Interaction of Cloud Services with External Software and its Implementation on the IACPaaS Platform

Valeria Gribova[a], Leonid Fedorischev[a], Philip Moskalenko[a] and Vadim Timchenko[a]

[a] *Institute of Automation and Control Processes Far Eastern Branch of the Russian Academy of Sciences (IACP FEB RAS), Radio 5, Vladivostok, 690041, Russia*

Abstract

Cloud computing applications have a number of advantages: no installation to hardware is required, simplified maintenance, access, teamwork, etc. At the same time, there are tasks that cannot be moved to or executed in cloud platform environment (e.g. due to software or hardware limitations or restrictions). Therefore, there is an obvious need for standards, technologies and tools that allow combining cloud platforms with non-cloud software. The mechanisms for implementation of cloud platform interaction with external software are considered. The architecture of such interaction consists of four main components. Two of them are placed on the cloud platform: the service and the gateway component. Two others – on the remote system: intermediary program and external software. The studied mechanisms are used as the basis for creation of the IACPaaS platform technology for the development of cloud services which interact with external software. The technology consists of two main stages, and detailed features description is given for each of them. Usage example of the presented technology in solving problems of transport modeling is given.

Keywords 1

development technology, service, cloud platform, multi-agent system, remote interaction

1. Introduction

The development of cloud technologies has led to the active use of service platforms for software development (PaaS[1]) [1-3]. Cloud platforms have a number of certain advantages, including the following: absence of need to provide software packages for various operating systems and hardware, ease of maintenance, making of team development easier, expansion of the potential users range, etc. Still non-SaaS[2] applications have not lost their relevance. On the one hand, cloud analogues are not created for existing desktop software (for various reasons). On the other hand – development of software for supercomputer platforms and high-performance hardware (for processing large amounts of data, or for performing calculations with high computational complexity) is still relevant.

Study of possible ways of combination for such different-type systems reveals the existence of hybrid distributed computing on heterogeneous architectures approach [4]. This area is constantly evolving and the work of its standardization is underway. In particular, the OGF organization has developed an Open interface standard for cloud computing (OCCI) [5]. It describes interaction at the cloud level and is suitable for IaaS, PaaS, and SaaS service delivery models [6]. However, those standards and interfaces are not suitable for combining cloud platforms with non-SaaS software. We must also mention the heterogeneous applications interaction technologies implemented in workflow management systems for composite applications in distributed computing environments (Taverna [7], Pegasus [8], Weka4WS [9], DVega [10], DIS3GNO [11], etc [12]). For the most part, these are

VI International Conference Information Technologies and High-Performance Computing (ITHPC-2021), September 14–16, 2021, Khabarovsk, Russia

EMAIL: gribova@iacp.dvo.com (A. 1); fleo@iacp.dvo.ru (A. 2); philipmm@iacp.dvo.ru (A. 3); vadim@dvo.ru (A. 4)
ORCID: 0000-0001-9393-351X (A. 1); 0000-0002-2049-2570 (A. 2); 0000-0002-0509-9471 (A. 3); 0000-0002-1314-7656 (A. 4)

[1] Platform as a Service
[2] Software as a Service

applications for scientific activities support, which make it possible to simulate various complex experiments. Some of the differences between such systems are: the types of software units which can be utilized (batch applications, web-services, local and remote command execution) and the mechanisms for transferring data between them; types of supported computing resources (web-servers, grid, cloud); mechanisms for embedding existing software as depended program units.

The paper presents a scheme for organizing system interaction when cloud platform services should be expanded with the functionality of external software. The description of the IACPaaS platform [13] technology for the creation (basing on this scheme) of such cloud services is also given.

2. Organizing of cloud service interaction with external software

By *external software*, we shall consider software that is capable of solving some applied problem and is not part of the *cloud platform*, but interaction with which is necessary during the *service* execution. External software can be a software system of any level of complexity, being run, for example, on a supercomputer or another cloud platform.

To implement the interaction of cloud platform services and external software, it is necessary to bring them to a common level of data exchange. For this purpose, the platform must provide software application developers with the ability to send data from their services to external software and receive its responses by some *protocol*. This can be initially available (as part of the programming languages used to create services), or must be additionally provided by the platform developers, if due to the development technology used (for example, due to selected programming languages or security restrictions) such functionality is not available or prohibited. In the latter case, the requirement for a cloud platform is to use the mechanism for building a service as a multi-component application.

Then the task of the platform developers is to place such a component in its toolkit – *gateway* (with application programming interface (API)) that can be included in the service for organizing data exchange. Also the formats for sent and received data, including technical parameters like system addresses, processing status and transferred data characteristics, should be provided.

With the capabilities described, service developers create their application from two groups of software components: those that are located on the platform side (service, for which the gateway is used as a component) and those that work remotely (*remote system*). The second group includes the external software mentioned above, as well as an *intermediary program* that can communicate with the gateway (according to the given protocol) and call the external software. In some cases, the external software may already have an implementation that includes certain mechanisms (not related to the solution of the original applied problem) for exchanging data with other systems and thus are used as an intermediary program.

Let us note the fundamental role of the initial choice (by the platform developers) of the protocol (protocols) that the gateway and the intermediary program will use to "communicate". Since cloud platforms are accessible via the Internet, the selection should be one of the generally accepted Internet protocols: HTTP (Hyper Text Transfer Protocol), FTP (File Transfer Protocol), TELNET, etc. Their usage can give the platform developers an opportunity to count on a large number of libraries that they can use to implement the gateway, and later service developers – for implementing intermediary programs. Rare or proprietary protocols can significantly reduce the readiness of service developers to implement (or use) them in the intermediary program. When choosing a protocol, it must be kept in mind what kind of interaction will be provided with its help between the gateway and intermediary programs – synchronous or asynchronous.

Based on the chosen interaction protocol(s) and the developed gateway along with format of communicating with it (if this stage was performed), platform developers should introduce a technology for development of services which interact with external software and provide it to all interested parties in the form of documentation and updated toolkit. The conceptual architecture of such interaction is shown in Figure 1.

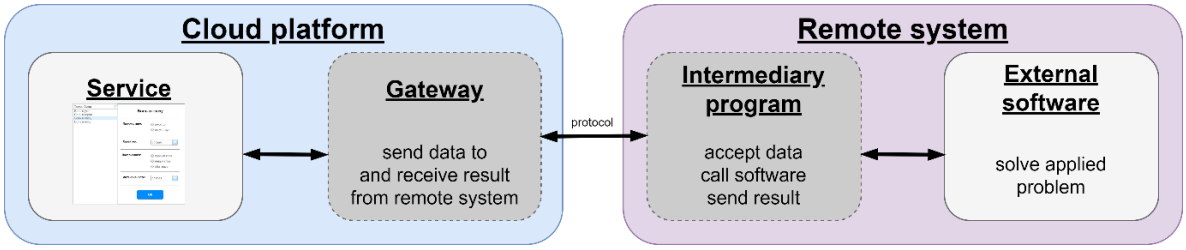

Figure 1: Remote interaction scheme

3. Interaction of the IACPaaS platform services with external software

3.1. Modification of the platform to support interaction of services with external software

The IACPaaS platform [13] supports three cloud service delivery models – PaaS, SaaS, and DaaS (Desktop as a Service) and is designed for AI systems engineering through DevOps practices. It supports the development, control and remote use of the applied intelligent services – systems with knowledge bases for various classes of problems and domains (decision support systems, intelligent assistants, etc.), including cross-domain solutions. The IACPaaS platform uses problem solvers to construct services. A problem solver is a set of agents that exchange messages, which are formed by templates. Agents within their production blocks process data and knowledge to solve the problem.

Currently, most of the platform's services operate interactively, so among the listed (in section 2) protocols the HTTP-request mechanism is suitable for instant data exchange and direct activation of the intermediary program [14]. Thus, the format of the sent information is limited to HTTP-requests containing named string parameters and binary files, and the format of the received information is HTTP-response with binary data (with its length and MIME type). Also, the choice of this interaction format simplifies the implementations of the gateway and of the intermediary program – due to the wide availability of software components (both in the form of source code and as compiled libraries for different platforms) that implement the functionality of the HTTP-client and HTTP-server.

Based on this choice, platform developers have extended its toolkit by placing a new gateway software component (in this case, an agent) that will be used in applied services (which solvers consist of agents) to interact with remote systems (being HTTP-client). This specialized agent "*External request agent*" encapsulates all the necessary functionality for sending HTTP-requests and receiving responses (GET and POST methods are implemented). It communicates with the service agents using the messages formed by specially introduced templates: "*Data for an external request template*" and "*Response to an external request template*". This agent and message templates are placed in the Fund of the platform (section "*IACPaaS Platform / HTTP-interaction with web-servers*"), from where service developers can use them. Code of classes, which represent the procedural parts of these message templates, encapsulates the methods for processing the content of messages created using these templates (forming of sent and reading of received data).

Figure 2 demonstrates the architectural scheme of remote interaction of the IACPaaS cloud platform service with external software. The arrows indicate the direction of data transfer between software components.

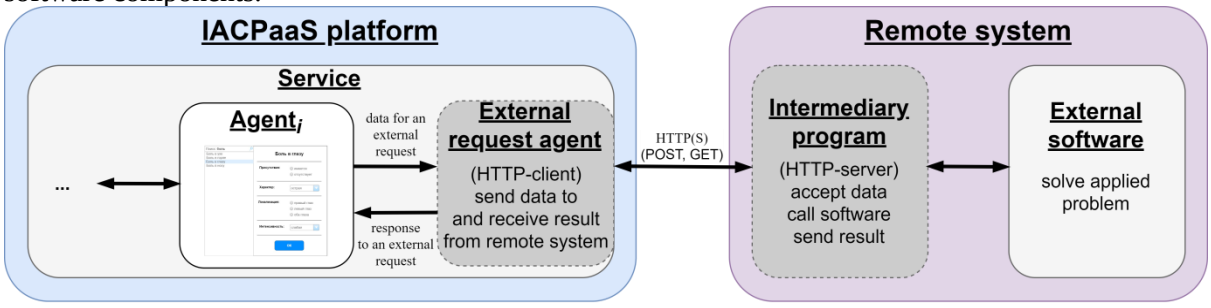

Figure 2: Remote interaction scheme for the IACPaaS platform

3.2. Technology for the development of services interacting with external software

3.2.1. General information

One of the development stages is the creation of a service on the IACPaaS platform. It should be is used at least to enter/select data for external software and to display the result of its operation. The service includes the described specialized agent, with which data is exchanged via messages formed by listed templates.

The remote system is formed on the basis of developed or existing external software. Herewith, developers must have the opportunity to host it on their own or public server, available over the Internet. In some cases, the external software can work as a service and be available over HTTP "out of the box". If there is no such mode of operation, then the recommended way of creating the remote system is to expand the functionality of the external software by including code that acts as a web-server (capable of processing HTTP-requests with GET and POST methods). This new part of the source code is the intermediary program in this case. This method should be used if the developers can change the source code of the external software and its implementation languages have means (libraries) for implementing the web-server functionality.

If this condition is not met (or if external software source code language does not support processing of HTTP-requests), you should develop a separate intermediary program as a wrapper over the external software. It must be able to handle HTTP-requests, call the external software, pass requests data to its input, and transmit received results as HTTP-responses. This can be done if the external software, when started (from another application), is able to accept passed parameters and their values or files as input data and put its result to specified files (or output streams).

Thus, in general, the IACPaaS platform technology for the development of multi-agent intelligent services which interact with external software consists of the following stages:

A) development of a cloud service on the platform by the basic or some other technology (with inclusion of the gateway agent into its solver),

B) development of a remote system:

B.1) development of external software (or adaptation of the existing one),

B.2) development of the intermediary program.

In this case, (B) may precede (A), but it is important that in both cases the result of the first stage completed will be some preliminary fixation of the language of "communication" of the two systems:

- either it is at first fixed what data the cloud service can send and what data it should receive, and then the remote system is implemented under this set of requirements,
- or at first it is fixed what data is needed for the external software and what data it produces, and then the service and the intermediary program are implemented under this set of requirements.

Note that at the second stage of development the "communication" language can be refined.

3.2.2. Cloud service development

The creation of the cloud service is performed in accordance with one of the technologies provided by the IACPaaS platform [15, 16] and consists, in general, of the following stages:

A.1) development of information resources used by the problem solver;

A.2) development of the service problem solver;

A.3) development of problem solver agents (consisting of a non-empty set of production blocks);

A.4) development of new message templates

A.5) development of the service user interface.

The act of interaction of the developed problem solver agent with external software consists of two steps – sending data to the agent "*External request agent*" (included in the solver) and receiving results from it. Accordingly, the special requirement for the development of the former is that its set of production blocks must necessarily contain:

1. a production block, the set of outgoing message templates of which contains "*Data for an external request template*" (in order to create and send messages to the "*External request agent*");
2. a production block, the execution of which is initiated by messages formed by the message template "*Response to an external request template*" (for accepting response messages from agent "*External request agent*" after it was addressed from the previously described production block and performed the interaction with the remote system).

The "*Data for an external request template*" describes the format of request and data transmitted to a remote system. The IACPaaS platform API provides the *ExternalRequestDataMessage* class to work with messages formed by this template. The developer is provided with the following methods:

- *create*(String *url*) – create a message by this template (leads to creation of an object that represents it in RAM and the information resource that will hold its data in the storage) with the specification of url address of the remote system (which must explicitly specify the protocol used for access – http or https) and automatic setting of token and communication method to POST[3];
- *addParam*(String *name*, String *value*) – add a parameter with the specified name and value to put this data in the request, which the "External request agent" will send to the remote system;
- *addFile*(String *filename*, Blob *data*) – add file with the specified filename and content (binary data in BLOB format) to include this data in the request;
- *addInforesource*(IInforesource *inforesource*, String *paramOrFileName*, boolean *asFile*, boolean *isUniversalJsonFormat*) – add the information resource (*inforesource*), which content represented in JSON format will be included in the request, with setting of the following:
  - *isUniversalJsonFormat* – a logical attribute whose value determines which of the two JSON formats supported by the IACPaaS platform should be used to represent an information resource – universal (true) or ontology-oriented (false),
  - *asFile* – a logical attribute whose value determines how the data transfer should be performed – as contents of the file (true), or as a value of a parameter (false),
  - *paramOrFileName* – the name of the parameter or file (in accordance with *asFile* parameter);
- *getToken*() – get the token (integer number) assigned to the message, which can later be used to identify the response corresponding to the request generated by this message.

The "*External request agent*" is always automatically set as destination for messages generated by the described template. It is not necessary to specify parameters, files, and information resources in case no such data is needed for work of the external software. Inclusion of files and information resources to the request is allowed only if the communication method is POST.

Thus creating a request message and filling it with data may look like this:

```
ExternalRequestDataMessage msg =
rc.externalRequestDataMessage.create("http://site.ru/compute.php", false);
// URL is specified, POST method is chosen, message object acquired
msg.addParam("param1", "value1"); // param1 with value1 is added
Blob data2 = ...; // some BLOB data is acquired
msg.addFile("file2", data2); // file file2 is added with contents of data2
msg.addInforesource(getInputs()[3], "knowledge.base", true, false);
// fourth input information resource is added as a file named "knowledge.base"
// containing its ontology-oriented JSON format representation
```

After work of the production block, which forms the message by "*Data for an external request template*", it will be sent to the agent "*External request agent*" at some point in time – according to the scheme of the multi-agent system operation on the IACPaaS platform. Upon receiving of this message the "*External request agent*" will form (using the message data) a standard HTTP-request. The intermediary program of the remote system will receive this request, pass data to external software which will process it and form an answer. It will be returned by intermediary program to the "*External request agent*", which will then forward it in a message (created by the message template "*Response to an external request template*") to the agent which has initiated this communication.

Next, you need to ensure that this agent can receive and process such messages. To do this, the second production block mentioned above is used, where response data is extracted and processed. The class *ExternalRequestReplyMessage* of the IACPaaS platform API should be used for that. It provides developers with the following methods:

[3] There is an extended version of this method – *create*(String *url*, boolean *asGet*, long *token*), which allows you to set the communication method by *asGet* parameter: true – GET request method, false – POST request method, and to set the integer ID of the request.

- *getStatus*() – get request execution status (a stringed message indicating the success of the request execution – "OK" or indicating the occurred error);
- *getContentType*() – get response content type (a stringed name of MIME type that allows you to determine how to process the response data);
- *getContent*() – get contents of the response (binary data, which should be interpreted according to the MIME type);
- *getToken*() – get the token (integer number) that was assigned to the initial message with the request data (it can be used to identify the original request, if the agent sends multiple requests and receives multiple responses).

Thus extracting the response data from message may look like this:

```
public void runProduction(ExternalRequestReplyMessage msg, …)
{ If ("OK".equals(msg.getStatus())
  { Blob content = msg.getContent();
    switch (msg.getContentType())
    { case "text/plain":
        String resultString = content.toStringUTF8();
        //...
      default:
        byte[] resultData = content.getBytes();
        // ...
} } }
```

3.2.3. Remote system development

The easiest way to create a remote system is to extend the external software source code (or use the capabilities of the installed product) to support the functionality of the web-server. In this case, the external software includes an intermediary program that processes incoming requests – see Figure 3.

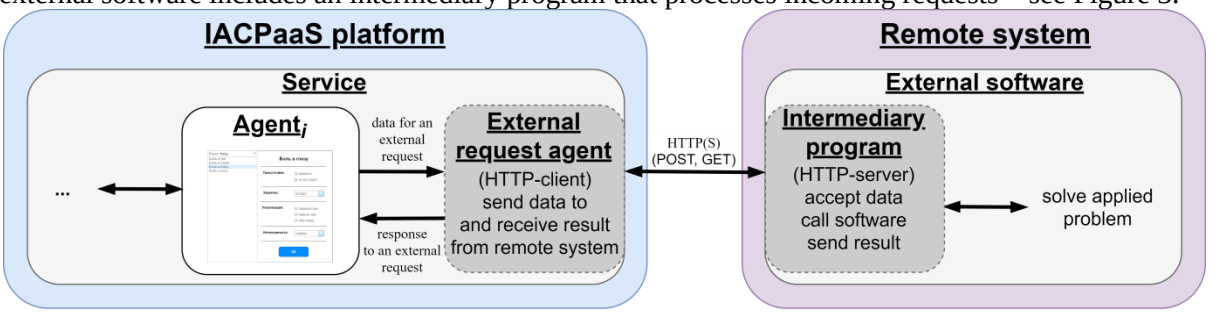

Figure 3: Simple case of remote interaction scheme for the IACPaaS platform

Let's consider a more general case when the external software does not have the functionality of a web-server and cannot be extended to support it (due to language limitations or absence of source code). An important requirement for such external software is the ability to be run by other software (that is connected to the Internet. As an intermediary program in this case, you need to use a bundle of: a web-server program, some CGI / FastCGI gateway for certain programming language and an application on this language. Such an application (a software wrapper) should implement the launch of external software with the transfer of data received from the web-server, and return the results to the web-server.

Of the existing software tools a web-server (for example, Apache HTTP Server, Microsoft IIS, nginx, etc) and a CGI/FastCGI gateway for some programming language are selected. Then they are installed and integrated on the node accessible over Internet. (In the case of using a third-party hosting this software may be pre-installed and already bound.) The external software can be installed on the same node – to be run locally, or on the other one (accessible from the network of the original one) – for a remote call over, for example, the SSH protocol. When choosing a CGI/FastCGI implementation, it is proposed to focus on the languages used in web-programming (e.g. PHP, Perl), because they usually implement high-level methods for processing HTTP requests. The resulting interaction scheme is shown in Figure 4.

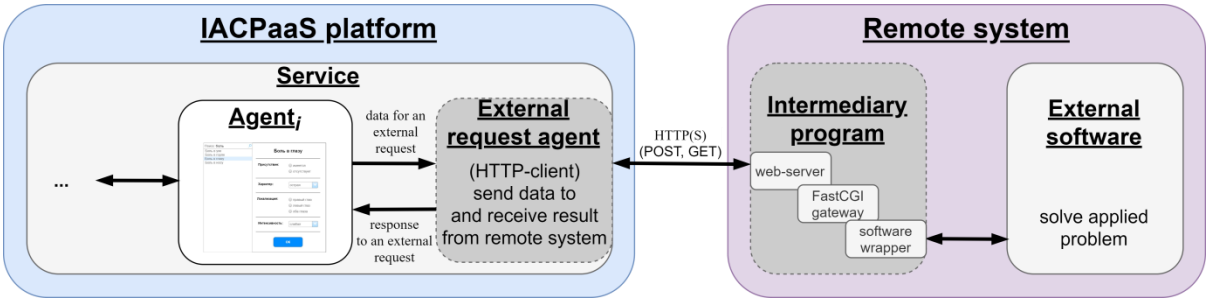

Figure 4: Complex case remote interaction scheme for the IACPaaS platform

Let's list the tasks sequentially executed by software wrapper.

1. Accept the request from the IACPaaS platform service received via the web-server. The IACPaaS platform service sends requests over the HTTP(S) protocol using the POST and GET methods. Thus, in the intermediary program, you need to get data over these protocol and method. Here is an example of getting a request parameter and file in PHP:

```
$param1 = $_REQUEST['reqParam']
$file1 = $_FILES['reqFile']
```

where:

- *$param1* – the variable for storing parameter value;
- *reqParam* – the name of the parameter which value you want to get (the name of the parameter must match the one passed in the request from the IACPaaS platform service, there can be any number of parameters);
- *$file1* – the variable for storing information about received file;
- *reqFile* – the name of the file which you want to get.

2. Call the external software and pass to it the parameters obtained from the received request. Data exchange with external software can be organized through standard I/O streams and files. E.g., for calling an executable program file in PHP the `exec(...)` function can be used.

3. Get the result of the external software work. Some common ways to implement this are:

- to use standard output streams (intercepted by the software wrapper);
- to redirect them to files (via external software execution command in the software wrapper);
- to write the result of the work to files (databases) that the software wrapper has access to.

4. Send the received result (possibly, with some post-processing) to the web-server (which will forward it back to the IACPaaS platform service). Depending on what is developed first, the sent data is either adjusted to the previously defined (when developing the IACPaaS platform service) format, or such format is set now – when developing the remote system (including the case when data generated by the external software is unchanged) and then used when developing the service. There are no special requirements for the response format: this can be an arbitrary text, binary data, formatted data, files, etc. Here is example (in PHP) of transferring the result of the external software work obtained by intercepting standard output stream:

```
$output=null;
$retval=null;
exec('external_soft.exe', $output, $retval);
echo "STATUS=$retval\n" . "RESULT:\n";
print_r($output);
```

4. Usage example: Service for interactive simulation of interregional trade

The described technology has been applied in project of interactive simulation of interregional trade [17]. Here, the main computational load is put on external software, and the IACPaaS platform hosts services for visualizing the results of these computations. The service is designed to simulate trade flows between regions and determine the most likely characteristics for communication systems in conditions of incomplete information. The mathematical model of the problem belongs to the class of convex problems of mathematical programming, assumes the numerical solution of a nonlinear optimization problem with linear constraints and is implemented on a high-performance server. The graphical interface of the service allows user to change the communication parameters and to view the

resulting changes in the flow of goods between cities (see Figure 5). Interactive editing of the graph, which represents the scheme of communications between cities of several regions, involves changing the parameters of both its vertices and arcs. By clicking on the corresponding graph element, the user receives structured information about the object, which can be changed. Using the resulting graph, the user can add a map-view to it to evaluate the visualization results more accurately.

In accordance with the presented technology, service development consisted of two stages.

4.1. Stage A. Service development on the IACPaaS platform

A.1) Development of information resources. No knowledge or data bases were required.

A.2) Development of the service problem solver. At this stage, the information resource was created which represents the declarative specification of the problem solver. The "*Transport Model. Interface agent*" was set as the *interface controller* and as the *root* agents, the name and content for "*Transport Model*" web-page were set and it was specified as the start one (see Figure 6). The other elements (input, output, native and temporary information resources) were specified as empty sets.

Figure 5: Visualization of transport communications

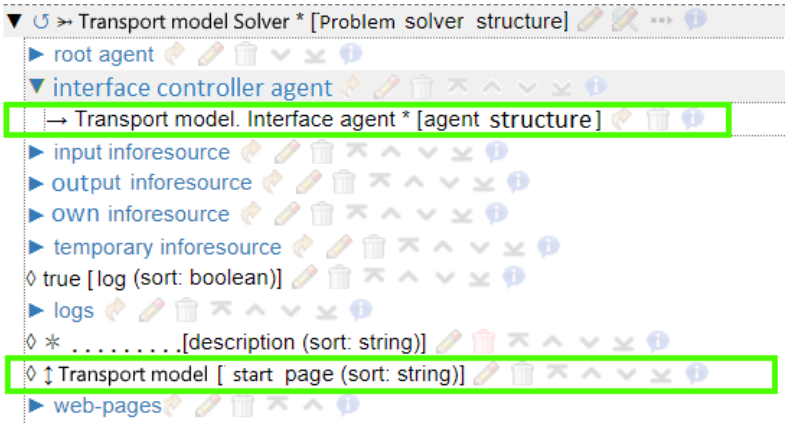

Figure 6: The problem solver of the "Transport Model" service

A.3) Development of agents. At this stage, an information resource was created that represents a declarative specification of the single developed agent for the problem solver. It contains two substantial production blocks (see Figure 7). The first one is designed to respond to user actions and send the entered data to the "*External request agent*". It contains the following ready-to-use (platform) message templates: template for incoming messages (received from the system agent) – "*Request from View agent template*" and templates for outgoing messages: "*Show window template*", "*Return string template*" and "*Data for external request template*". The second production block of is intended for obtaining the results of data processing and their visualization. It contains the following ready-to-use (platform) message templates: template for incoming messages (received from the system agent "*External request agent*") – "*Response to external request template*" and templates for outgoing messages: "*Show window template*", "*Return string template*". This production block builds the user interface with the help of Java Script library.

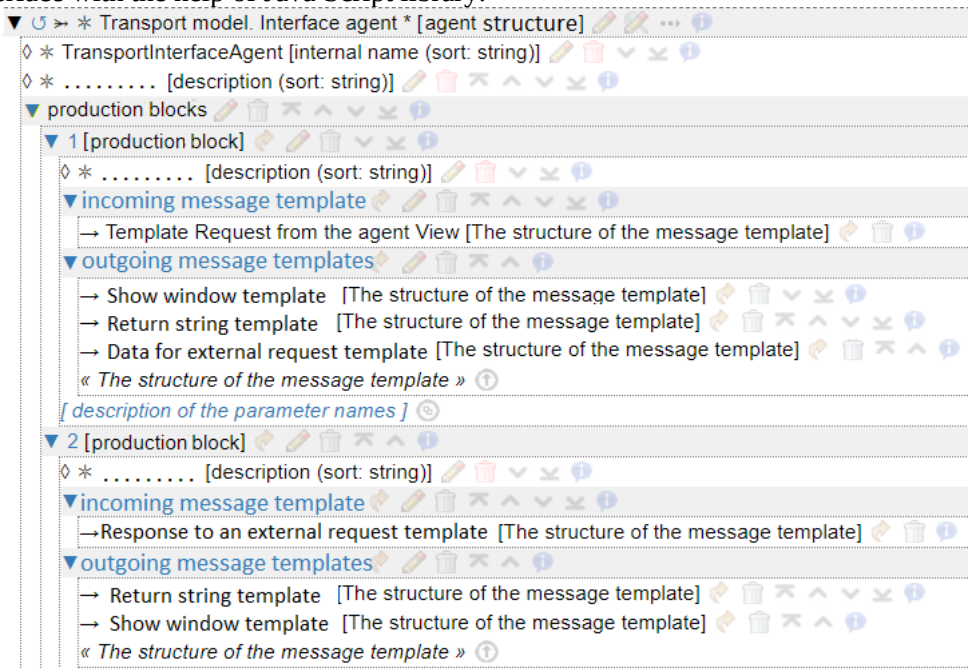

Figure 7: Declarative specification of the agent "Transport model. Interface Agent"

A.4) Development of message templates. No new message templates ware required.

A.5) Development of the user interface of the service. The user interface of the service (graph visualization and modification) was implemented as Java Script library and was bound to the developed agent.

4.2. Stage B. Remote system

B.1) External software. The external software is a program (in MATLAB programming language) running on a computing cluster that calculates the values of the characteristics of communication systems on the basis of the corresponding mathematical model.

B.2) Intermediary program. The Apache HTTP Server 2.2.29 (Linux/SUSE) was installed on the remote system for use as a web-server. PHP interpreter was used as a FastCGI gateway. Its installation and binding with Apache HTTP Server was performed in accordance with manual. The PHP script (software wrapper) implements the functions of receiving two types of requests: to get the initial values of model parameters and to get their changed values. The interaction between the intermediary program and the external software is done by writing and reading shared files.

5. Conclusion

The considered mechanisms for linking of cloud platforms with non-SaaS software were used for creation of the technology for hybrid service development on the IACPaaS platform. The described technology was used for the development of services solving the problems of transport modeling.

Currently, the proposed technology is used for city traffic modeling [18] and to solve some problems in the field of medicine. Particularly, intelligent electronic health record service is being developed. The UI processing and data storage are performed on the IACPaaS platform, and risk indicators for some pathological conditions are calculated using requests to external Python software.

6. Acknowledgements

The work was partially supported by Russian Foundation for Basic Research, project Nos. 19-07-00244 and 20-07-00670.

7. References

[1] T. V. Batura, F. A. Murzin, D. F. Semich, Cloud technologies: basic models, applications, concepts and trends of development, Software & systems 3 (2014) 64–72.
[2] A. Medvedev, Cloud technologies: trends of development, examples of execution, Contemporary Technologies in Automation 2 (2013) 6–9.
[3] G. V. Rybina, Intelligent Systems: from A to Z. Series of monographs in three books // Book 3. Problem-specific intelligent systems. Instrumental tools for building intelligent systems, OOO «Nauchtekhlitizdat», Moscow, 2015.
[4] V. G. Bogdanova, S. A. Gorskiy, A. A. Pashinin, Service-oriented tools for solving Boolean feasibility problems, Fundamental research 2-6 (2015) 1151–1156.
[5] Open Cloud Computing Interface (OCCI) URL: http://occi-wg.org/community/.
[6] T. F. Wallace. IaaS, PaaS or SaaS: what's the difference and which one suits you best // URL: https://itglobal.com/company/blog/iaas-paas-saas/.
[7] T. Oinn, M. Addis, J. Ferris, et al., Taverna: a tool for the composition and enactment of bioinformatics workflows, Bioinformatics 20(17) (2004) 3045–3054.
[8] E. Deelman, G. Singh, M. H. Su, et al., Pegasus: A framework for mapping complex scientific workflows onto distributed systems, Scientific programming 13(3) (2005) 219–237.
[9] M. Lackovic, D. Talia, P. Trunfio, A framework for composing knowledge discovery workflows in grids, Foundations of computational intelligence 6 (2009) 345–369.
[10] R. Tolosana-Calasanz, J. A. Bañares, P. Álvarez, et al., An uncoordinated asynchronous checkpointing model for hierarchical scientific workflows, J. of Computer and System Sciences 76(6) (2010) 403–415.
[11] E. Cesario, M. Lackovic, D. Talia, and P. Trunfio, Service-oriented data analysis in distributed computing systems, in: High Performance Computing: From Grids and Clouds to Exascale, I.

Foster, W. Gentzsch, L. Grandinetti, and G. Joubert, Eds., IOS Press, Lansdale, Pa, USA, 2011, pp. 225–245.
[12] D. Talia, Workflow Systems for Science: Concepts and Tools, International Scholarly Research Notices 2013 (2013).
[13] V. Gribova, A. Kleschev, Ph. Moskalenko, V. Timchenko, L. Fedorischev, E. Shalfeeva, The IACPaaS cloud platform: features and perspectives, IEEE Xplore (2017) 80-84. doi:10.1109/RPC.2017.8168073.
[14] I. Grigorik, HTTP protocols, 1st edition, O'Reilly Media, Inc. (2017) URL: https://www.oreilly.com/library/view/http-protocols/9781492030478/.
[15] V. V. Gribova, A. S. Kleschev, D. A. Krylov, Ph. M. Moskalenko, V. A. Timchenko, E. A. Shalfeyeva, A base technology for development of intelligent services with the use of IACPaaS cloud platform. Part 1. A development of knowledge base and problem solver, Software engineering 12 (2015) 3-11. doi:10.17587/prin._12_2015_1.
[16] V. Gribova, A. Kleschev, Ph. Moskalenko, V. Timchenko, L. Fedorischev, E. Shalfeeva, E. Zamburg, Technology for the development of intelligent service shells based on extended generative graph grammars, IEEE Xplore (2018). doi:10.1109/RPC.2018.8482129.
[17] A. S. Velichko, V. V. Gribova, L. A. Fedorishchev, Cloud Service for Interactive Simulation of Interregional Trade // Automatic control and computer sciences. 53(7) (2019) 811–820. doi:10.3103/S0146411619070289.
[18] V. V. Gribova, N. B. Shamray, L. A. Fedorishchev, Traffic modeling flows in a developing urban infrastructure with a software suite for creating interactive virtual environments, Automation and remote control 78 (2) (2017) 235–246. doi:10.1134/S0005117917020047.