Научна конференция „Иновационни софтуерни инструменти и технологии с приложения в научни изследвания по математика, информатика и педагогика на обучението“, 23-24 ноември 2017 г., Пампорово
Scientific Conference „Innovative Software Tools and Technologies with Applications in Research in Mathematics, Informatics and Pedagogy of Education”, 23-24 November 2017, Pamporovo, Bulgaria

NATURAL-ORIENTED PROGRAMMING

Stoyan Cheresharov

Abstract. This paper describes a model for software architecture, called “Natural-Oriented Programming” and its four principles. The existing paradigms of the object-oriented, aspect-oriented, event-driven, domain-driven, test-driven development, design patterns and other research does not fully answer the question how the entire software architecture of a system should look like. We propose a generalized model for software architecture and its four principles. They include the principles of communication, unit, time and space. It is a new paradigm and way of thinking. We have built prototype software systems as a proof of a concept, using the model. Each prototype is sound and stable. The model allows the development of enterprise applications to become standard and trivial process instead of craft and art. The wide adoption of models like this could allow the industrialization of the software development.

Keywords: Software Architecture, Object-Oriented Programming, Aspect-Oriented Programming, Event-Driven Development, Design Patterns

1. Introduction

Building a Complex Software Application is a difficult task. The Design Patterns [1-3], Best Practices [4-6], give only common ideas of how a system should be built. “Strong cohesion” [7-9], “Loose coupling” [10-12], “Refactoring” [13, 14], the Principles of the Object [15, 16] and Aspect [17, 18] Oriented Programming have proven their advantages in the software development. But the main question “How to follow them and build an application in a consistent way?” remains. The problem is that the given general directions still keep the process of creating software systems in the state of an art and craft. The main limitation is that no clear steps are defined.

Every system has its own architecture principles, ideas and philosophy. This makes the process of development and maintenance very difficult and time consuming. The developers have to go through the slow process of knowledge transfer for each system. We need models that allow industrialization of the software development. Workers with no skills and experience should be able to participate. Solving this problem is interesting and important because the software becomes a crucial part of the human society. We are talking about AI, IoT, 3D printing, Single Board Computers, Serverless Architectures, Augmented Reality etc. more and more quality software have to be build. Finding a way to produce software in the industrial scale is important. This study extends and augments our knowledge and principles. It gives a different perspective and way of thinking for solving common well known problems, by modeling the world in our dimension. A parallel is made between the natural world and the software systems.

The proposed principles and model are used for building an Expert System for the Education [19]. But these principles are more fundamental and can be applied for any kind of a system.

Building an Expert System for example, is in fact building an Artificial Intelligence [20]. But the Intelligence is the highest form of existence. How do you build intelligence? Some approaches already have been thoroughly studied in many papers [21]. They cover different aspects of the Expert Systems such as the decision making algorithms [22], Neuro-Fuzzy Expert System [23]. We want to move the focus on the fundamental architecture of the system. This is an innovative approach, inspired by the fundamental principles of the creation. The only intelligent life we know so far has been built by the nature. Artificial Intelligence has not been built completely by the human beings so far. So, we have to follow the steps of the nature.

Building an application is like solving a party chess. You cannot predict the entire game, but what you can do instead, is to increase your chances to win, by making moves that allow you maximum flexibility and taking the best positions. By following the nature we are doing exactly this.

Let’s identify some requirements of the system.

The system has to be highly decoupled, modular, allowing adding and removing modules without affecting the rest. The modules have to be reusable. Our goal is to offer maximum flexibility, detachment, strong cohesion and low cost of change.

The system should be Web/Cloud based and to offer Application Programming Interface [24] based on “RESTful” and “RPC” Web Services [25], allowing mobile devices connectivity. It is obvious, that this is a client server application, which consists of many elements more or less independent. Such system can grow infinitely.

Let’s take a look some modern tendencies.

The entire Internet can be seen as a supercomputer [26] offering endless amount of APIs. The operating system for example has only limited amount of APIs which are used mostly to drive the hardware. Internet is the best knowledge repository. An application that can communicate with the Internet API’s has almost unlimited knowledge base. Such application can run on mobile devices.

The web turns into a web of services. The last tendency in Internet is converting the World Wide Web into a Web of Services, Internet of Things [27]. We want to shift the responsibility of interpreting the data and converting it into information to the machines. Right now the machines are doing the easiest part of the process, they just keep and serve the data. This situation is rapidly changing with the newest HTML5, ES5, CSS3 standards and tendencies. The heavy lifting, analyzing and interpretation of the information can be done by the machines. They don’t only keep data, but become more and more active part of turning this data into information. The data turns into information when meaning is add to it. We want to follow this tendency.

If the Artificial Intelligence is to be born, there is a high probability to be born in Internet or at least to have very tight connections with it [28].

To meet the described above requirements and tendencies of the modern systems, the proposal we make is to copy the principles of the nature. The main goal is to find a consistent way of building applications, model of a software architecture, which gives clear defined structural elements and step by step instructions. Simple and easy to understand architecture, which will allow developers without skills and experience to participate in the project. The developers will benefit from the known structure and less knowledge transfer. The main idea is to copy the nature in order to achieve the goal. It is a new paradigm and way of thinking not a framework. We are proposing the idea that every application should have clearly distinct and recognisable elements (systems): Communication, Unit, Space and Time. As a bare minimum we should have at least a communication system. A group of elements with a common Communication System form an unit. Having a notion of Time and Space in the unit is optional.

2. Identifying the main principles (methods) of the model

The research is based on the inspiring paradigms and methods of Object-oriented, Aspect-oriented Programming, Event-driven, Domain-driven, Test-driven development, Design patterns etc.

The main requirements of the model are:

- The main requirement is to be simple to understand and use.
- The model should be inspired by nature and use very well know terms and concepts.
- The system has to be highly decoupled, modular, to allow adding and removing modules without affecting the rest.
- The modules have to be reusable.
- Our goal is to offer maximum flexibility, separation of concern, strong cohesion, loose coupling and low cost of change.

To meet the described above requirements and tendencies of the modern systems, the proposal we make is to copy the principles of the nature.

We are proposing the idea that every application should follow four principles and have clearly distinct and recognizable elements (systems):

2.1. Communication principle

Every system should have at least one communication layer for sending and receiving signals. If the system consists of many units (modules) each module should have its own communication layer. In fact the existence of a communication layer separates the groups of elements into an unit.

Look on creating an application like creating a new universe. So when we ask ourselves where do we start, we can look our dimension and ask the same question "Where and how everything started in this dimension?". There is no certain scientific answer to this question. We don't know for certain where and how everything started. The answers are based on the theories and are more or less intuitive and spiritual. But what we see for certain is the fact that everything is linked in this world. There is a communication environment or environments, through which, the entities on every level are linked to form the world as we know it. We have no choice, but look from the philosophical point of view or even spiritual. Even if we go to the spiritual level because there are no scientific answers, we can find that the first was the communication. As it is written in the holly bible “In the beginning was the Word, and the Word was with God, and the Word was God.” [29] The word is the communication. It is the initial vibration. As creators of our own worlds we have to follow this model of creation. First we have to create the communication channel. Simply said: “Don’t think any further. Start your application by defining a class Communication, Event Manager, Connect.” There are many terms used in the software engineering for the same thing. In fact this principle is easy to recognize everywhere in the modern systems. Even if we start by looking the lowest level in the stack – the hardware, we will recognise the communication systems. The signals travel through the hardware buses and connect the parts of the computer. The microprocessor enters an infinite loop and waits for interruptions trough another buses - communication environment. We can use the same approach and build our communication buses using the principles of the OOP. If we see the things from this perspective it becomes obvious how to “Decouple” a system.

Communication system is crucial for every organism on the planet. In fact the different communication systems allow for separation the constellations of cells called organisms. They are united to form more complex organisms and environments etc. While the principles described in the paper are not unknown they have to be put together and formalised.

While we are not reinventing the “Event Driven Development” we are giving a different perspective and way of thinking, which may help for solving complex problems. Also we identify the fact that system build by using this approach is easy to extend, maintain and scale. The modules become extremely reusable.

Communication layer is known under many different names in the applications and frameworks. Very often the name “Event Manager” [30] is used to describe a system for communication. The processes of sending and receiving signals in this environment (carrier, medium) are also called differently. Some examples are publish/subscribe, attach/trigger, listen/notify. But generally speaking they have the same nature and are just called differently. In fact so called “Event Manager” plays the role of the environment through which the signals travel.

To illustrate the Communication principle we will provide a very simple implementation. The class plays the role of the connection/communication environment for the signals send back and forth in this level of the system. From this class a communication/connection objects can be instantiated for every level (unit) in the system. Here the name "EventManager" is used as the name of the class and attach/trigger for the names of the main methods.

```
function EventManager() {
    var listeners = {};

    this.attach = function(event, listener) {
        if (!listeners[event]) {
            listeners[event] = [];
        }
        listeners[event].push(listener);
    };

    this.trigger = function(event, target, options) {
        if (!listeners[event]) return;
        for (var i = 0; i < listeners[event].length; i++) {
            listeners[event][i]();
        }
    }
}
```

2.2. Unit (Module, Entity, Cell, World) principle

A common communication layer is a base for building a unit (Module, Entity, Cell, World). The common communication layer can be called "internal" for a given unit. The communication layer of the parent unit becomes external for the child units. Every unit can be built by infinite amount of nested child units. Every child unit on its turn can be built by infinite amount of other child units.

The units can be seen as cells in a living organism.

The world is infinite in both directions inside and outside. In the micro world (inside) we discover more and more particles. But the same is valid in the macro world (outside). Our telescopes discover more and more worlds. And the worlds are nested. Each world consists of many other smaller worlds.

The world is infinite in both directions. On every level there is a common communication environment. Some examples from the nature can be taken.

We can make analogy with a flock of fishes or birds. The fishes in the flock create a living organism. They act as one creature. What allows them to do that is the common environment, the ability to communicate. This bound makes them one organism. If we look an individual fish it is like a cell of this organism. But itself the fish is separated from the common, by having its own communication/signal system. This time the system unites the individual cells of the body. If we look deeper every cell is independent from the rest of the cells and if one cell dies the rest of the cells continue to live. On the upper level if one of the fishes dies the flock continues to exist. This flock on its turn is a part of bigger ecological system and so on and so fort. The examples in the nature are endless. So all we have to do is just emulate/copy this world in our applications. When a new object gets created it gets injected with the reference to the common communication environment on its level. So this object can receive and emit signals in this communication environment. On its turn the newly created object creates its own world with its own internal communication environment. It is internal from its own point of view. But it can be external for the internal systems. This environment is shared by all modules building our object. On its turn every smaller individual unit (object) can be seen as a cell. And so on and so forth. In fact we know very well the event driven development. But we are proposing a different perspective.

```
function Unit(externalEventManager, id) {
…
// Create an own communication system for this level
var eventManager = new EventManager();
...
}
```

The communication object is mandatory to create a unit (world). Each instance of the class “EntityManager” creates a new world with own communication system. Each world can have space and time if necessary.

2.3. Time principle

There is time in the world of the human. We use it to measure and synchronise events. So we may need to create a model of the time. Each communication layer could have a notion of its own time. Some very simple applications may not need time in their units (worlds). If the units in our virtual world don't depend on time we don't have to follow the "Time Principle". This is the reason it is optional.

If our system needs a notion of time a class representing this entity also can be created. The time and all other entities on each level depend on a Communication Layer described in 2.1. The time object gets injected with an instance of "EventsManager". A few lines of code are enough to allow the time to start ticking in each level of communication in our system

```
function Time(eventManager) {
    this.startTicking = function(timeperiod) {
        setInterval(function(){
            eventManager.trigger('tick');
        }, timeperiod);
    }
}
```

2.4. Space principle

The motivation for creating a space in our system is the fact, that we have space in our world. Some units may change their position in the virtual space. If the relationships between the units and the units itself depend on their positions in space, we have to add a notion of space. There are some applications where the position of the units in space is not a concern. In this case we don't have to create object representing the space. Each unit can have its own space.

The space is an optional entity in our system. If we need a notion of space a very simple class can be defined and object can be instantiated on the necessary levels. So each world has its own space.

```
function Space() {
    this.coordinates = [];

    this.placeInSpace = function(x, y, z, entity) {
        this.coordinates[x][y][z].push = entity;
    }

    this.getFromSpace = function(x,y,z) {
        return this.coordinates[x][y][z];
    }
}
```

3. Results and tools for building the system

Prototype software systems were built as a proof of a concept, using the model. Each prototype is sound and stable. They have been built from developers without skills and experience. The transfer of knowledge was short and easy. The model allows each software artefact to be developed and tested separately from the rest of the system by different teams. The model enforces strong cohesion and loose coupling. The developers can look for clearly defined, standard, familiar software artefacts in the software systems. This can make the knowledge transfer much faster and can increase the productivity of the software teams. Many different programming languages and technologies can be used. There are Expert Systems build with “LAMP stack” (Linux, Apache, MySQL, PHP/Perl/Python) [31]. There are many other technologies used for the same purpose.

Our choice is JavaScript. JavaScript has many advantages [32]. The language can be used on the server and client side. If we see Internet like a global supercomputer we will need a terminal to communicate with it. The web browser turns to be the universal User Interface Platform and operating system for communication with this global supercomputer. JavaScript is a language especially created to run in the browser. It is extremely popular nowadays. There are other technologies like Java Applets, Adobe Flash and Microsoft Silverlight created to run on the browser, but the newest tendencies is to use JavaScript [33]. Also JavaScript can be used on mobile devices as well. This allows the development to be done with only one language on all devices operating systems and environments. The code can be reused. With JavaScript we can apply Prototype Based, Class Less, Object-Oriented Event Driven Development. It is the perfect language for applying the described in this paper principles. JavaScript can be used for creating Artificial Intelligence [34]. It supports many different programming styles.

We are using the “MEAN stack” [35] (MongoDB, Express, AngularJS, NodeJS). The user interface is built with HTML5, CSS3. The DOM tree is used as a template (presentation layer of the application). The application on the client side is a SPA (Single Page Application).

The server side offers RESTful API. But RPC services can be offered as well..

The system should be able to change its own code based on the environment and situations.

4. The conclusion

The described model allows the development to be industrialized. The tasks can be separated in individual modules and delegated to different developers. The communication inside of the modules and between the modules is standard with simple and easy to implement interfaces.

The Natural-Oriented Programming helps to build better software in a consistent way. The development of the enterprise applications can become standard and trivial process instead of craft and art. The wide adoption of models like this could allow the industrialization of the software development. The outcome of the software projects will be more predictable and controllable.

An interesting perspective is to use mathematical formalisms such as Interval Temporal Logic, Petri Nets, Generalized Nets and others to describe and abstract the terms in the model. Using models like the described one, could allow automatic code and system generation. The combination with Artificial Intelligence (AI) creates new exciting opportunities.

Acknowledgements

This work is partially supported by the FP17-FMI-008 project of the Scientific Fund of the Plovdiv University Paisii Hilendarski, Bulgaria.

References

[1] M.A.F de Souza, M.A.G.V Ferreira, Designing reusable rule-based architectures with design patterns, *Expert Systems with Applications*, Vol. 23, Issue 4, November 2002, Pages 395–403.

[2] Bruce Douglass (2013) Chapter 4 – Software Design Architecture and Patterns for Embedded Systems. Software Engineering for Embedded Systems, 2013, Pages 93–122.

[3] Apostolos Ampatzoglou, Apostolos Kritikos, George Kakarontzas, Ioannis Stamelos, An empirical investigation on the reusability of design patterns and

software packages, *Journal of Systems and Software*, Vol. 84, Issue 12, December 2011, Pages 2265–2283.

[4] Maria Paasivaara, Casper Lassenius, Communities of practice in a large distributed agile software development organization case ericsson, *Information and Software Technology*, In Press, Accepted Manuscript, Available online 26 June 2014.

[5] Xiaodan Yu, Stacie Petter, Understanding agile software development practices using shared mental models theory, *Information and Software Technology*, Vol. 56, Issue 8, August 2014, 911–921.

[6] Diana Kirk, Ewan Tempero, A lightweight framework for describing software practices. Journal of Systems and Software, Volume 85, Issue 3, March 2012, Pages 582–595.

[7] Varun Gupta, Jitender Kumar Chhabra, Dynamic cohesion measures for object-oriented software, *Journal of Systems Architecture*, Vol. 57, Issue 4, April 2011, 452–462.

[8] Istehad Chowdhury, Mohammad Zulkernine, Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities, *Journal of Systems Architecture*, Vol. 57, Issue 3, March 2011, 294–313.

[9] Jehad Al Dallal, Lionel C. Briand (2010) An object-oriented high-level design-based class cohesion metric, *Information and Software Technology*, Vol. 52, Issue 12, December 2010, 1346–1361.

[10] P.C. Jha, Vikram Bali, Sonam Narula, Mala Kalra, Optimal component selection based on cohesion & coupling for component based software system under build-or-buy scheme, *Journal of Computational Science*, Vol. 5, Issue 2, March 2014, 233–242.

[11] Pekka Alho, Jouni Mattila, Software fault detection and recovery in critical real-time systems: An approach based on loose coupling, *Fusion Engineering and Design*, In Press, Corrected Proof, Available online 14 May 2014.

[12] Istehad Chowdhury, Mohammad Zulkernine, Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities, *Journal of Systems Architecture*, Vol. 57, Issue 3, March 2011, 294–313.

[13] Michael Stal, Chapter 3 – Refactoring Software Architectures. Agile Software Architecture, 2014, 63–82.

[14] Mohammad Alshayeb, Empirical investigation of refactoring effect on software quality, *Information and Software Technology*, Vol. 51, Issue 9, September 2009, 1319–1326.

[15] Bruce Anderson, Object-oriented programming, *Microprocessors and Microsystems*, Vol. 12, Issue 8, October 1988, 433–442.

[16] Francesco Logozzo, Agostino Cortesi, Abstract Interpretation and Object-oriented Programming: Quo Vadis?, *Electronic Notes in Theoretical Computer Science*, Vol. 131, 24 May 2005, 75–84.

[17] David Robinson, 1 – Introduction to Aspect Oriented Programming (AOP). Aspect-Oriented Programming with the e Verification Language, 2007, 1–30.

[18] Pascal Fradet, Ralf Lämmel, Special issue on foundations of aspect-oriented programming, *Science of Computer Programming*, Vol. 63, Issue 3, 15 December 2006, 203–206.

[19] Gökhan Engin, Burak Aksoyer, Melike Avdagic, Damla Bozanlı, Umutcan Hanay, et al., Rule-based Expert Systems for Supporting University Students, *Procedia Computer Science*, Vol. 31, 2014, 22–31.

[20] Joan Marc Llargues Asensio, Juan Peralta, Raul Arrabales, Manuel Gonzalez Bedia, Paulo Cortez, et al., Artificial Intelligence approaches for the generation and assessment of believable human-like behaviour in virtual characters, *Expert Systems with Applications*, Vol. 41, Issue 16, 15 November 2014, 7281–7290.

[21] S. Sahin, M.R. Tolun, R. Hassanpour, Hybrid expert systems: A survey of current approaches and applicationsReview Article, *Expert Systems with Applications*, Vol. 39, Issue 4, March 2012, 4609–4617.

[22] Mehdi Piltan, Erfan Mehmanchi, S.F. Ghaderi, Proposing a decision-making model using analytical hierarchy process and fuzzy expert system for prioritizing industries in installation of combined heat and power systems, *Expert Systems with Applications*, Vol. 39, Issue 1, January 2012, 1124–1133.

[23] B.A. Akinnuwesi, Faith-Michael E. Uzoka, Abayomi O. Osamiluyi, Neuro-Fuzzy Expert System for evaluating the performance of Distributed Software System Architecture, *Expert Systems with Applications* 40, 2013, 3313–3327.

[24] Dimitris N. Chorafas, Heinrich Steinmann, CHAPTER 11 – Application Programming Interface, Formats and Protocols, and Remote Data Access, *Solutions for Networked Databases*, 1993, 185–198.

[25] Mauricio Arroqui, Cristian Mateos, Claudio Machado, Alejandro Zunino, RESTful Web Services improve the efficiency of data transfer of a whole-farm

simulator accessed by Android smartphones, *Computers and Electronics in Agriculture*, Vol. 87, September 2012, 14–18.

[26] Toni Giorgino, M.J. Harvey, Gianni de Fabritiis, Distributed computing as a virtual supercomputer: Tools to run and manage large-scale BOINC simulations, *Computer Physics Communications*, Vol. 181, Issue 8, August 2010, 1402–1409.

[27] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, Marimuthu Palaniswami, Internet of Things (IoT): A vision, architectural elements, and future directions, *Future Generation Computer Systems*, Vol. 29, Issue 7, September 2013, 1645–1660.

[28] Alexander Bolonkin, 13 – Setting God in a Computer-Internet Net, *Universe, Human Immortality and Future Human Evaluation*, 2012, 115–122.

[29] http://biblehub.com/john/1.htm

[30] Sang Seok Lim, Kyu Ho Park, ECEM: an event correlation based event manager for an I/O-intensive application, *Journal of Systems and Software*, Vol. 74, Issue 3, 1 February 2005, 229–242.

[31] George J.Moridis, Matthew T. Reagan, Heidi Anderson Kuzma, Thomas A. Blasingame, Y. Wayne Huang, et al., SeTES: Aself-teaching expert system for the analysis, design, and prediction of gas production from unconventional gas resources, *Computers & Geosciences* 58, 2013, 100–115.

[32] William J. Buchanan, 26 – JavaScript, *Software Development for Engineers*, 1997, 415–436.

[33] Jack Moffett, Chapter 9 – Looking Toward the Horizon, *Bridging UX and Web Development*, 2014, 163–188.

[34] http://mind.sourceforge.net/js.html

[35] http://mean.io/#!/

Faculty of Mathematics and Informatics
Plovdiv University
236 Bulgaria Blvd,
Plovdiv 4003, Bulgaria
E-mail: cheresharov@uni-plovdiv.bg

ЕСТЕСТВЕНО-ОРИЕНТИРАНО ПРОГРАМИРАНЕ

Стоян Черешаров

Резюме. Настоящата статия описва модел за софтуерна архитектура за изграждане на софтуерни системи наречен „Естествено-ориентирано програмиране“. Съществуващите парадигми на обектно-ориентираното, аспектно-ориентираното, събитийното програмиране, разработката базирани на тестове и предметната област, шаблоните за дизайн както и други изследвания не отговарят напълно на въпроса как трябва да изглежда цялостната архитектура на една софтуерна система. Ние предлагаме обобщен модел за софтуерна архитектура и нейните четири принципа. Те включват принципите на комуникация, единица, време и пространство. Това е нова парадигма и начин на мислене. Бяха изградени прототипни софтуерни системи като доказателство за концепцията, използвайки модела. Всеки прототип е надежден и стабилен. Моделът позволява разработването на корпоративни приложения да стане стандартен и тривиален процес вместо занаят и изкуство. Широкото приемане и използване на подобни модели би могло да позволи индустриализацията на разработката на софтуер.