UDC 519.85:004.42

WEB-SERVICE. RESTFUL ARCHITECTURE

M. Melnichuk[1], Yu. Kornienko[2], O. Boytsova[3]

[1,2,3]Odessa National Academy of Food Technologies, Odessa, Ukraine
ORCID: 20000-0003-3630-8384, 30000-0001-9994-587X
Scopus ID: 256578764800
E-mail: [1]mickscorsese77@gmail.com, [2]yurikkorn@gmail.com, [3]ola_odessa@ukr.net

DOI: 10.15673/atbp.v10i1.876

Abstract: Network technology for interaction between two applications via the HTTP protocol was considered in article.When client works with REST API - it means it works with "resources", and in SOAP work is performed with operations. To build REST web services, you must follow certain principles: explicit use of HTTP methods, access to resources by URI, stateless, HATEAOS, caching, transfer of objects in JSON or XML representation. But sometimes some principles are ignored to ensure a higher speed of work and to reduce development time.

The pros and cons of using JSON and XML representations were considered, and it can be said that using the JSON format reduces the amount of data transfer, and with the use of XML, the readability of data increases.

Also, two main ways of data transfer in REST web services were considered: converting the file to Base64 and transferring it as an object field or transferring the file using the usual HTTP multipart. The Base64 standard approach gives a higher speed for multiple files in a single request, because only one HTTP connection is created, but these files are stored in RAM during request processing, which increases chance of the application crashing.

In the conclusion, the advantages of using web services and their wide use in other architectural approaches were considered, which increases the popularity of web services.

Анотація: У статті було розглянето мережеву технологію для взаємодії між двома додатками через протокол HTTP. Клієнт працюючи з REST API - працює з "ресурсами", а в SOAP робота йде саме з операціями. Для побудови REST веб-сервісу було дотримано визначених принципів: явне використання HTTP методів, доступ до ресурсів через URI, відсутність стану, HATEAOS, кешування, передача об'єктів у JSON або XML уявленні. Але іноді ігнорують деякі принципи, для отримання більшої швидкості роботи або для зменшення часу на розробку.

Були розглянуті плюси та мінуси JSON та XML уявлень, та можна сказати, що при використанні JSON формату зменшується кількість переданих даних, а при XML збільшується читаність даних.

Також було розглянуто два основні способи передачі файлів у REST веб-сервісах: конвертування файлу у Base64 та передача його як поле об'єкту або передача файлу завдяки звичайному HTTP multipart. Підхід використовуючи Base64 стандарт надає більшу швидкість передаючи декілька файлів у одному запиті тому що створюється лише один HTTP зв'язок але при цьому файли під час обробки запиту зберігаються у оперативній пам'яті, що збільшує ймовірність краху веб-додатку.

У висновку були розглянуті переваги використання веб-сервісів та їх широке використання у інших архітектурних підходах, що збільшує популярність веб-сервісів.

Keywords: HTTP, SOAP, REST API, URI, HATEAOS, JSON, XML.
Ключові слова: HTTP, SOAP, REST API, URI, HATEAOS, JSON, XML.

Introduction

Web service - a network technology that creates the ability to interact with different applications over the network by sending messages between each other. HTTP / HTTPS is used as the transport protocol, although some web services may use other protocols. The W3C consortium defines a web service as a "software system developed to support interoperable intercomputer network interoperability" [1].

Initially web servers sent the information only in the form of hypertext documents (HTML), browsers were used to display the documents and most of these documents were static, thus for adding new information developers had to create new HTML pages and add links to new pages from old pages. The next step was a massive transition to the use of page generators and the construction of dynamic websites. At the same time there were web services, but they were used only in enterprise applications for integration with various external systems.

After JavaScript became popular and the AJAX (Asynchronous Javascript and XML) approach was offered, web services began to be used to work with regular websites, but only part of the requests were sent to web services. Most of the data is generated on the basis of page generator on the server side.

There are several principles that underlie web services:

- Processing of one request consists of three operations: receiving a request, processing, sending a response
- Each web service has its own request and response format
- Any machine connected to the Internet can send request to web service

Research methodology

The following methods were chosen to research web services namely Restful architecture:

- Analysis - consideration of the approach, taking into account its individual properties.
- Experiment - testing the approach in specially created conditions.
- Comparison - comparing several approaches or their properties with each other according to a particular parameters.
- Measurement - based on accurate calculations and numerical values.

During the analysis, the basic principles of Restful Architecture will be considered, their efficiency and expediency. Also, sizes of representations of the objects will be compared with each other.

In the practical part, an experiment will be conducted on measuring file transfer speeds using two different approaches and comparing the results and analyzing the properties of each approach.

Theoretical part

Architectural principles that create Restful architecture in total are offered by Roy Fielding in 2000 in a dissertation entitled "Architectural Styles and Design of Network Software Architectures", while this architectural approach was not adopted by the community, but has now become widespread.

It is accepted that the client is working with the REST API - working with "resources". Each resource must have its own unique URI (Uniform Resource Identifier). Here you can pay attention to the difference between SOAP, because SOAP works with operations and it makes easier to develop more complex logic. SOAP is more often used in the development of applications for complex logic enterprises.

As already mentioned, REST is a set of architectural principles, and if they are all followed, REST APIs will be built. But often some of the principles are not followed to optimize performance, reduce development time and more.

Explicit use of HTTP methods. One of the architectural principles of RESTful architecture suggests more explicit use of HTTP-methods, namely:

- POST - Creating a Resource (Create)
- GET - Getting a Resource (Retrieve)
- PUT - Change state of the resource (Update)
- DELETE - Delete the resource (Delete)

These operations: create, retrieve, update, delete in the sum called CRUD. Often these operations are quite enough to perform all the operations of the web service

URI. URI is a unified resource identifier.

REST web services use a certain URI formation style. Suppose we create a web service in which we work with entity “student”, over this entity can do at least 4 operations: creation, retrieving, updating and deletion. Using the REST style of Web services, the URLs will look like this ("service.com" - the conditional domain of the service):

- GET service.com/students - Getting the status of all students
- GET service.com/students/5 - Getting the status of student with ID equals "5"
- POST service.com/students - Creating a new student, the object itself is sent in the body of the request
- PUT service.com/students/5 - Changing state of student with ID equals "5"
- DELETE service.com/students/5 - Delete a student with ID equals "5"

Stateless. Another fundamental principle of RESTful architecture is not to store the current state, that is, each request includes all the necessary information for its processing and is completely independent from the previous and next request. This is also expressed in the absence of sessions that a user receives on regular websites when authorizing. There are several ways to avoid using sessions in order to build the true REST API.

The most popular way is to use Token Authentication. The sense of this authentication method is that the user sends the account name or email and password using the HTTPS protocol to the web service. Then a token is created on the web service, which includes the name of the account / email and additional information that will be useful for further work with the user, such as the role of the user, the date of the expiration of the token service, and so on. After additional information has been collected in the token, to protect the data and to allow only the current web service to work with the token, the token will be encrypted using keys that were previously generated by the developer or administrator. Additional information is required in order to reduce the number of queries to the database for the recovery of the required data, considering that in subsequent actions of the user, the token will be sent each time to the request header and on the server side it will be decoded to verify the validity of the token.

HATEOAS. In addition, there is another principle that is important for creating a true REST API. This principle is called HATEOAS (Hypermedia as the Engine of Application State). It states that work should be conducted only with hyperlinks, these resources contain information about themselves and links to other resources. That is, on the example of the "student"

entity, in the transferred object should be a reference to this resource, to avoid working with resource identifiers in the form of the normal value of the object and for example the entities of "course", "group", and others. It is still necessary to add links to the indirect changes of the state of the resource to the body of the answer. This principle is often not used.

Caching. Not so rarely, web services based on Restful architecture are subject to high load on the part of customers, then to reduce the time to answer - it is recommended to apply caching resources on their side. To do this, the server must be able to work correctly with the "prompts" for the client, namely, which resource can be cached, which is the validity period of the cached resource. This principle is usually not used on web services with a low load and a small amount of data transmitted due to the complexity of its proper implementation and no need.

Data exchange formats. Often, a particular web service uses a certain format for data exchange. For example, when communicating with REST, the web service will most likely use JSON (JavaScript Object Notation) as the format of the exchange, while SOAP uses XML, but it's really confronted with the REST API, which will interact with the server using XML.

When using RESTful architecture using the JSON object transfer format instead of XML, the load on the communication channel is reduced, higher productivity due to the fact that the body size of the response from the web service significantly decreases.

For example, working with the conditional entity "student", which has three fields: email, first name, last name when using:

●JSON:

```
{
    "email":"email@domain.com",
    "first_name":"Ivan",
    "last_name":"Ivanov"
}
```

Fig. 1 – JSON representation of the object "student"

●XML:

```
<student>
    <email>email@domain.com</email>
    <firstName>Ivan</firstName>
    <lastName>Ivanov</lastName>
</student>
```

Fig. 2 – The first version of the XML representation of the object "student"

Or

```
<student email="email@domain.com" firstName="Ivan" lastName="Ivanov">
</student>
```

Fig. 3 – The second version of the XML representation of the object "student"

In this case, the JSON submission takes 71 bytes instead of 120/80 bytes in the XML submission. And from this recommended submission is precisely the JSON format, which is a classic option.

If the API is intended to be public, then it is recommended to support both types of data exchange, so that third-party developers themselves can determine for themselves a more convenient type of representations. However, it should be remembered that in case of abundant use of XML representations on the communication channel will be more load. We should not forget about the documentation for third-party developers, because only they will be able to fully use the API. The documentation should describe the main use cases of users, as well as fully consider examples of server requests (protocol, headline and body) and responses.

Practical part

There are two most popular approaches to working with REST files, but you can not identify the best one, so each one has its own advantages and disadvantages.

The first approach is to convert binary data to Base64 format, which increases the file size, usually by 25 - 35%. This approach allows you to transfer files as parameters of JSON or XML objects. Also, using this approach, it is possible to conveniently record data in a database as a normal parameter of an object, but it is also a disadvantage because when extracting a Base64 file from a database, the entire operation slows down, that is, when we want to get a user profile and return it to his client we need to fetch his photo or avatar from the database and this operation is very difficult because the volume of the file is very large and much more than all the data from the profile and after the operation of deleting the profile, the data fall into the RAM and to the moment until they are sent to the user in full they can not be cleared from memory.

The second approach is to provide each file its own URI and its regular sending via the HTTP protocol as a multipart. Using this approach, files can also be saved in the database, but it does not make sense because it will also have to spend the RAM until they are sent. As a habit, files are stored in a specific directory on the server and when requested by the client, it

checks does user have access to these files and, if yes, uses this file to redirect the buffered thread immediately to the client. That is, in memory there is only a buffer size inserted, which is much smaller than the file itself. But for each file you need to create a new HTTP connection that takes a while.

To test these two approaches, the REST web service was written in which work is being conducted with distance learning at a higher education institution. For example, 2 operations were selected: updating the avatar of the profile and receiving the changed profile from server.

The purpose of testing is to compare the speed of these two approaches and the load on the resources of the server.

As an avatar was found a picture of 184 pixels in height and 184 pixels wide in the expanses of the Internet. The size of the picture in binary form is 59592 bytes. The size of the picture in the Base64 standard is 79,456 bytes, that is, in this format the size is larger by 33%.

For comparison, two basic values will be used: server processing time and time of the entire operation (is the amount of server processing time and time to create an HTTP connection and time to transfer data).

Consider the first operation - updating the avatar of the profile, this operation consists of checking the JWT token on the side of the server and saving the image. When using the representation of a file in the Base64 standard, the image was sent to the server as a single parameter of the JSON object, and in the second approach the image was the entire body of the request.

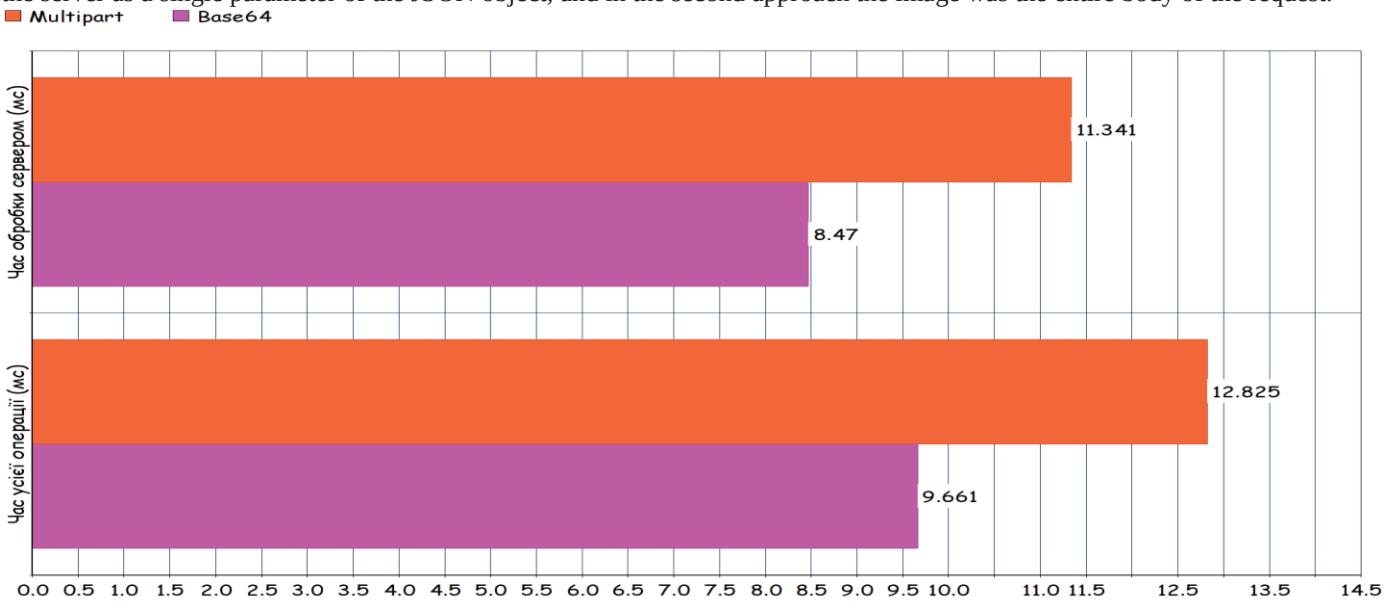

Fig. 4 – Operation updating the avatar

The second operation is to receive profile information and avatar. The difference in both approaches is that when using a file in the Base64 standard, only one HTTP connection will be created in which all data in the form of a JSON object will be transmitted. In the second approach, the profile information and the avatar's own URLs are first transmitted in the first HTTP response, and then another HTTP connection is created to get the avatar itself.

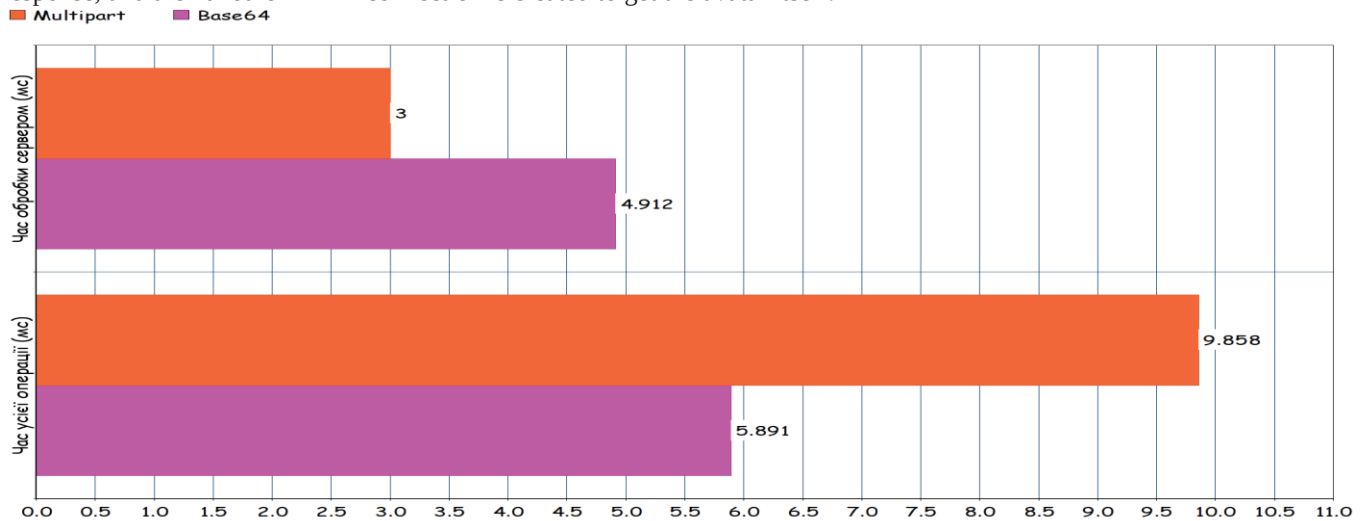

Fig. 5 – Getting profile and avatars

As you can see from the diagrams, the operation using the Base64 standard is faster, because getting the file from the database and working with it in the RAM provides a higher speed, but it is not very safe because with a large number of server requests for the purpose get files, free RAM at the time of processing can end and the server can fall down.

You should also pay attention to the fact that when performing the operations of getting profile and avatar server processing time using a second approach is much smaller than in the first approach, that is, ⅔ of the operation involved the creation of communication and the transfer of the file itself, which is well reflected in the low loaded on the server.

Another thing to say about the load capacity of the communication channel is that using the other approach the amount of information transmitted is about 33 percent less, in this example it does not matter because the picture was small and at one time only one request was fulfilled, but at a large number of requests and a large amount of files - the bandwidth may not be enough and there will be long delays.

Conclusions

Recently, the approach to creating sites that use JavaScript to get all data from the server has gained popularity. That is, when you go to the URL browser receives HTML markup and JavaScript scripts. The markup does not contain any data and is only a template for further filling in the data. After the client has got markup and scripts, he caches them so that next time it is not necessary to reload this, which considerably accelerates the speed of work. Then the client sends requests to web services, after receiving responses containing the data itself, the client updates the markup based on the data received. That is, it turns out that the page generator is not on the server side but on the client side. This approach is convenient for several reasons:

- The server has less load, because the client is engaged in converting data into HTML markup
- Also, the amount of information transmitted is much less in all cases, except for the initial loading of all templates and scripts.
- When creating mobile, desktop applications you do not need to rewrite the server part. That is, the server part is created regardless of which client platforms will use it.

Another thing to say about the popular architecture of server applications - "microservice architecture". The essence of this architecture is that instead of building a monolithic application, a set of services is built, each of which contains a certain logic and performs a certain functionality. For complete processing of client requests on a server, these services interact with each other through web services. I want to highlight one of the many benefits of this approach - each service from the many services that form the application can be written in another programming language, unlike other services, may contain any DBMS and generally use completely different approaches. This greatly facilitates the set of team development, the process of developing and selecting an optimal stack for specific tasks, which in turn can significantly improve the performance impact. All this is made possible by web services as well as Restful architecture.

Referenses

[1] “REST - Semantic Web Standards.” [Online]. Available: https://www.bing.com/cr?IG=F978DB1BF7E84D46A8B33EB5D1790266&CID=082B3C6257986E2C1FE437C156376FB3&rd=1&h=2787hUGMzYLuSUnq_ZTeGW7jVIt41I1GMVZInqJAxfI&v=1&r=https%3a%2f%2fwww.w3.org%2f2001%2fsw%2fwiki%2fREST&p=DevEx,5065.1. [Accessed: 27-Jan-2017].

[2] “Web-сервисы RESTful: основы,” IBM - United States, 09-Feb-2015. [Online]. Available: https://www.ibm.com/developerworks/ru/library/ws-restfu/index.html. [Accessed: 20-Mar-2017].

[3] “Архитектура REST / Хабрахабр.” [Online]. Available: https://www.bing.com/cr?IG=F91CA7B4229A4D37A78C983971CBC099&CID=14C9F405228A6C34246FFFA623256D63&rd=1&h=Er_fYKwoLRG0FIGpQLF_DXeX5C7Aq3S7A42X2_7mnjE&v=1&r=https%3a%2f%2fhabrahabr.ru%2fpost%2f38730%2f&p=DevEx,5063.1. [Accessed: 07-May-2017].

[4] “github.com.” [Online]. Available: https://www.bing.com/cr?IG=B8C587E99499466A8B1985EE50AB4E56&CID=067B0DD130F6683D165F067231596980&rd=1&h=DvMnugNmFXffO88ET_7MVMcn6uXYrh_Ib7-Ju-Si-es&v=1&r=https%3a%2f%2fgithub.com%2fzzet%2fRestApiTutorial.ru%2fblob%2fmaster%2fhttpstatuscodes.html&p=DevEx,5040.1. [Accessed: 17-Jun-2017].

[5] Lastnitescurry, “lastnitescurry/j2ee-rest-jersey-jetty-tomcat-maven-jenkinsfile,” GitHub, 12-Nov-2016. [Online]. Available: https://github.com/lastnitescurry/j2ee-rest-jersey-jetty-tomcat-maven-jenkinsfile. [Accessed: 27-Oct-2017].

Література

[1] REST - Semantic Web Standards [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://www.bing.com/cr?IG=F978DB1BF7E84D46A8B33EB5D1790266&CID=082B3C6257986E2C1FE437C156376FB3&rd=1&h=2787hUGMzYLuSUnq_ZTeGW7jVIt41I1GMVZInqJAxfI&v=1&r=https%3a%2f%2fwww.w3.org%2f2001%2fsw%2fwiki%2fREST&p=DevEx,5065.1.

[2] Web-сервисы RESTful: основы,” IBM - United States [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://www.ibm.com/developerworks/ru/library/ws-restfu/index.html.

[3] Архитектура REST [Електронний ресурс] // Хабрахабр. – 2017. – Режим доступу до ресурсу: https://www.bing.com/cr?IG=F91CA7B4229A4D37A78C983971CBC099&CID=14C9F405228A6C34246FFFA623256D63&rd=1&h=Er_fYKwoLRG0FIGpQLF_DXeX5C7Aq3S7A42X2_7mnjE&v=1&r=https%3a%2f%2fhabrahabr.ru%2fpost%2f38730%2f&p=DevEx,5063.1.

[4] github [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://www.bing.com/cr?IG=B8C587E99499466A8B1985EE50AB4E56&CID=067B0DD130F6683D165F06723159698

0&rd=1&h=DvMnugNmFXffO88ET_7MVMcn6uXYrh_Ib7-Ju-Si-es&v=1&r=https%3a%2f%2fgithub.com%2fzzet%2fRestApiTutorial.ru%2fblob%2fmaster%2fhttpstatuscodes.html&p=DevEx,5040.1.
[5] Lastnitescurry, "lastnitescurry/j2ee-rest-jersey-jetty-tomcat-maven-jenkinsfile [Електронний ресурс] // GitHub. – 2016. – Режим доступу до ресурсу: https://github.com/lastnitescurry/j2ee-rest-jersey-jetty-tomcat-maven-jenkinsfile.

УДК 681.513.7

НАСТРОЙКА НЕЙРОННОЙ СЕТИ ПРИ АВТОМАТИЧЕСКОМ СИНТЕЗЕ СЕТЕЙ ПЕТРИ

Гурский А.А.[1], Дубна С.М.

Институт компьютерных систем и технологий им. П.Н. Платонова Одесской национальной академии пищевых технологий, г. Одесса
[1]ORCID: http://orcid.org/0000-0001-5158-2125
[1]E-mail: Gurskiya2017@gmail.com

DOI: 10.15673/atbp.v10i1.877

Аннотация: В настоящей работе представлен определенный этап разработки интеллектуальной системы, связанной с автоматическим синтезом сетей Петри. Рассматривается определенная архитектура искусственной нейронной сети, которая положена в основу интеллектуальной системы, направленной главным образом на формирование алгоритмов настройки координирующих систем автоматического управления. Особенность функционирования рассматриваемой архитектуры нейронной сети заключается в том, что в любой момент времени может быть активен только один нейрон из определенного количества возможных для активизации других нейронов сети. В работе представляется алгоритм настройки данной нейронной сети, который связан с инцидентной матрицей формируемой сети Петри. При этом формируемая сеть Петри представляет алгоритм настройки координирующей системы автоматического управления.

В заключительной части работы представлена разработанная в программной среде MATLAB/Simulink система, формирующая инцидентную матрицу сети Петри на базе функционирования нейронной сети. Отражается визуализация процесса формирования сети Петри, представляющей алгоритм настройки системы управления. Данная визуализация дает возможность представить результат формирования алгоритма, тем самым позволяет определить специалисту, при необходимости, нужную корректировку данного алгоритма.

Abstract: The certain stage in the development of the artificial intelligent system is considered in this paper. This intelligent system associates with the automatic synthesis and the composition of Petri nets. The intelligent system consisting of the certain architecture of the neural network is considered. This intelligent system is designed to develop and generate algorithms for tuning the coordinating systems of automatic control. The peculiarity of the neural network architecture lets to activate one neuron of possible at any moment of time. In this case the activity of one neuron causes the activation of other neurons in the network. The algorithm for tuning this neural network is presented in this scientific work. This tuning algorithm associates with the incident matrix formed by Petri net. The Petri net represents the algorithm for tuning of the coordinating systems of automatic control.

The system designed in the MATLAB / Simulink software environment is presented in the final part of the scientific work. This system forms the incident matrix of the Petri net during the functioning of the neural network. The visualization of the process of the Petri net formation is presented. This visualization of the Petri net provides the opportunity to find out the result of the algorithm formation. The specialist can determine the necessary adjustment of the algorithm for tuning of the coordinating systems of automatic control. The attempt was made to use principle of reinforcement learning in this artificial intelligence system. This principle of reinforcement learning represents the actively developing field of the artificial intelligence, neural network modeling and control.