Algorithmic Networks and Object-Oriented Programming

Vladimir Marley¹, Dmitrii Vasilenko¹, Lubov Tyndykar⁷, Sergey Plotnikov¹, and Oleg Korolev²

¹Admiral Makarov State University of Maritime and Inland Shipping, Drinskaya st., 5/7, Saint-Petersburg, 198035, Russia
²St. Petersburg Institute of Informatics and Automation Russian Academy of Sciences, Line 14, 39, Saint-Petersburg, 199178, Russia

Abstract. The article considers relevance of using the formalism of algorithmic networks in terms of object-oriented programming and the development of the basic ideology of using the formalism of algorithmic networks to solve problems in the framework of object-oriented systems is considered. A solution is proposed to create a network of connected objects of a certain structure with the formalism of algorithmic networks and its algorithmic completeness. The choice of the platform for the implementation of the required task on the software part was made, providing full support for the object-oriented paradigm and allowing the use of the event mechanism and the delegate mechanism. The main classes of the system are presented, and the interaction between them is discussed in detail in the execution diagram. The process of setting an algorithmic network and its calculation is considered. The ability to set algorithmic networks programatically is implemented, which further allows you to create the required configuration of the algorithmic network and save it as files. The calculation process is considered.

1 Introduction

Algorithmic networks have not lost their appeal to users, which is due to the convenience of applying the AN formalism to solve problems that use recursive computation, reinforced by the continuous improvement of AN formalism-based modelling techniques and tools [1-12]. Therefore, the issue of using the AN formalism in terms of object-oriented programming (OOP) seems extremely relevant. The reason for this is that programmers can have problems when planning the calculation of recursive tasks [13-14]. Which obviously can be classified as a barrier between the user and the computer. It should be noted that in this case we cannot limit ourselves to creating a software package that allows modelling and computation of problems to be solved.

* Corresponding author: tyndykarl@gumr.ru
2 Materials and methods

It is necessary to develop a basic ideology of using AN formalism to solve problems within object-oriented systems and to create a basic class library reflecting the proposed approach and facilitating the programmer's work. As such a solution, it is proposed to create a network of related objects of a certain structure (type). These objects can be divided into two main subclasses: computable nodes and delay nodes. Delay nodes are singled out into a separate subclass because of task specifics. All calculations are organized by means of the CommandNode class that receives an instance of an operation and a sequence of nodes whose values this operation should be applied to to obtain the required result at each iteration. Instances of the CommandNode class can be reused as data nodes for other computable nodes provided no closed loop is obtained. Delay nodes (TimerNode) are used to form closed loops.

The network computation continues while all nodes are in the Active or Waiting state. If at least one of the nodes detects that the input data set is completely exhausted, the AN calculation process stops.

![Diagram](image)

Fig. 1. Basic classes.

The Microsoft.Net platform was chosen as the implementation platform for the required task. C# was chosen as the implementation language. The choice is explained by the fact that nowadays Microsoft.Net fully supports the object-oriented paradigm and in addition to that it allows using the events mechanism (events) and delegates mechanism (delegates).
Using them enabled to greatly reduce the code's size and complexity. All in all it has had a positive impact on the system's implementation. We should notice that events and delegates are not something new for the object-oriented paradigm. It's just a simplification for commonly used templates such as Observer and Command. Thus, with only minor formal changes, the proposed approach can be transferred to other object-oriented programming languages such as: Java, C++, etc.

An algorithmic network is constructed dynamically by linking network nodes into a specialised graph. All nodes of the network must be part of a hierarchy of nodes, the apex of which is an abstract class Node, which defines the basic logic for changing states and the order in which values are calculated. Synchronization of computation and delays is entrusted to the NetworkManager class. An instance of NetworkManager class with a set of attached nodes can be regarded as an algorithmic network. This is because the basic logic of computing an algorithmic network is not defined in instances of classes, but in the dependency structure (interconnection) of individual nodes with each other. Fig. 1 shows the main classes of the system, and the interaction between them is detailed in the execution (sequence) diagram - Fig. 2.

![Diagram of the algorithmic network](image_url)

*Fig. 2. Interrelation of the AN basic components.*
Next, consider the process of setting up an algorithmic network and calculating it in more detail. The first instance of the algorithmic network manager is created. Next, instances of the network nodes are created. The nodes are created sequentially and joined together immediately. The existence of a large variety of node types allows the creation of many algorithmic networks by simply combining them. At the moment, it is possible to define algorithmic networks programmatically, but in the future there is no limitation to create a graphical environment that allows creating the required configuration of algorithmic networks and saving them as files.

Once the nodes are created and the links between them are fully defined, they are registered in the algorithmic network manager. During registration, the PrepareEvent and CalculateEvent event handlers of the network manager are initialized. These events are used to synchronize the computation of the network nodes, at each iteration.

The PrepareEvent event is used to synchronize all nodes to the Waiting state. The CalculateEvent event causes cascading of the network nodes.

The sequence diagram in Figure 2 illustrates the process of assigning and calculating an algorithmic network.

Theoretically, it is possible to allow nodes to be created and linked together in any order. The only restriction is that the nodes must be created before any links between them can be set up. In practice, however, creating nodes and linking them in the order in which they appear in the diagram makes it much easier to specify the network and find errors in it.

3 Node class

The Node class is the base class for all network nodes. This class defines the basic logic for reacting to events initiated by the network manager and the basic functionality for calculating node values. Event handlers for events initiated by the network manager are added in the Assign function, which is added when a node is added to the network. Therefore, each node added to the network will respond to PrepareEvent and CalculateEvent events. Since everyone has the logic to respond to these events, it was reasonable to define it within the base class Node.

4 NetworkManager

The NetworkManager class is designed to manage network nodes and synchronize calculations. The main function of the NetworkManager class is the Run function which starts the calculation process of the algorithmic network. AC calculation is reduced to the serial execution of StartStep functions, until the whole network is switched to the state of completion. The logic of the NetworkManager class is shown in Figure 2. From which you can see that the main task of the StartStep function is to initiate all the delayed nodes (NextStep call) and activate two events PrepareEvent and CalculateEvent by calling the OnPrepare function and OnCalculate function respectively. The Wait function allows the network manager to wait until all nodes have calculated their values and can move on to the next iteration.

5 TimerNode

The TimerNode is designed to move the network to the next iteration of the calculation. The delay node is associated with two nodes. The delay node monitors the values of the input node (inputNode). When the state of the monitored node becomes active, the delay
node also becomes active. And when it moves to the next iteration step, the NextStep function is called, which transfers the value of the input node to the output node.

6 Results and discussion

Next, we need to take a closer look at the process of calculating a node's value. After all, in the active state, a node's value can be obtained by accessing the Value property. If this is the first reference within an iteration, the node value is not yet known. Therefore, a virtual (abstract) Calculate function can be invoked to calculate this value. When control is passed to the descendant class, the class instance remains the same. The logic defined in the virtual functions changes. The logic of this function must be overridden in each inherited class. The resulting value is stored in an internal variable and when the Value property is called again, the previously stored value will be returned.

![Class hierarchy diagram](image.png)

Fig. 3. Class hierarchy diagram.
7 Conclusion

Algorithmic network calculations have been organised, the basic logic for changing states of the algorithmic network has been defined, and the required configuration of the algorithmic network has been created. A complete diagram of the hierarchy is shown in Figure 3. Algorithmic networks, in their "classical form" with focus on the implementation of algorithmic models based on functional programming have been observed. Algorithmic networks with links in nodes to other algorithmic networks are equal to the class of structural algorithms and allow to realize distributed models rather effectively, but it is difficult to make changes in distributed ones. The object-oriented approach simplifies and localizes changes in distributed algorithmic models.

References