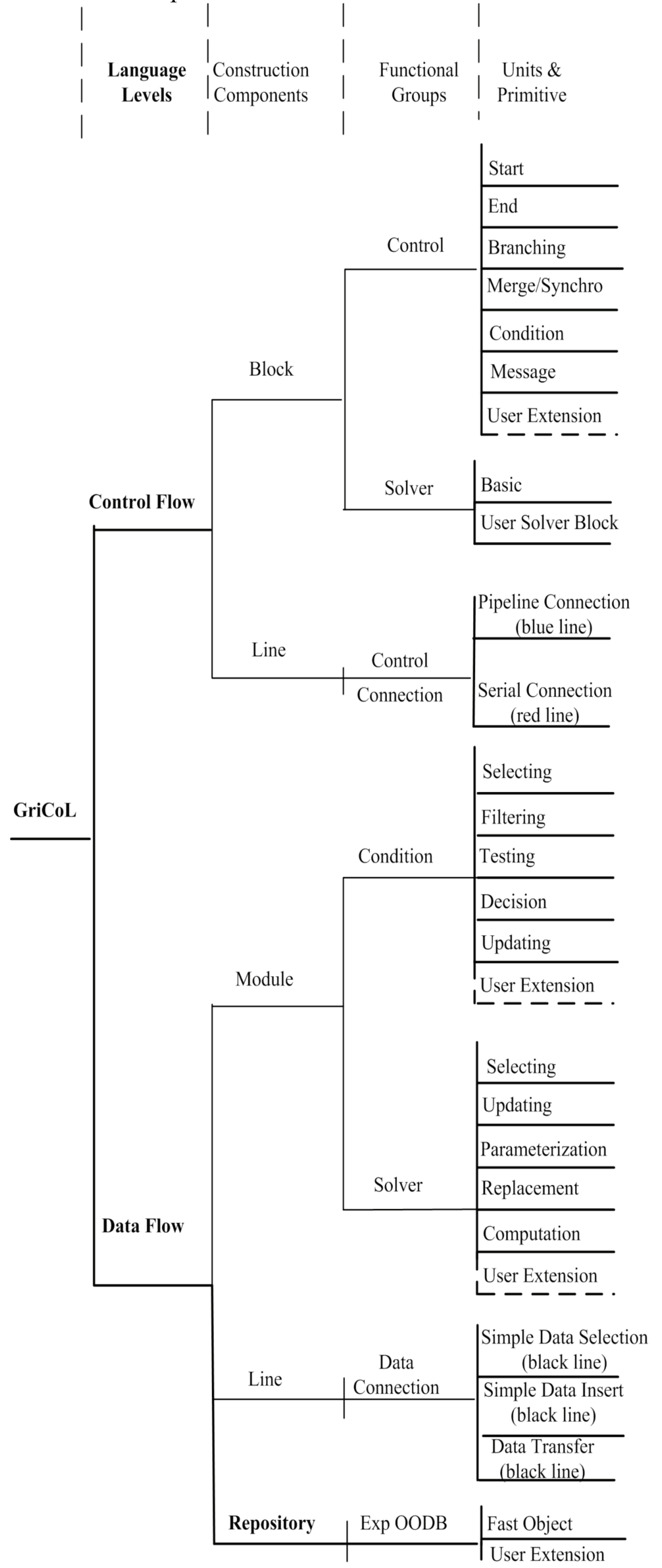


Fig. 1. Structure of GriCoL

## 2.2 Structure and Components

GriCoL combines the component model of programming with the methodology of parallel programming. The language represents a multi-tiered model of organization. This enables the user when describing the experiment to concentrate primarily on the logic of the experiment program and subsequently on the description of the individual parts of the program. Now we describe the internal organization of this language. The elements of GriCoL are shown in Figure 1.

GriCoL has a two-layer model for the description of the scientific experiment and an additional sub-layer (Repository) for the description of the database required for the creation of the experiment. The top level of the experiment program is the control flow level, which describes the logical sequence of execution. The main elements of this level are blocks: control blocks and solver blocks. A solver block is the program object which performs some complete operation. The standard example of a solver block can be a simple parameter sweep. The control block is the program object which allows the changing of the sequence of the execution according to a specified criterion. The lower level, the data flow level, provides a detailed description of components at the top level, the control flow level. The main elements of the data flow level are program modules and database sections.

The sublayer provides a common description of the database and a section for making additions to the database if necessary.

The elements of the language have graphical notation and are represented by icons (for modules and blocks) or as connection lines.

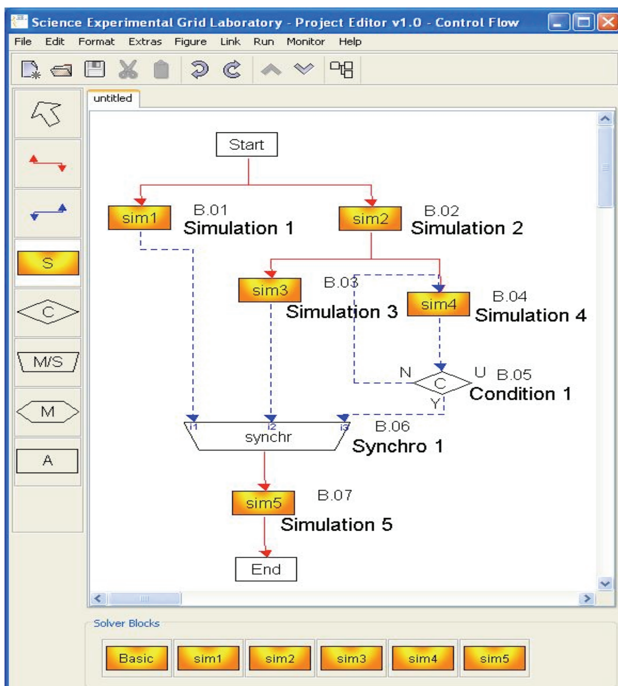


Fig. 2. Experiment Program at the Control Flow Level

### 2.2.1 Control Flow Level

Figure 2 illustrates the above mentioned for an experiment at the control flow level. As can be seen from this figure the language components make it possible to generate multilayer dynamic-control experiment programs with branches. GriCoL offers the user a complete range of control mechanisms on experiment processes: parallelization, testing of conditions and branching, synchronization and fusion, as well as exchange of messages and signals.

Solver blocks represent the nodes of data processing. Control blocks are either nodes of data analysis or nodes for the synchronization of data computation processes. They evaluate results and then choose a path for further experiment development. Another important language element on the control flow level is the connection line. Connection lines indicate the sequence of execution of blocks in the experiment and, together with control blocks, describe the logic of execution of the experiment program. There are two mechanisms of interaction between blocks which are described with the help of connection lines (either red-solid or blue-dashed). If the connection line is blue in colour, the procedure is as follows: each time the computation of an individual data set has been finished, i.e. after completion of a program run within a block, control is transferred to the next block. This process is repeated until all program runs in the block have been completed. That means a pipelined operation on the set of runs. If the connection line is red in colour, control is not passed to the next block before all runs in the previous block have been finished. That means a barrier on the set of runs.

Experiment operations always begin with the start block and finish with the end block. In the example (see Figure 2), the start block begins a parallel operation in solver blocks (notated as “sim” blocks in the figure) B.01 and B.02. After execution of B.02, processes begin in solver blocks B.03 and B.04. Each data set which has been computed in B.04 is evaluated in control block B.05. The data sets meeting the criterion are selected for further computation. These operations are repeated until all data sets from the output of B.04 have been evaluated. The data sets selected in this way (in our example these of the input of B.06) are synchronized by a merge/synchronize block with the corresponding data sets of the other inputs of B.06. The final computation takes place in solver block B.07.

### 2.2.2 Data Flow Level

A detail of programming on the data flow level is represented on Figure 3. A typical example of a solver block program is a modeling program (or a program fragment) which cyclically computes a large number of input data sets. At this level, the user can describe the manipulation of data in a very fine grained way. The solver block consists of computation (C), replacement

(R), parameterization (P) modules and a database (Exp.DB). These are connected to each other with connection lines showing the data transfer between modules and the sequence of execution during the computation process.

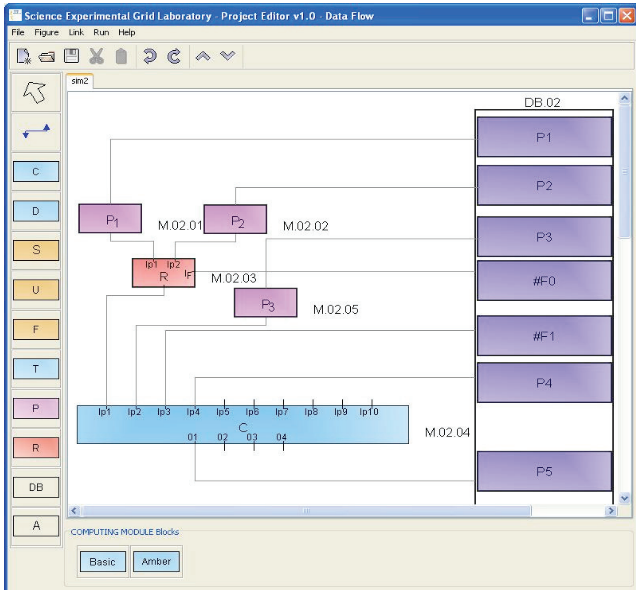


Fig. 3. Experiment Program at the Data Flow Level

Each module is a Java object, which has a standard structure and consists of several sections. For example: each computation module (C) consists of four sections. The first section organizes the preparation of input data. The second generates the job and controls its execution. The third initializes and controls the record of the result in the experiment database. The fourth section controls the execution of module operation. It also informs the main program of the block about the manipulation of certain sets of data and when execution within a block is complete.

After a block is started, the parameterization module (P) and replacement module (R) wait for the request from the corresponding inputs of the computation module (C). After that, they generate a set of input data according to rules specified by the user, either as mathematical formulae or a list of parameter values. In this example three variants of parameterization are represented:

(a) Direct transmission of the parameter values with the job. In this case, parameterization module (P3) transfers the generated parameter value to the computation module (C) upon its request. The computation module generates the job, including converting parameter values into corresponding job parameters. This method can be used if the parameterized value is a number, symbol or combination of both.

(b) Parameterized objects are large arrays of information (DB.02-P4 in Figure 3) which are kept in the experiment database. These parameters are copied directly from the experiment database to the corresponding file server and then written with the same array name with the index of the number of the stage. In this case, attributes of the job are sent to the file server as references (an array of data).

(c) If it is important, then the preparation of the data is moved outside of the main program. This allows the creation of a more universal computation module. Furthermore, it allows scaling, i.e. avoiding limitations in the size, position, type and number of the parameterized objects used in a module. In these cases the replacement module is used.

During the preparation of the next set of input data, new parameter values P1 und P2 are generated. The generated parameter set is linked with replacement processes and then delivered to the corresponding file server, where the replacement process is executed.

A typical control block program carries out an iterative analysis of the data sets from previous steps of the experiment program and selects either the direction for the further development of the experiment or examines whether the input data sets are ready for further computation, and subsequently synchronizes their further processing. An example of such a program can be seen in Figure 4.

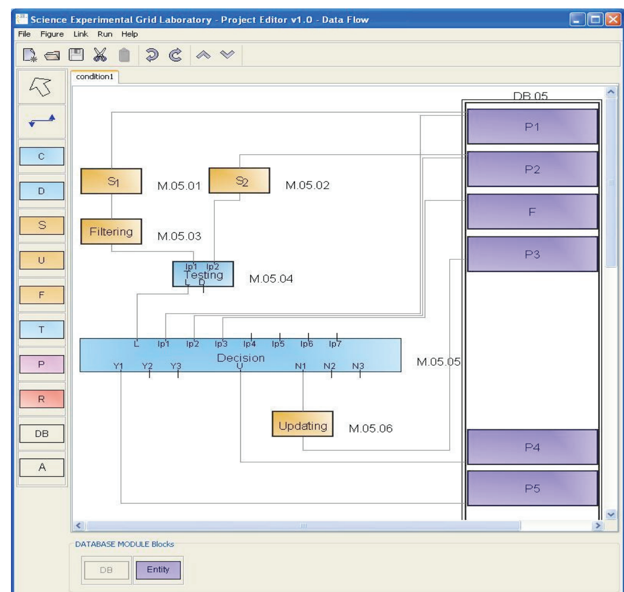


Fig. 4. Example Control Block Program (Condition Block) at the Data Flow Level

The condition block shown in this figure consists of two selection modules, a filtering, testing, decision and updating module, as well as a database. A testing module is a functional program which analyses input data sets on the basis of certain criteria. A decision module redirects data for further computation depending on the results obtained by the testing module. The selection and update modules are for the interaction between modules (computation, testing, etc.) and the database.

As we can see, the main elements are modules. In principle all simple control blocks can be mono-programs, which consist of units rather than modules, as for example the synchronization block in Figure 2. Using a small number of modules, it is possible to create a large number of different program blocks for experiments in different fields of science and engineering.

For the creation of a block program it is necessary to indicate all nodes (modules) of computation data and to show the data transfer between modules. Data transfer is indicated with the help of connection lines which connect inputs and outputs of modules. Interaction between modules of a block is based on the request and reply principle. The block contains a module which takes the role of request module (in our cases this is computation module M.02.05 in Figure 3 and decision module M.05.05 in Figure 4). The other modules of the block make replies. Requests go from bottom to top (i.e. from the generating/request module to the reply module), and replies go from top to bottom (i.e. from the reply module to the request module). In each reply module processing continues until the list of parameter inputs is exhausted.

## 3 SEGL (Science Experimental Grid Laboratory)

SEGL is a problem solving environment enabling the automated creation, start and monitoring of complex experiments and supports its effective execution on the Grid. SEGL is the environment within which the GriCoL language is utilized. Figure 5 shows the system architecture of SEGL. It consists of three main components: the User Workstation (Client), the ExpApplicationServer (Server) and the ExpDBServer (OODB). The system operates according to a Client-Server-Model in which the ApplicationServer interacts with remote target computers using a Grid Middleware Service such as UNICORE and SSH. The implementation is based on the Java 2 Platform Enterprise Edition (J2EE) specification and JBOSS Application Server. The database used is an Object Oriented Database (OODB) with a library tailored to the application domain of the experiment.

The two key parts of SEGL are: Experiment Designer (ExpDesigner), used for the design of experiments by working with elements of GriCoL, and the runtime system (ExpEngine). From the user's perspective, complex experiment scenarios are realized in Experiment Designer using GriCoL to represent the experiment. The technical mapping from this user perspective to the underlying infrastructure is carried out via the use of control flow and data flow. The control flow level is used for the description of the logical schema of the experiment. On this level the user makes a logical connection between blocks: direction, condition, and sequence of the execution of blocks. Each block can be represented as a simple parameter study. The data flow level is used for the local description of interblock computation processes. The description of processes for each block is displayed in a new window.

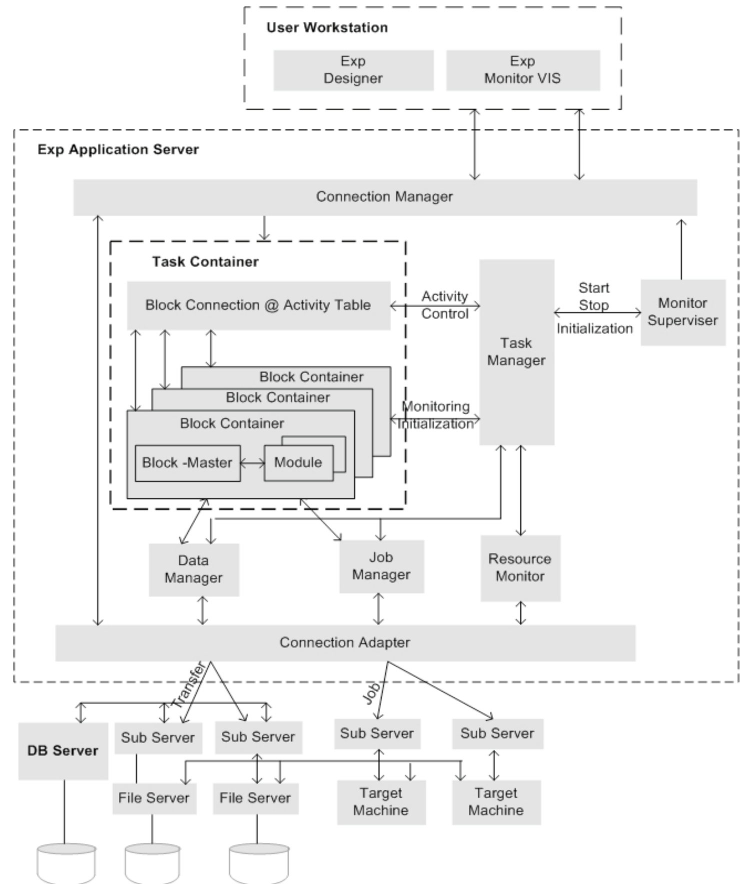


Fig. 5. Architecture of SEGL

After completion of the design of the program at the graphical icon-level, it is "compiled". During the "compiling" the following is created:

(a) The program objects (modules), which belong to the block are incorporated in Block Containers.

(b) Block Containers are incorporated in the Task Container

In addition, the Task Container also includes the Connection @ Activity Table. This table describes the sequence of execution of experiment program blocks. At the control flow level, when a new connection is made between the output of a block and the input of the next block, a new element is created in the Block Connection @ Activity Table to describe the connection.

Parallel to this, the experiment's database aggregates the data base icon objects from all blocks/windows at the

data flow level and generates QL-descriptions of the experiment's database. The container application of the experiment is transferred to the Application Server and the QL-descriptions are transferred to the server database. Here, the meta data repository is created.

The runtime system of SEGL (ExpEngine) chooses the necessary computer resources, organizes and controls the sequence of execution according to the task flow and the condition of the experiment program, monitors and steers the experiment, informing the user of the current status. This is described in more detail below.

ExpApplication Server consists of the ExpEngine, Task, the ExpMonitorSupervisor and the ResourceMonitor. The Task is the container application. The ResourceMonitor holds information about the available resources in the Grid environment. The MonitorSupervisor controls the work of the runtime system and informs the Client about the current status of the jobs and the individual processes. The ExpEngine is the controlling subsystem of the SEGL (runtime subsystem). It consists of three subsystems: the TaskManager, the JobManager and the DataManager. The TaskManager is the central dispatcher of the ExpEngine. It coordinates the work of the DataManager and the JobManager.

1. It organizes and controls the sequence of execution of the program blocks. It starts the execution of the program blocks according to the task flow and the condition of experiment program.
2. It activates a particular block according to the task flow, chooses the necessary computer resources for the execution of the program and deactivates the block when this section of the program has been executed.
3. It informs the MonitorSupervisor about the current status of the program.

The DataManager organizes data exchange between the ApplicationServer and the FileServer and between the FileServer and the ExpDBServer. Furthermore, it controls all parameterization processes of input data. The JobManager generates jobs, places them in the corresponding SubServer of the target machines. It controls the placing of jobs in the queue and observes their execution. The SubServer informs the JobManager about the status of the execution of the jobs.

The final component of the SEGL is the database server (ExpDBServer). All data which occurred during the experiment, initial and generated, are kept in the ExpDBServer. The ExpDBServer also hosts a library tailored to the application domain of the experiment. For the realization of the data base we choose an object-oriented database because its functional capabilities meet the requirements of an information repository for scientific experiments. The interaction between ApplicationServer and the Grid resources is done through a Grid adaptor. Currently, e.g. Globus and UNICORE offer these services.

## 4 Use Case: Power Plant Simulation by Varying Burners and Fuel Quality

The liberalization of the energy markets puts more and more pressure on the competitiveness of power companies all over the world. In order to maintain their competitive power it is necessary to optimize the operation of existing power plants towards minimum operation costs. Potential optimization targets can be minimization of excess air (increasing efficiency) or NOx-emission (reducing DeNOx operation costs). Pure experimental optimizations without computer-aided techniques are time-consuming and require a significant higher manpower effort. Furthermore, in the case of necessary design changes the technical risks involved with the investment decision can only be assessed with computer-aided techniques. Computer-aided methods are well accepted in the power industry.

The optimization procedure applied by the SEGL for the present problem is based on an evolutionary algorithm. In evolutionary algorithms the mechanisms of the natural evolution are applied to build an optimization algorithm. Four mainstreams of evolutionary algorithms are distinguished in literature: genetic algorithms, genetic programming, evolutionary strategies and evolutionary programming [11], [12]. In the present work a genetic algorithm (GA) is used as the core of the optimization environment, because the coding properties of GA's are ideally suited for limited parameter ranges, like those appearing in boiler optimization problems.

### 4.1 General Arrangement of the Workflow

In order to work on boiler optimization problems with the GriCoL the parameters that have to be optimized are coded in binary form and assembled to a chromosome. The chromosome carries the properties of the so called individuals. A certain number of these artificial individuals are generated initially, the so called population, and the GA of the GriCoL imitates the natural evolution process. The imitation is done by applying the genetic mechanisms selection, recombination, and mutation.. The basic workflow can be described with the following pseudo code:

1. Binary coding of optimization parameters and chromosome assembly.
2. Generation of an initial population.
3. Decoding of the chromosome information for each individual.

4. Simulation of the decoded set of optimization parameters with the 3D-furnace simulation code RECOM-AIOLOS for each individual.
5. Filtering the 3D-results of the furnace simulation to derive the derive the target values for each individual.
6. Evaluation of the performance level for each individual (Terminate the optimization process if desired optimization level is reached).
7. Selection of suitable individuals for reproduction and recombination/mutation of the chromosome information for the selected individuals to generate new individuals.
8. Return to point 3. for new individuals.

## 4.2 Industrial Applicability

An experimental operation optimization exercise performed in 1991 at a power station in Italy is used to demonstrate the capabilities of the SEGL. The unit under consideration is ENEL's coal-fired Fusina #2. The underlying firing system is a corner-fired, tangential arrangement with a windbox. In a windbox the amount of air flowing through a nozzle is controlled by the damper setting of the nozzle. A damper setting of 100% means that the flow passage of the nozzle is fully open. Reducing the damper setting of a single nozzle allows to reduce the air mass flow through the nozzle, but at the same time the air mass flows for all other nozzles in the windbox are increased. The combustion chamber has a cross section of 8 x 10 m and a height of 35 m.

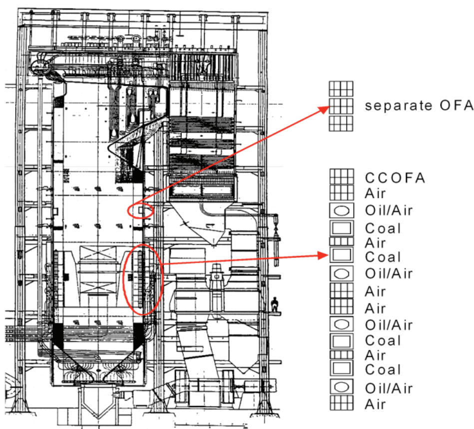


Fig. 6. Diagram of the Main Combustion Zone

In 1991 separate overfire air nozzles (separate OFA) were installed above the main combustion zone (see Figure 6) to minimize NOx-emissions. A new operation mode was required after the successful installation of the separate overfire air to maintain the lowest possible NOx-emission together with a minimum unburned carbon loss. In 1991 this optimization exercise was solved experimentally. In a series of 15 tests over a duration of approximately ten days, a number of 15 operation modes were tested with varying amounts of close coupled overfire air (CCOFA), separate OFA, and tilting angle of the separate OFA (±30°).

The following operation experience was recorded to identify an optimized operation:

a)For a horizontal orientation of the separate OFA the maximum NOx-reduction is reached with dampers 100% open.

b) A tilting of the separate OFA to –30° has a minor effect on the NOx-emission but improves the burnout (reduced unburned carbon loss).

c) A tilting of the separate OFA to +30° leads to a NOx-reduction but increases the unburned carbon loss significantly.

d) Closing the CCOFA completely at 100% open separate OFA has only a minor effect on the NOx-emission.

In order to work on this combustion optimization problem in virtual reality, a high-resolution boiler model with 1 Mio. grid points was generated.

| | NOx-emission [$mg/m^3_n$, 6 % $O_2$] | | C in Ash [%] | |
|---|---|---|---|---|
| Setting | measured | calculated | measured | calculated |
| No OFA<br>No CCOFA | 950 - 966 | 954 | 6.41 – 7.50 | 5.66 |
| No OFA<br>CCOFA: 100 % | 847 - 858 | 794 | 7.47 – 7.61 | 6.58 |
| OFA: 100 %<br>CCOFA: 100 % | 410 - 413 | 457 | 10.43 – 11.48 | 10.28 |

Table 1. Measured and Calculated (High-Resolution) NOx-Emission and C in Ash

As shown in Table 1, an accuracy of approximately ±10% between simulation and reality can be reached on the high-resolution boiler model. The optimization parameters "OFA damper setting", "CCOFA damper setting", and "Tilting Angle" were coded with 4 bit on the chromosomes. NOx-emission and C in Ash values achieved in the model were combined to a target function for the evaluation of the individuals. The underlying combined evaluation target functions are shown in Figure 7.

*Target Function = Evaluation[NOx] + Evaluation [C in Ash]*

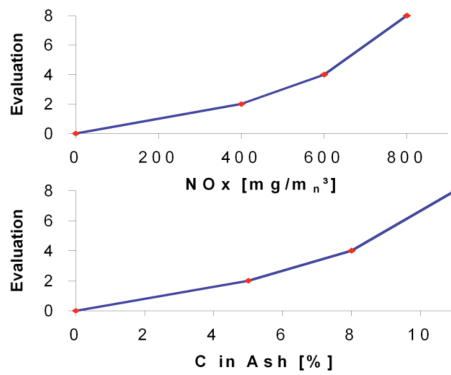


Fig. 7. Combined Evaluation Target Functions

The GA required approximately 11 generations with 10 individuals per population to identify an optimized parameter set. During the course of the automatic optimization, approximately 51 of the entire 4096 (24 · 24 · 24) coded combinations of parameter settings were evaluated with respect to the target functions.

| Generation | Target-Value | OFA [%] | CCOFA [%] | Tilting Angle [°] | NOx [$mg/m^3_n$] | C in Ash [%] |
|---|---|---|---|---|---|---|
| Basis | 12,070 | 0 | 0 | 0 | 805 | 3.39 |
| 1 | 10,061 | 100 | 100 | -30 | 479 | 10.84 |
| 5 | 9,600 | 93 | 93 | -30 | 473 | 10.42 |
| 10 | 9,177 | 93 | 20 | -30 | 458 | 10.26 |

Table 2. Development of Best Individuals in each Generation during Automatic Optimization

Table 2 shows the development of the best individuals in each generation in the course of the automatic optimization. The results demonstrate that SEGL is able to identify the same positive measures that were found in the experimental optimization. The final run on the high-resolution boiler model led to an Nox-emission of 476mg/m3 at 6% $O_2$ and a C in Ash value of 8.42%. Both values are in the range of the emission and C in Ash values that were observed in the field after the optimization exercise. The total duration of the automated optimization was only 3.5 days on a high-performance vector computer.

## 4 Conclusion

This paper presented a Grid framework composed of GriCoL, a language for describing complex modeling experiments, and the problem solving environment, SEGL, within which it is utilized to leverage the resources of the Grid. The framework can implement complex parameter studies that offer an efficient way to execute scientific experiments.

*References*

[1] Foster, I., Kesselman, C., *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.

[2] Abramson, D., Giddy, J., Kotler, L., High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?, International Parallel and Distributed Processing Symposium (IPDPS), pp. 520-528, Cancun, Mexico, May 2000.

[3] Yarrow, M., McCann, K., Biswas, R., van der Wijngaart, R.: An Advanced User Interface Approach for Complex Parameter Study Process Specification on the Information Power Grid, Proceedings of the 1st Workshop on Grid Computing (GRID 2002), Bangalore,India, December 2000.

[4] McCann, K. M., Yarrow, M., deVivo, A., Mehrotra P.: *SkyFlow*: An Environment for the Visual Specification and Execution of Scientific Workflows, GGF10 Workshop on Workflow in Grid Systems, Berlin, 2004.

[5] Thain, D., Tannenbaum, T., and Livny, M., Condor and the Grid; in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, *Grid Computing: Making The Global Infrastructure a Reality*, John Wiley, 2003.

[6] Erwin, D. (Ed.): Joint Project Report for the BMBF Project UNICORE Plus, Grant Number: 01 IR 001 A-D, Duration: January 2000 - December 2002.

[7] Casanova, H., Obertelli, G., Berman, F., Wolski, R., The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid, Proceedings of the Super Computing (SC 2002) Conference, Dallas, USA, 2002.

[8] Yu, J., Buyya, R.: A Taxonomy of Workflow Management Systems for Grid Computing, *Journal of Grid Computing*, Volume 3, Numbers 3-4, pp. 171-200, September 2005.

[9] Tony, A., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Specification: Business Process Execution Language for Web Services Version 1.1, May 05, 2003.

[10] Currle-Linde, N., Küster, U., Resch, M., Risio, B.: Science Experimental Grid Laboratory (SEGL) Dynamical Parameter Study in Distributed Systems, ParCo 2005, Malaga, Spain, 2005.

[11] Bäck, T., *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford Univ. Press, 1996.

[12] Koza, J.R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.