

# Data Flow Processing, eventBased Algorithms and Data Main philosophy

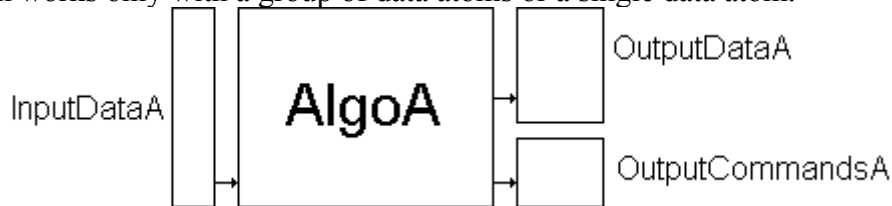
Garo Garabedian  
Independent researcher,  
Student at Technical  
University- Sofia, Bulgaria  
[garabedian@gmail.com](mailto:garabedian@gmail.com)

*Everything around us is data flow architecture and we all react on events about changes on which we have declared our interest. I have tried to understand and find how to optimize our and computers' reactions in order to save time and work.*

Humans' work can be generalized into three main steps which are repeated periodically: collecting data, applying policy/ algorithm (taking decision), producing output data/ commands. When an algorithm or data is changed the whole process of work have to start again (reapplication) in order to apply (react on) the change.

Man works a lot on real-life problems, he takes many atomic input data, applies many times one algorithm and produces many output data or commands or both. When the algorithm is changed the man reapplies the updated algorithm on all input data, but only where the changed part of algorithm is used in the computation of these concrete input data (it is supposed to produce a different final result than when the algorithm wasn't updated). When some input data is changed the man reapplies the algorithm on the changed data, but only this part of algorithm which processes the data which has been changed.

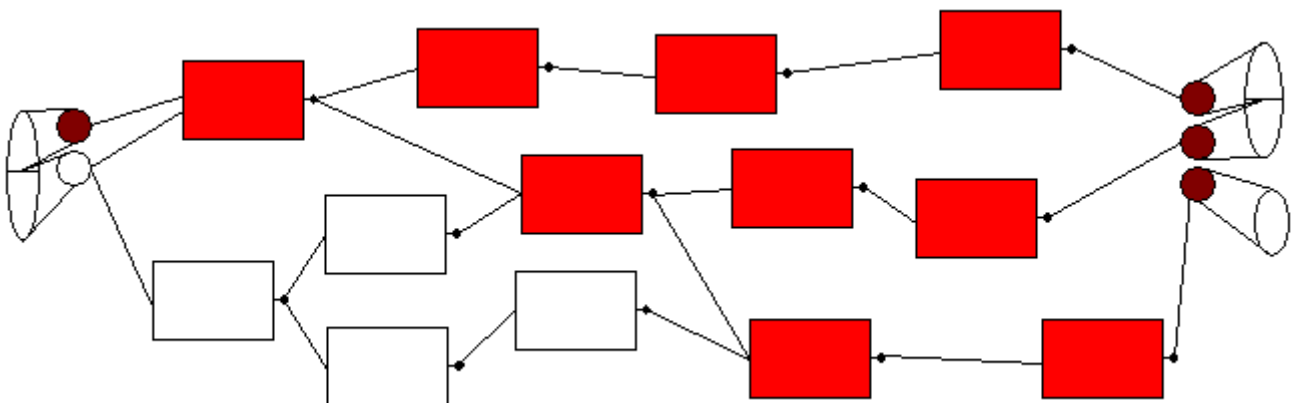
I offer to declare every algorithm with its computational atoms and every data with its structural atoms. In order to be checked after every step (computational atom) is there a difference between the result of the past computation and the changed (input data, algorithm or both) one. Algorithm atom works only with a group of data atoms or a single data atom.



*Diagram 1: The atom of the work, input data, algorithm, output data.*

Note: Everything (said here) applicable to data is applicable to commands, too.

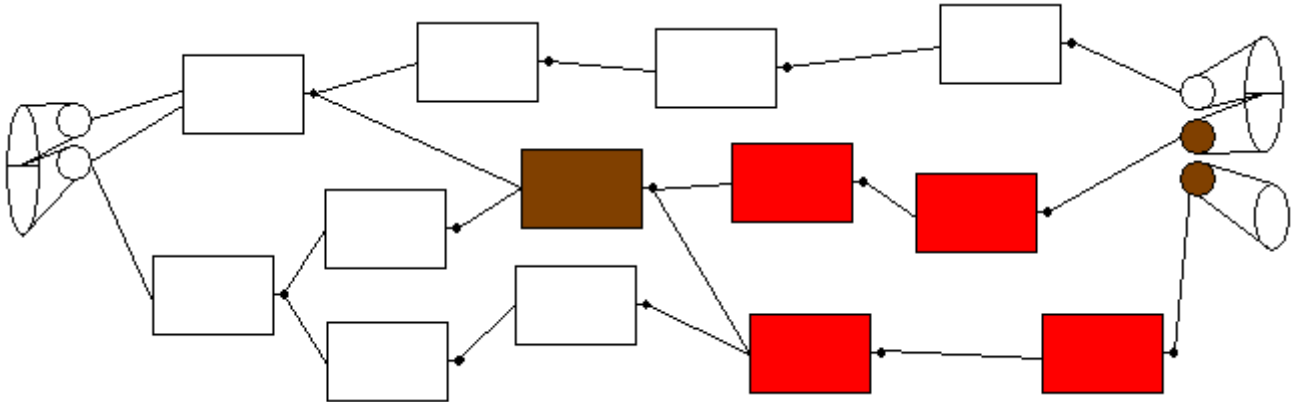
Changing of the input data (marked in brown) will produce need of additional rework (marked in red) and possible affection of the output (affected is marked in brown). The steps of additional rework are executed according to the main algorithm:



*Diagram 2: Process of rework when one input data is changed.*

Changing of the group of algorithms (marked in brown) will produce need of additional rework

(marked in red) and possible affection of the output (marked in brown). The steps of additional rework are according to the main algorithm. The changed algorithm (in brown) is recomputed first:



*Diagram 3: Process of a work and how it have to be reworked when one algorithm is changed.*

In practice we can't make atoms enough smaller in order to produce different output when presented with different inputs, even if doing so there are cases when for many different input data the result is the same. In efficiency reasons, check after every step is there a difference between the past computation and the changed (input data, algorithm or both) one.

### **Computer programs, computers applying algorithms**

In computer perspective the machine must be able to save the all input and output data. In this reason, the application must be written on enough abstract platform which enables sucking out (and future injection) of the all amount of input and output data and commands.

In developers' perspective, find and declare as much atomic parts of the algorithms in order to technologically enable with a big percent of efficiency the searching and recomputing the only necessary input data when an update is applied.

### **Content editors, user works on data through computer applications**

Applying it to a human work, we can't recognize the all variations of the algorithm which people apply. But when somebody changes something we can inform him about the declared piece(s) of data which according to the declaration(s) is relevant in a described or not described path to the changed piece.

Data in the human work is connected by theirs relevance. Enabling technological storing of this relations can secure the user from making mistakes when changing some pieces of a big document. Without reading the whole document and trying to find the relations between the piece which he wants to edit and the rest parts of the document, the user can see the all relevant parts as they are declared in the system. One data atom can contain pieces of text which are parts of other data atoms.

There is no limitation a piece of data to be relevant to many pieces in different meanings

Because people apply different algorithms on content and this algorithms' count is not finite and because the relations between pieces is a function of the applied algorithm. The user will not be 100% secure of not changing content and in the view of the all document making it nonsense.

While the user is editing, every related piece of content to the edited piece have to be presented to him.

It is useful to enable the user to declare abstract algorithm atoms and model the process of work on one content, the algorithm atoms in their connections between themselves and with the data atoms.