УДК: 004.415(043.2)

Rasul Falaki, Arash Ayati
(Research institute of Green Petrol Ltd, Ukraine),
Sergii Kotla, Assistant Professor
(National University of Ukraine)

OBJECT ORIENTED PROGRAMMING ARCHITECTURE FOR NEURAL NETWORK MODULES ABSTRACT ARCHITECTURE FOR ROBOT SOFTWARE REALIZATION

An abstracted design of intelligent software constructed of NN modules is investigated. The root of the design is in general three-tire architecture that separates presentation from data and processing model as in MVC [4]. Having NN module designed at hand, the question of the placement of such modules in the architecture of the whole system is faced. Designing software systems that hold the famous principles of Object Oriented Programming known as encapsulation, polymorphism and inheritance, is crucial even for the simplest tasks. The goal of authors is to come to an abstract architecture that lets the programmer to place the NN module in it and have the communication between it and all other parts of the code organized so that to hold the OOP principles realized.

Introduction. Neural Network programming is the microscopic-induction approach of soft computing in the body of Artificial Intelligence [1], directly inspired from biological systems.

The NN programming involves same general steps of requirement analysis, design, implementation, process studying and deployment, together known as 4+1 view [5]. A process based on these views usually ends up with architecture over the whole system that maintains the OOA/D principles in the system evolution. The investigation, discovery, reuse and hybrid combination of such architectural models has been a known practice in software engineering and is taking place also for NN software development. As an example, Dart, Senyar and Sterling [2] present a unified framework in the form of an extension to the capability maturity model [3] which is known as the neural network process model (NNPM). Authors provide the result of investigating an effective behavioral based abstract architecture for robotic software development.

Neural Network Mathematical considerations

Incoming signal of the robot data model are depicted to the formula (1)

$$y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + ... + \omega_n x_n \quad (1)$$

$$y = \sigma(\omega_0 + \omega_1 x_1 + \omega_2 x_2 + ... + \omega_n x_n) \quad (2)$$

Our activation sigma function defined as (2). For the purpose of demonstration in this article we use the sigmoid function as our activation which has together with derivative is defined:

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad (3)$$

$$\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2} \quad (4)$$

The graph of them:

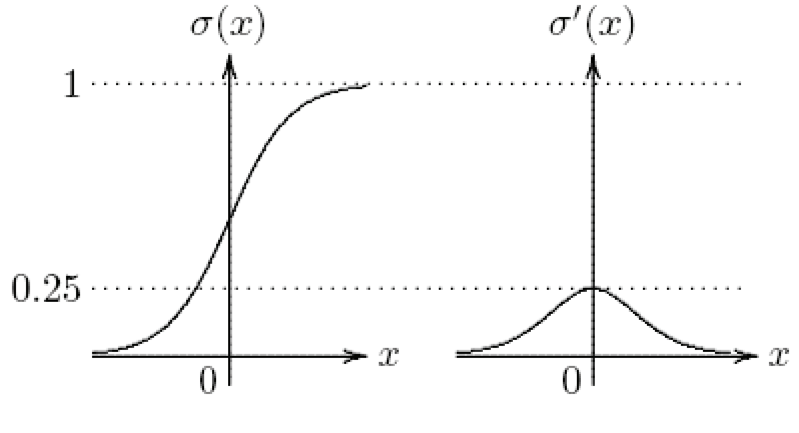

Fig. 1, Graph of activation functions

The general NN model will be as:

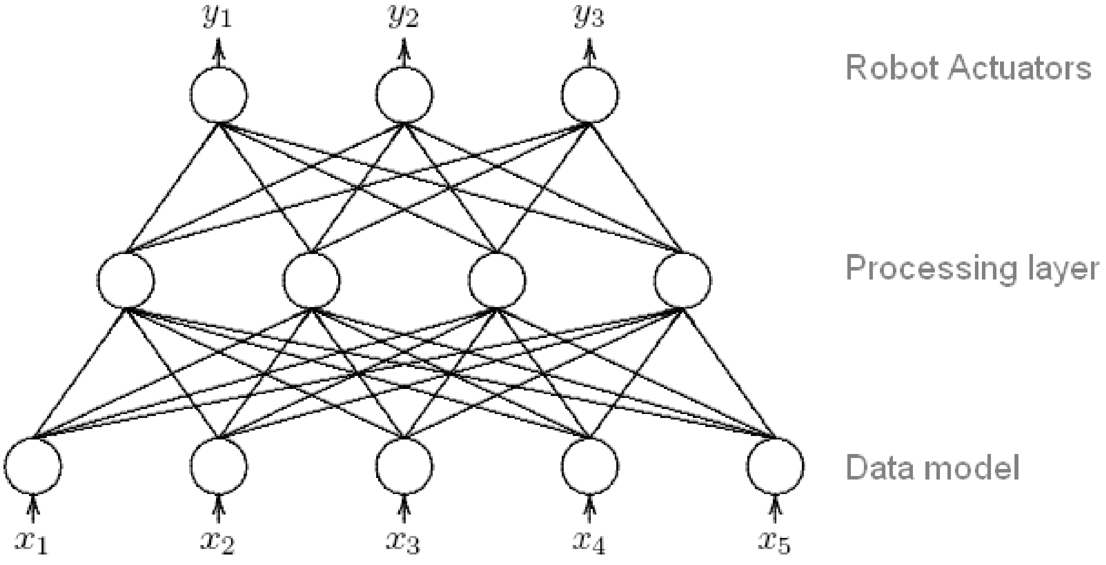

Fig. 2, The NN layer organization

By making some assumption for the purpose of demonstration, we will have n neurons in the data-model, m neurons for robot actuators and p neurons in our processing layer. The output of processing layer neurons will defined (5)

$$x_i^1 = \sigma(\sum_{j=1}^{n} \omega_{ij}^0 x_j + \omega_{i0}^0), i = 1,2,...,p \quad (5)$$

$$y_i = \sum_{j=1}^{p} \omega_{ij}^1 x_j^1 + \omega_{i0}^1), i = 1,2,...,m \quad (6)$$

After processing layer our output will be (6). This is a general setup of NN that makes the stage ready for deploying any model of function approximation. We do not further develop concrete models here as we are dealing with a general abstract architecture of the network. The above configuration is enough for us to depict the decomposition of NN parts in the software architecture.

Embedding NN in architecture with object oriented design

In the architectural model that we present, the famous proxy design pattern [5] is realized. The NN is assumed to be processed on a software server and it also could be deployed on a physically different machine than the robot. An example is when the brain machine is over the internet.

In this way, the world gestures stimulate the robot actuators and this input to the robot mind perception activates events. Each event is connected to the robot command system through the controller. In this way, controller is like a junction that connects events and commands. Events and commands have their own separated class which pulls in the principles of Encapsulation, polymorphism and inheritance. Controller has its own class. The data-model class is a singleton [5] and it is updated by NN via proxy pattern. The command is called by controller based on the event that happens and there are three basis methods designed in the command class: execute, result and fault. In the execute function, the command calls the proxy service related to the event, and just with the same name as it is on the server. This is performed by calling one of the delegate class methods. Delegate class is a proxy between the NN server and other part of robot mind model. The delegate encapsulates the connection with NN server part, being it physically placed in the robot or out of it. NN performs the service, i.e. the actual processing and sends the data back to the delegate. The data flow continues from the delegate back to the command function. If the data back is error, then the fault function of the command object is called, otherwise the result function is called and the data is passed to it. The result function in turn may update the data-model which in turn may fire needed events that manipulate actuators. Actuators may be directly a function of data-model updates so that the actuator manipulation takes place automatically without the need of extra events. Delegate class may get configuration files from the xml files as depicted on the following picture:

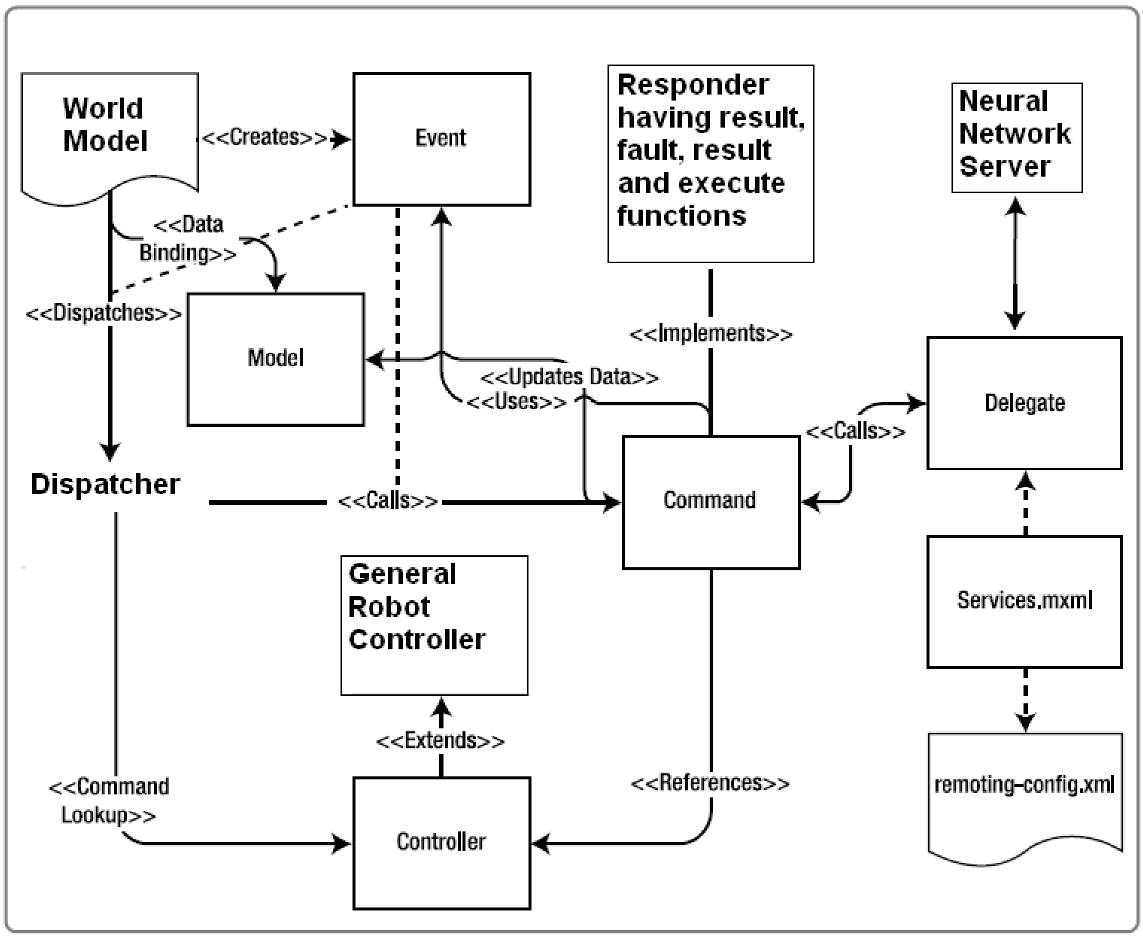

Fig. 3, The architecture for OOP of NN

Implementation:

Authors implemented the architecture on a robot based on the OpenWRT Linux Kernel having two motors and two bumper switch sensors. If sensors output gives (+1, +1), which means both are pressed, then the motors should not work so the output of NN have to be (-1,-1). Logic matrix of this information is (7).

$$\begin{matrix} +1 & +1 & -1 & -1 \\ +1 & -1 & +1 & -1 \\ -1 & +1 & -1 & +1 \\ -1 & -1 & +1 & +1 \end{matrix} \quad (7) \qquad \begin{matrix} +1 & +1 & -1 & -1 \\ +1 & +1 & -1 & -1 \\ -1 & -1 & +1 & +1 \\ -1 & -1 & +1 & +1 \end{matrix} \quad (8)$$

Each vector of the above matrix is a pathway of the NN. If NN works, for a data model of (+1, +1) we must get (-1,-1), which is the first column of the above matrix. By multiplication of it to itself we get (8).

Having this in the NN, we can ask the question (+1,+1) and the NN by using above matrix and multiplying it to (+1,+1) will give us the answer which turns to be a column vector as (+2,+2,-2,-2). In normalization process, results larger than zero change to +1 and less than zero change to -1 and the resulting vector turns to be (+1,+1,-1,-1), extracting the answer from it we get (-1,-1) which is the right answer. So NN worked the right way.

Conclusion

The NN module of the last part has been successfully tested on the architecture above realized on the target robot. The architecture lets the programmer to hold the true principles of Object

Oriented programming. Every task has its own class so it is performed **encapsulated** in own object. Next, for instance every new command inherits the base command class holding **inheritance** principle. Finally, as an example, every command implements three main functions named as execute, result and fault, no matter what the command is going to perform. This is the **polymorphism** implemented in the architecture. That means that the goal of the project has been achieved. The further material including video of the robot that is realized with NN based on this article could be found on Robotic sub club of ZabavaClub.com.

References

1. *Toshinori Munakata, Fundamentals of the new Artificial Intelligence*, Springer-Verlag London Limited 2008, ISBN: 978-1-84628-838-8.

2. *P. Dart, A. Senyard, and L. Sterling. Towards the Software Engineering of Neural Networks: a Maturity Model*, In proceedings of the 2000 Australian Software Engineering Conference, page 45, 2000.

3. *W.S. Humphrey, Managing the software Process*. Addison-Wesley, 1989.

4. *Glenn E. Krasner and Stephen T. Pope*. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. Journal of Object-Oriented Programming, 1(3):26–49, August/September 1988.

5. *James Rumbaugh, Ivar Jacobson, Grady Booch*, The unified modeling language reference manual, Addison Wesley Longman, Inc., ISBN 0-201-30998-X

6. *Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides*, Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley Pub Co; 1st edition (January 15, 1995).