SEGL GriCol Manual

T. Krasikova, Y. Yudin, Y. Dorozhko

27 September 2010

Contents

1 The purposes of the language **2**

2 Levels of describing of the experiment **5**

3 How to use Dataspace, Dataset and Datasetnumber **7**

4 ControlFlow level **12**

5 DataFlow level **19**

Chapter 1

The purposes of the language

GriCoL (Grid Concurrent Language) is a language for describing complex computational experiments. GriCoL was developed as part of a SEGL system. In the SEGL system the language is used to describe models of computational experiments. The described model experiments are stored in the SEGL system and will subsequently be used to run computational experiments on HPC resources.

Described once with GriCoL language an experiment model can be reused many times. This means that, based on the model of the described experiment, users can run a lot of experiments. Model of the experiment is an abstract description of the sequence of calls to programs on the one hand and of work with data sets on the other hand. The model contains information about the sequence of programs execution during the experiment and dealing with input and output data for each program.

SEGL system allows to describe and run on HPC resources experiments, represented as WorkFlow. So, SEGL is a workflow management system in which GriCoL language is used to describe various WorkFlows.

A workflow management system is a computer system that manages and defines a series of tasks within an organisation to produce a final outcome or outcomes. Workflow Management Systems allow you to define different workflows for different types of jobs or processes. At each stage in the workflow, one HPC resource is responsible for a specific task. Once the task is complete, the workflow software ensures that the individuals responsible for the next task are notified and receive the data they need to execute their stage of the process. Workflow management systems also automate redundant tasks and ensure uncompleted tasks are followed up. A workflow management system reflects the dependencies required for the completion of each task.

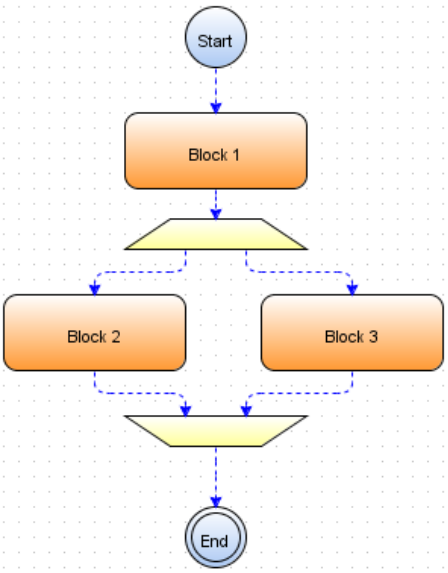

Figure 1.1: Work flow.

WorkFlow example is shown in Figure 1. The first and last elements of any WorkFlow are "Start" and "End" elements, respectively. After the "Start" element follows an "Block 1" element. This means that after the start of the WorkFlow "Block 1" will be executed. Then, in parallel (simultaneously) will be executed "Block 2" and "Block 3". After that, the execution of the WorkFlow will be completed.

So, each Block of the Workflow represents a task. In the SEGL system and in GriCoL language such a task is a program that will be runned on the HPC resource, the so-called "job". In the following chapters this will be discussed in detail.

GriCoL language supports the description of experiment models with multiple data sets. This approach allows to solve the problem of class "parameter sweep".

A parameter sweep job allows the user to specify one job description where there may be variation in one or more of the values defined in the job description. The job execution service will then expand this description into a number of jobs each of which represents one of the possible variations in the original document.

GriCoL language allows to describe the work with data of the experiment model as an abstraction, without reference to the actual data, file names, directories, etc. To do this, language has a special element - Dataspace. Also, in describing the experiment model it is not necessary to accurately specify the number of data sets, which will be used to run the experiment. For example, a user wants to run an experiment, by submitting to it 100 sets of data for which will vary the value of an option. This experiment model will look as well

as for a single data set, but in the experiment model will be described varying parameter, which will be used to start the program. In this case, the proper value for each data set will be used.

It should be noted that working with multiple sets of data in the experiment model can be described as complex dependencies. For example, originally one of the jobs of the experiment has N data sets. Such a task will be runned N times. However, the next task will be started only once, but will use all N sets of data obtained by the N previous jobs. Assume that for the first task was generated N distinct data sets.

Then, in the first task for each data set was calculated the value of a function. And then, in the next task, which will be runned only once, will be chosen one optimal function value of N values obtained in the previous step. In this case, such a task should be started only after the completion of all previous N jobs.

And all this can be described with the help of the GriCoL language.

Chapter 2

Levels of describing of the experiment

The experiment model in GriCoL language is described within two levels: the Control flow level and the Data flow level.

First, the main level, that describes the whole model of the experiment is Control flow level. At this level the process of the experiment execution is determined, it's WorkFlow. But this level does not contain descriptions of job (computer problems), the parameters to work with data of the experiment, the input data of the experiment or a particular problem, the output data, descriptions how to store intermediate data. Such descriptions are contained at the Data flow level.

Description of the experiment model at the Control flow level is a set of blocks that are interconnected unidirectional transitions - Links. Links define the order of execution of tasks in the experiment. Link can be of two types: batch and pipe. These types define the relationship between tasks performed sequentially in the case of a lot of set dataset and will be discussed below.

A set of block types in the GriCoL language is limited and will be discussed in detail below.

There is must be description of the Data flow level for certain types of blocks of Control flow level, namely:

1. Solver
2. Condition
3. Condition of cycle

Thus, the description of the experiment model in the GriCoL language always consists from one description of Control flow level and a set of descriptions for the Data flow level, which must be executed for each block of the above types.

Description of the block on Data flow level contains:

- the description of "computation" module. Computational module is a central element of Data flow level;
- Dataspaces - as variables for storing and sharing data during the experiment execution
- auxiliary modules for the parameterization of computing tasks and determining conditions of the experiment and the execution logic expressions that define the work with data.

Modules and Dataspaces on Data flow level are connected by a unidirectional transitions, called the transition. Each transition specifies the transfered data, parameters or values to start computing task.

The SEGL system contains a library of ready computation modules. This allows to describe the Data flow level by only choosing the executable module from the library and associating it by the necessary transitions with dataspaces and auxiliary modules.

However, the user can also describe a new executable and add it to the SEGL system modules library.

To describe how to work with the data and data sets themselves in abstract form, the Dataspace is used in GriCol language. Dataspace is a named area, like a variable that has a scope extending to the whole experiment. Dataspace is used to get the input data of the experiment, to save the output data, as well as to store intermediate data during execution of the experiment. Each experiment model operates with a given set of Dataspaces, whose number is usually greater than one.

Chapter 3

How to use Dataspace, Dataset and Datasetnumber

Very important concepts of GriCol language are Dataspace, Dataset and Datasetnumber. Let us examine them in detail.

The experiment, whose model is described using the GriCoL language, can be runned with several sets of data (this will be a single run of the experiment). Each set of data in terms of GriCoL language is Dataset. Thus one experiment can have multiple Datasets, whose number depends on the input data. At the stage of describing experimental model the number of Dataset is not defined and will be exactly known at the time of the experiment starting, at the stage of getting input data.

Number of Dataset for a single execution of the experiment (not to be confused with his experiment executable model) is always strictly constant. It is important to note that the GriGoL language has possibilities to describe the subexperiments. Since the subexperiment can be considered as a separate independent experiment, then it will have its own number of Dataset.

Each Dataset of the experiment during execution of the whole experiment has a fixed serial number. This is - Datasetnumber. For example, if a set of input data involves two Dataset, then the jobs of this experiment will be runned twice with Datasetnumber = 0 and Datasetnumber = 1. Numbering of Datasetnumber values in the experiment is made starting from zero. Numbering of Datasetnumber of nested experiments is own, independent from external experiments and also starts from zero.

Dataset notion is closely connected with the running of computing tasks, job. Strictly speaking, the data set needed to run a job is a Dataset with a known by system Datasetnumber. If within of a certain experiment one block will be runned N times, as the N jobs, then we can say that in this experiment, there are N Datasets.

It is possible that in the present experiment contains a subexperiment. Imagine that the number of Dataset of main experiment is N. So, subexperiment,

being the block of the main experiment will be runned N times. However, the number of Dataset in a subexperiment is M. Thus the number of runs of each block in a subexperiment throughout the main experiment will be N x M. Possible option when for each of the N runs of subexperiment, it will have its own number of Datasets, M1, M2, ... Mn.

Element "Link" on Control flow level can also affect the amount of Dataset in the experiment. If the type of a Link in the experiment model is "Batch", then all the blocks following after the "Batch" Link, will be executed only once. In this case, the execution of the first block following the "Batch" Link, will be initiated only after completion of the previous block for all the Dataset, which are defined in this experiment. This design allows to define dependencies between tasks of different Dataset. And this Datasetnumber for all running once blocks after "Batch" Link, will have a value of -1.

To give the input data in the runned experiment, to save the output data of the experiment, as well as to determine the exchange of data between the tasks of the experiment and in particular, to determine submission of input data for each task and save the output data of this task, the notion "Dataspace" is used.

Dataspace is a named area in the experiment model, which serves to store data. Set of Dataspaces of experiment is defined for the whole experiment once and is used as in the experiment, and in all nested subexperiments. Nested subexperiments do not have their own set of Dataspaces and operate the same Dataspaces, which are defined in the main experiment.

When you start the job, as input performs a set of files. In this case, at the Data flow level the user indicates that each such file is taken from a particular Dataspace. Output data of job also represent a set of files. And also each file is stored in a certain Dataspace. Since starting job done for a specific Datasetnumber, the system while maintaining data about the output files in a certain Dataspace saves also relevant information about the Datasetnumber.

When reading data from Dataspace for a given Datasetnumber the system will give a file written into this Dataspace with given Datasetnumber for the last time.

If Dataspace in a experiment model has been marked as "Input" dataspace, then the input data of the experiment should be transferred to a Dataspace before the start of the experiment. To do this, the user must select the folder which contains the relevant files and specify a file name mask. In a mask the special value %i can be used to indicate Datasetnumber in the file name. Also in the mask the * can be used to define an arbitrary set of symbols.

Consider an example: Suppose that a given folder contains the following files: file.dat, file_0.dat, file_1.dat, file_2.dat. File name mask is given as follows: file *%i.dat. Then in Dataspace will be placed files file_0.dat, file_1.dat, file_2.dat with Datasetnumber equal to 0, 1, 2, respectively. File file.dat does not match the given mask.

Directory name and file mask specified for the Input dataspace in the experiment model, but can be overridden by the user when running the experiment.

If Dataspace marked as "Output", after the completion of the experiment, for the user will be available the latest data recorded in this Dataspace for all

Datasetnumber.

It is possible to mark Dataspace as "Save intermediate data". In this case, after the completion of the experiment, the user can access all the data recorded in this Dataspace during the experiment execution.

Why do I need Dataset? Suppose we have two absolutely identical experiments, which are independent from each other and have different sets of input data. Each of these experiments, the user must start to execute and process the results. If you look at this situation from the standpoint of GriCol language, then we are working with a single experiment, which has two Dataset. This experiment will be runned automatically as many times as the many data will be given to the input. Thus the user do not need to manually start each experiment, especially if the number of runs is a lot. In this case, each experiment Dataset described by GriCol, will be executed in parallel with other Dataset, which significantly saves user's time.

Consider an example:

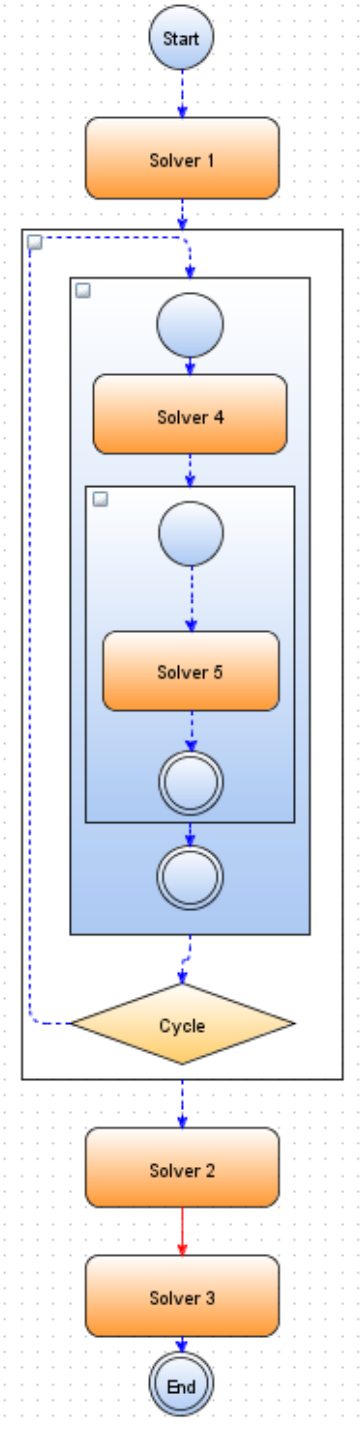

Figure 3.1: Experiment example.

In the figure we see an example of model description of the experiment in the GriCol language. This is a Control flow level. Suppose that, based on this model runs an experiment that has two Dataset. So, simultaneously will be runned two concurrent "job" for each block with different sets of data. Blue in the figure shows the Links of type "Pipe" and red as "Batch". "Pipe" is an execution of certain Dataset of experiment independent from each other. So "Solver1" will be simultaneously launched with Datasetnumber = 0 and Datasetnumber = 1, when "Solver1" with Datasetnumber = 0 or Datasetnumber = 1 is finished, "Cycle" will be started with the corresponding number Datasetnumber = 0 or Datasetnumber = 1 and etc. So Dataset of two experiments works independently of each other.

But! After the block "Solver2" we see a line of type "Batch", and this means that "Solver3" will be runned only after a completed Datasetnumber = 0 and Datasetnumber = 1 of "Solver2". And then block "Solver3" will have access to all data generated earlier in the tasks for all Dataset of experiment. After "Batch" line number of Dataset is always equal to one and gets Datasetnumber = -1.

The experiment described in the GriCol language, can have an unlimited nesting level, so experiment may contain subexperiment, which in turn also contains subexperiment etc. What we see in the picture - "Solver4" is a block of subexperiment of main experiment, and "Solver5" is a block of subexperiment even higher level of nesting.

However, there is a limited access to data at different levels of the experiment. Each block has access to only one nesting level below it's level. So "Solver2" does not have access to data generated in "Solver5", but has access to data, for example, of "Cycle", of "Solver1" and of "Solver4". To have access from "Solver2" to data of "Solver5", the user need to add a retransmission of data in block "Solver4" or in any other block on the same level as "Solver4". A retransmission of data means, that data will be copied from level "Solver5" to level "Solver4", and then these data will become available for "Solver2".

It is also important to note that after running the task data can be written in different Dataspace, and in each Dataspace can be recorded one or more files, depending on the executed script and description of Data flow level of experiment.

Chapter 4

ControlFlow level

The Control flow level describes the sequence of tasks of the experiment with blocks and directional connecting lines between the blocks (the links).

Mandatory blocks on the Control flow level are blocks "Start" and "End", and between them free model experiment is described.

Before describing the model of the experiment, let's look at what blocks exist at the Control flow level and for what purpose they are.

Blocks of Control flow level:

- **Start.** Indicates the beginning of the experiment or subexperiment.

Figure 4.1: Start block.

The block has no input and has only one output. If the user needs more than one output, the user can use Fork block:

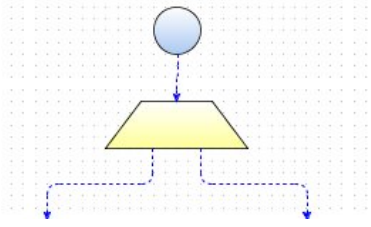

Figure 4.2: Fork block example.

- **End**. Marks the end of the experiment or subexperiment.

Figure 4.3: End block.

The block has one entrance and no exit.

- **Solver**. One of the main blocks of Control flow level, must have a description of the Data flow level. Functionality of the module depends on its Data flow level, but in fact block runs a job on resource, taking the input data and creating output data.

Figure 4.4: Solver block.

The block has one input and one output. In order to organize a branch, you can use the blocks "Fork" and "Merge".

- **Cycle**. This block allows the user to make loop. The block actually consists of two parts - the body of the loop and condition of the loop. Loop body is always a subexperiment and condition is "Condition" block.

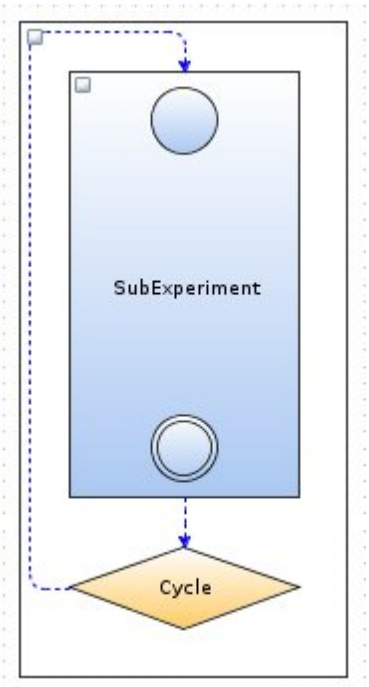

Figure 4.5: Cycle block.

In the figure SubExperiment is a body of the loop, and Cycle is a condition of the loop. In this case we see a cycle of type "do / while", but can also be a cycle of "while / do" type, then blocks SubExperiment and Cycle are swapped.

The block has one input and one output.

- **Condition**. This block allows you to organize the verification of a certain condition, which will determine the future course of the experiment for each Dataset, namely, block determines a branch of "true" or "false" to continue the experiment further. The block has one input and two outputs - "true / false", and must have a description of the Data flow level.

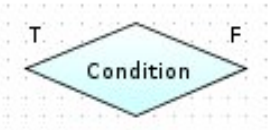

Figure 4.6: Condition block.

- **EndCondition**. This block is used in conjunction with the "Condition"

block and shows where the branching of logic ends, caused by the introduction of "Condition" before it.

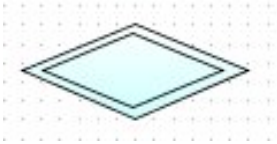

Figure 4.7: EndCondition block.

Condition and **EndCondition** blocks must be always used together. Example of using the blocks:

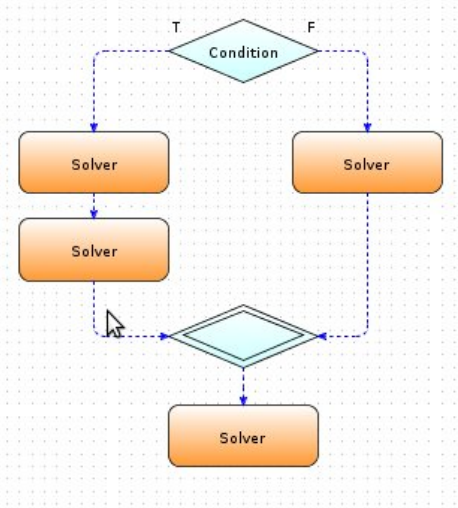

Figure 4.8: Example of Condition and EndCondition blocks.

- **Fork**. Block allows the user to enter an unlimited number of parallel branches of execution of the experiment that will be runned parallel to each other.

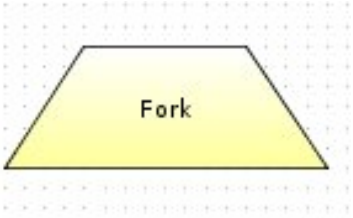

Figure 4.9: Fork block.

The block has one input and an unlimited number of outputs.

- **Merge**. Block brings together several branches of the experiment in one. So its functionality is the opposite for "Fork" block.

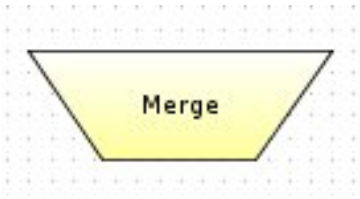

Figure 4.10: Merge block.

The block has an unlimited number of inputs and one output.

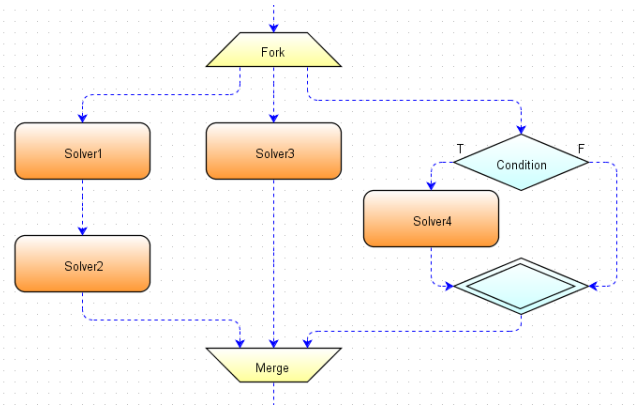

Figure 4.11: Merge and Fork blocks example.

- **SubExperiment**. As we have said above, the model of experiment can contain an unlimited number of nested subexperiments. To add a nested experiment the user should use a "SubExperiment" block, which as well as the main experiment contains "Start" and "End" blocks, and between them "SubExperiment" block has an arbitrary set of blocks. Among them including, perhaps again "SubExperiment" etc. So there is can be an unlimited nesting level of experiments.

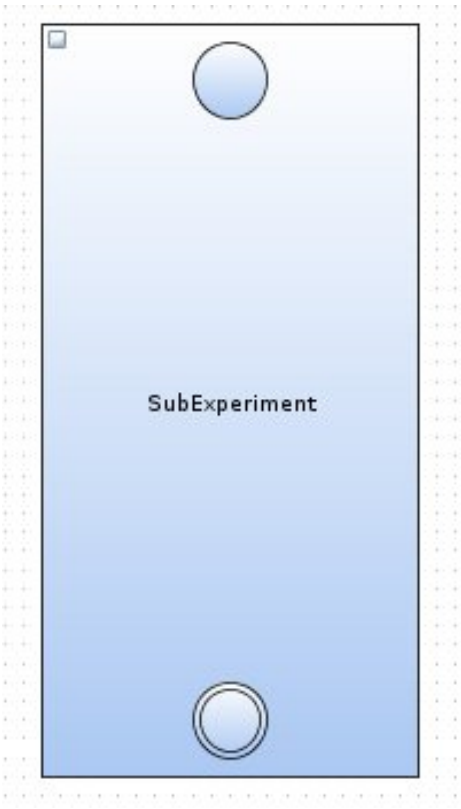

Figure 4.12: SubExperiment example.

In the figure we see a "SubExperiment", which represents the area of the experiment model and has already added the "Start" and "End" blocks.

Let's look at the example of model description of the experiment at the Control flow level using GriCol language.

Suppose that we wish to describe following algorithm with the GriCol language: first we need to run some program that receive the input user data, perform calculations and generate output files. In terms of GriCol language we are dealing with a "Solver" block.

Further, based on data generated by "Solver" runs a cycle in the body of that perform some calculations, and a new iteration of the loop only runs if we do not find the desired value. This cycle is in a GriCol language is a "Cycle" block, in which "SubExperiment" will perform some calculations, and "Condition" will check whether an exit condition of the loop. Thus, we describe an experiment in the language GriCol.

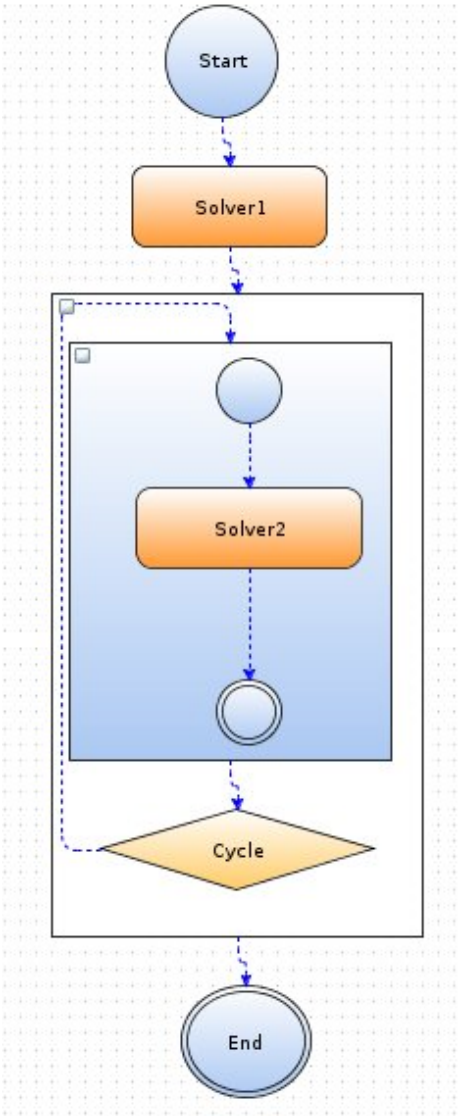

Figure 4.13: Experiment example.

Our experiment contains a "Start" block, and then will be runned the "Solver1" block, which will form the data upon which starts the "do / while" cycle. In the body of the loop "Solver2" block will perform calculations, and the "Cycle" block checks whether achieved the desired value, and if yes, the experiment will proceed to the "End" block and experiment will be completed, but if the desired value is not reached, another iteration of the loop will be startes and so on.

Chapter 5

DataFlow level

At the Data flow level the execution of computational tasks are described. When you start the computing task there are input and output data for it. As input data for computational tasks in the GriCoL language can be described files, or strings. As output data, described at the Dataflow level of GriCoL language only files can be used only files.

On the picture (Figure 3) we see an example of describing the Dataflow level in a graphical form. In this example, the computational module called "DemoComputationModule1". It has three inputs (the upper connection points for incoming transition) and three outputs (the lower connection points for outgoing transition).

Inputs and outputs an executable determine the point of connection of the module with the input and output data.

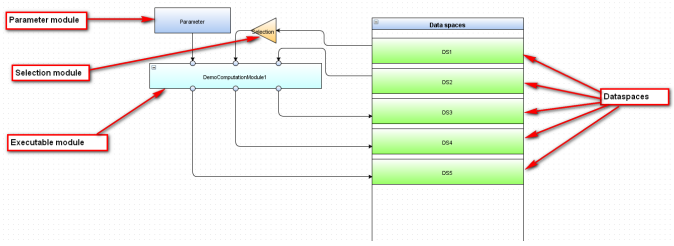

Figure 5.1: DataFlow level example.

During the execution of the experiment, the executable will be a separate job for each dataset. Job also eventually will be a shell-script, which uses names of input and output files, as well as string values as input. Each such input or output file is an input or output of computing module.

In this example, the computing module has three inputs and three outputs. Three outputs define a three files that will be as result if the job execution. Then, each of these three files will be recorded in the appropriate dataspace. A

detailed description of each input and output is contained in the description of the executable.

Three inputs in this example, define two file inputs and one input (left) for a string parameter.

To file entries may be submitted one or more files. For example, as a shell-script will be runned a program that reads in a working directory all files matching mask "*. dat". In this case, one of the inputs of the executable will be defined with a mask of input files "*. dat".

If the input of executable module is connected directly to the dataspace, it means that this input will be served the file (or set of files), which was recorded at the dataspace for the last time for the current datasetnumber value.

If you want to change the procedure for submitting files to the entrance, you must use an intermediate module selection in the connection between the dataspace and the input of the executable.

In the selection module must be defined script (using JavaScript language), which in terms of data of all files stored in the dataspace for all datasetnumber, can make the sample of files needed to feed to the input of the executable.

Using the module "parameter" allows us to describe the expression, a function that returns a string value, and use as the variable the datasetnumber value.