Article
Sergey Lukyanchikov · Apr 7, 2021 9m read

Distributed Artificial Intelligence with InterSystems IRIS

What is Distributed Artificial Intelligence (DAI)?

Attempts to find a " bullet-proof " definition have not produced result: it seems like the term is slightly " ahead of time " . Still, we can analyze semantically the term itself – deriving that distributed artificial intelligence is the same AI (see our effort to suggest an " applied " definition) though partitioned across several computers that are not clustered together (neither data-wise, nor via applications, not by providing access to particular computers in principle). I.e., ideally, distributed artificial intelligence should be arranged in such a way that none of the computers participating in that " distribution " have direct access to data nor applications of another computer: the only alternative becomes transmission of data samples and executable scripts via " transparent " messaging. Any deviations from that ideal should lead to an advent of " partially distributed artificial intelligence " – an example being distributed data with a central application server. Or its inverse. One way or the other, we obtain as a result a set of " federated " models (i.e., either models trained each on their own data sources, or each trained by their own algorithms, or " both at once ").

Distributed AI scenarios " for the masses "

We will not be discussing edge computations, confidential data operators, scattered mobile searches, or similar fascinating yet not the most consciously and wide-applied (not at this moment) scenarios. We will be much " closer to life " if, for instance, we consider the following scenario (its detailed demo can and should be watched here): a company runs a production-level AI/ML solution, the quality of its functioning is being systematically checked by an external data scientist (i.e., an expert that is not an employee of the company). For a number of reasons, the company cannot grant the data scientist access to the solution but it can send him a sample of records from a required table following a schedule or a particular event (for example, termination of a training session for one or several models by the solution). With that we assume, that the data scientist owns some version of the AI/ML mechanisms already integrated in the production-level solution that the company is running – and it is likely that they are being developed, improved, and adapted to concrete use cases of that concrete company, by the data scientist himself. Deployment of those mechanisms into the running solution, monitoring of their functioning, and other lifecycle aspects are being handled by a data engineer (the company employee).

An example of deployment of a production-level AI/ML solution on InterSystems IRIS platform that works autonomously with a flow of data coming from equipment, was provided by us in this article. The same solution runs in the demo under the link provided in the above paragraph. You can build your own solution prototype on InterSystems IRIS using the content (free with no time limit) in our repo Convergent Analytics (visit sections Links to Required Downloads and Root Resources).

Which " degree of distribution " of AI do we get via such scenario? In our opinion, in this scenario we are rather close to the ideal because the data scientist is " cut from " both the data (just a limited sample is transmitted – although crucial as of a point in time) and the algorithms of the company (data scientist ' s own " specimens " are never in 100% sync with the " live " mechanisms deployed and running as part of the real-time production-level solution), he has no access at all to the company IT infrastructure. Therefore, the data scientist ' s role resolves to a partial replay on his local computational resources of an episode of the company production-level AI/ML solution functioning, getting an estimate of the quality of that functioning at an acceptable confidence level – and returning a feedback to the company (formulated, in our concrete scenario, as " audit " results plus, maybe, an improved version of this or that AI/ML mechanism involved in the company solution).

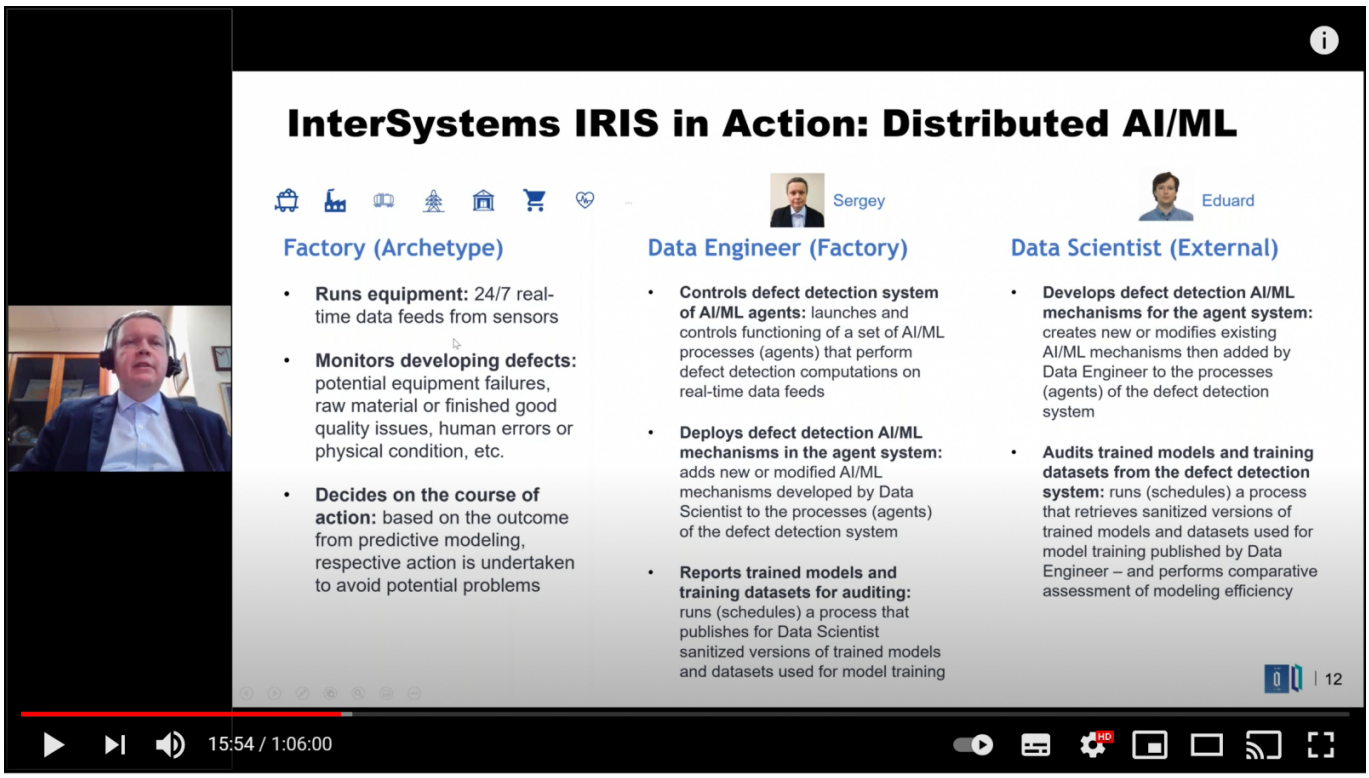

Figure 1 Distributed AI scenario formulation

We know that feedback may not necessarily need to be formulated and transmitted during an AI artifact exchange by humans, this follows from publications about modern instruments and already existing experience around implementations of distributed AI. However, the strength of InterSystems IRIS platform is that it allows equally efficiently to develop and launch both “ hybrid ” (a tandem of a human and a machine) and fully automated AI use cases – so we will continue our analysis based on the above “ hybrid ” example, while leaving a possibility for the reader to elaborate on its full automation on their own.

How a concrete distributed AI scenario runs on InterSystems IRIS platform

The intro to our video with the scenario demo that is mentioned in the above section of this article gives a general overview of InterSystems IRIS as real-time AI/ML platform and explains its support of DevOps macromechanisms. In the demo, the “ company-side ” business process that handles regular transmission of training datasets to the external data scientist, is not covered explicitly – so we will start from a short coverage of that business process and its steps.

A major “ engine ” of the sender business processes is the while-loop (implemented using InterSystems IRIS visual business process composer that is based on the BPL notation interpreted by the platform), responsible for a systematic sending of training datasets to the external data scientist. The following actions are executed inside that “ engine ” (see the diagram, skip data consistency actions):

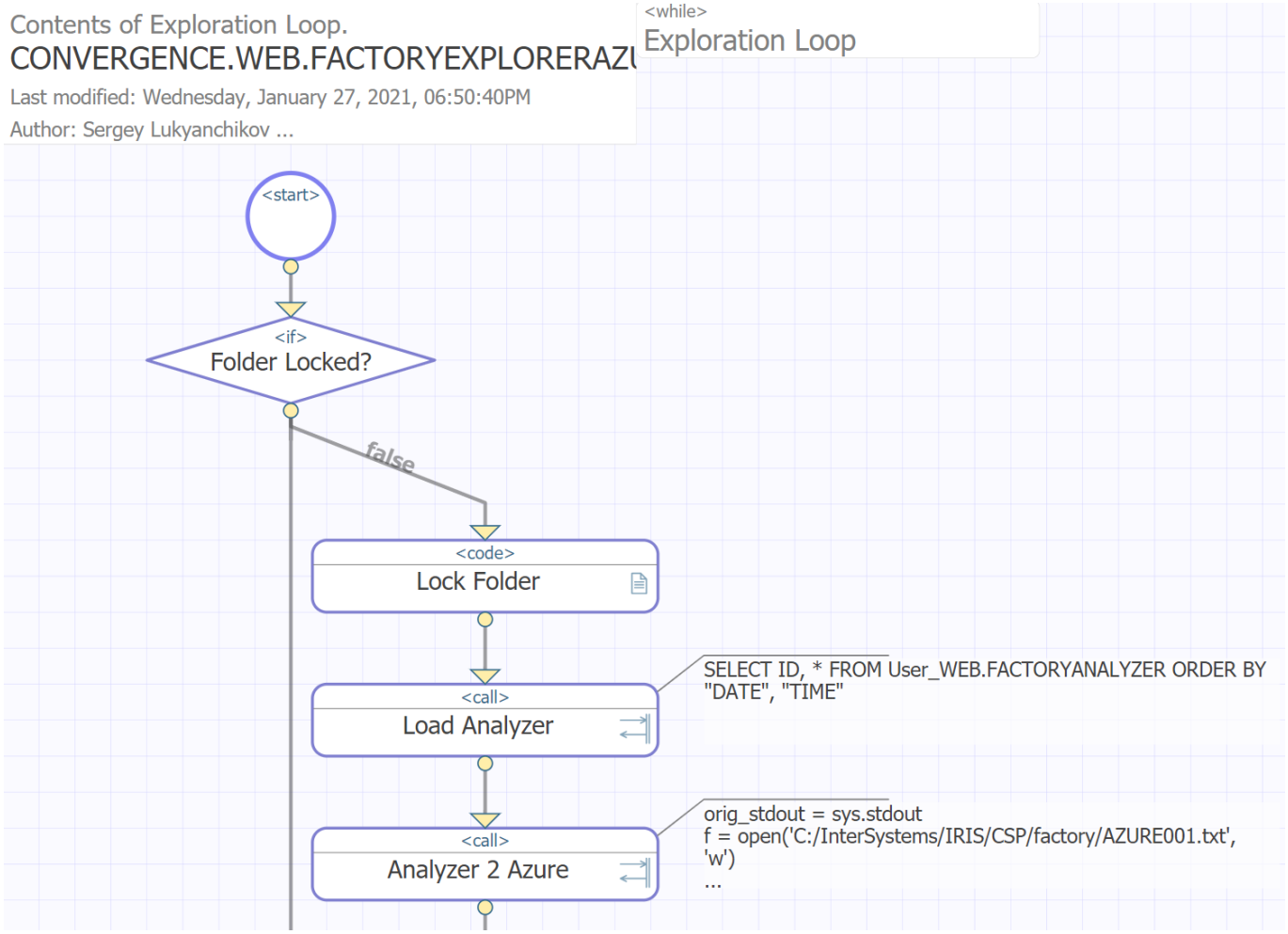

Figure 2 Main part of the " sender " business process

(a) Load Analyzer – loads the current set of records from the training dataset table into the business process and forms a dataframe in the Python session based on it. The call-action triggers an SQL query to InterSystems IRIS DBMS and a call to Python interface to transfer the SQL result to it so that the dataframe is formed;

(b) Analyzer 2 Azure – another call-action, triggers a call to Python interface to transfer it a set of Azure ML SDK for Python instructions to build required infrastructure in Azure and to deploy over that infrastructure the dataframe data formed in the previous action;

As a result of the above business process actions executed, we obtain a stored object (a .csv file) in Azure containing an export of the recent dataset used for model training by the production-level solution at the company:

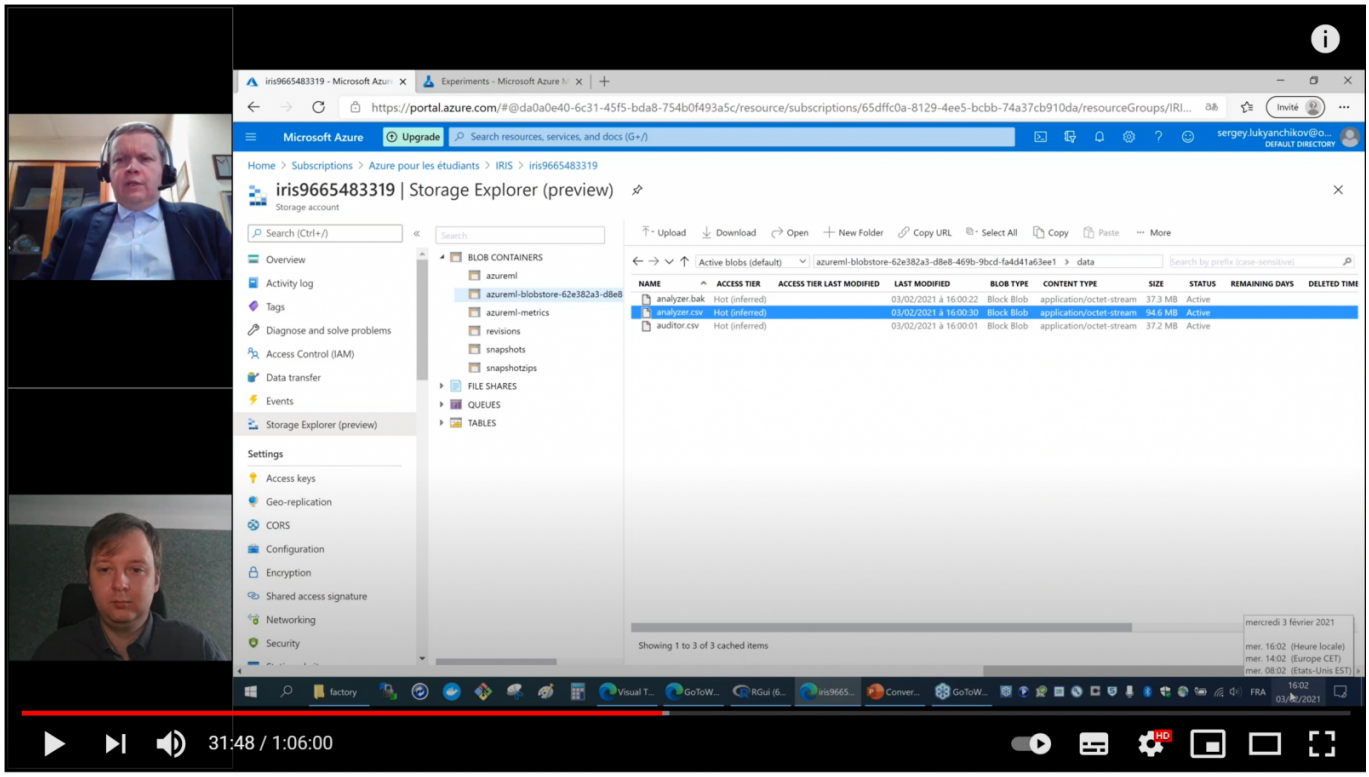

Figure 3 “ Arrival ” of the training dataset to Azure ML

With that, the main part of the sender business process is over, but we need to execute one more action keeping in mind that any computation resources that we create in Azure ML are billable (see the diagram, skip data consistency actions):

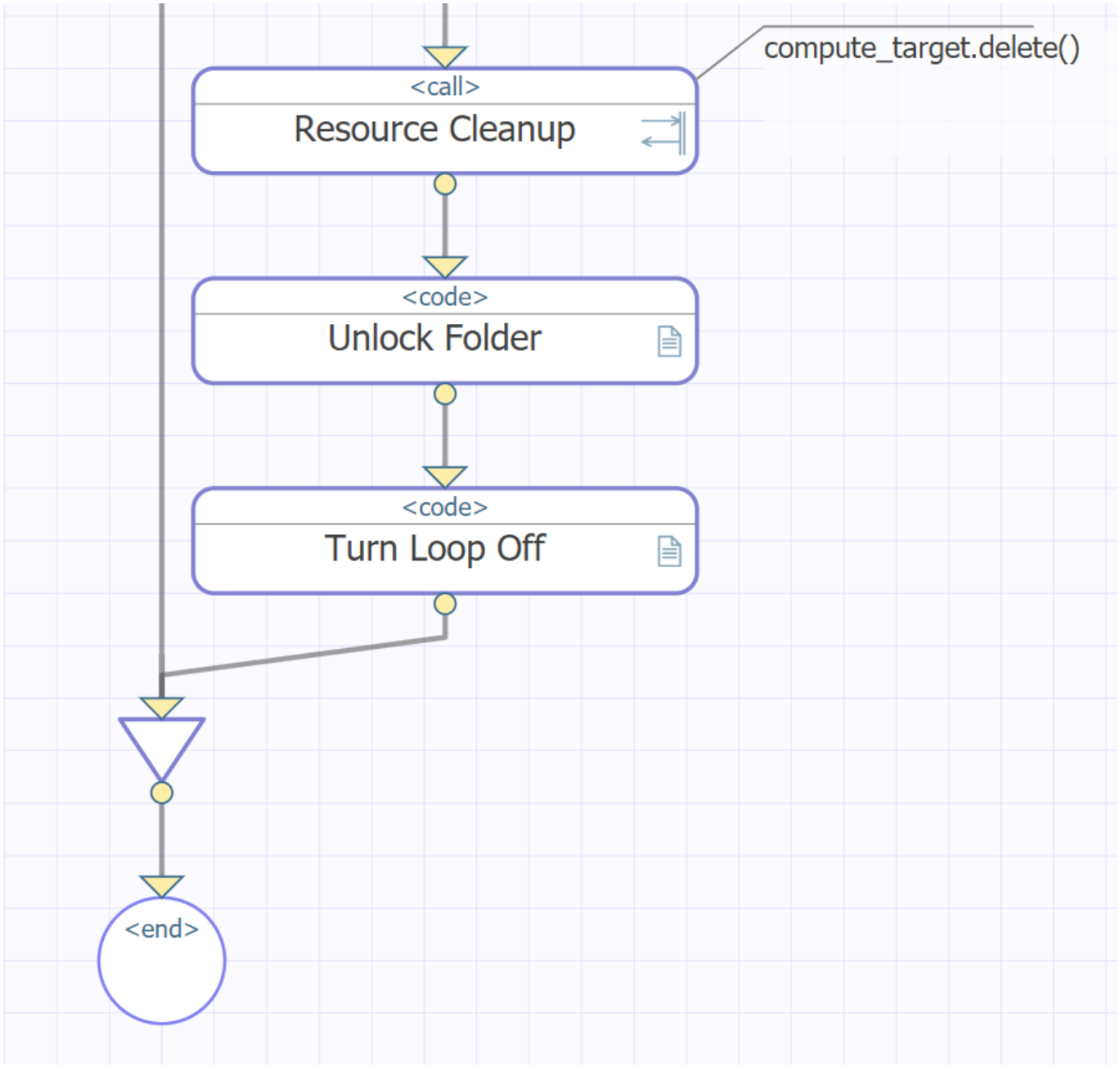

Figure 4 Final part of the " sender " business process

(c) Resource Cleanup – triggers a call to Python interface to transfer it a set of Azure ML SDK for Python instructions to remove from Azure the computational infrastructure built in the previous action.

The data required for the data scientist has been transmitted (the dataset is now in Azure), so we can proceed with launching the " external " business process that would access the dataset, run at least one alternative model training (algorithmically, an alternative model is distinct from the model running as part of the production-level solution), and return to the data scientist the resulting model quality metrics plus visualizations permitting to formulate " audit findings " about the company production-level solution functioning efficiency.

Let us now take a look at the receiver business process: unlike its sender counterpart (runs among the other business processes comprising the autonomous AI/ML solution at the company), it does not require a while-loop, but it contains instead a sequence of actions related to training of alternative models in Azure ML and in IntegratedML (the accelerator for use of auto-ML frameworks from within InterSystems IRIS), and extracting the training results into InterSystems IRIS (the platform is also considered installed locally at the data scientist ' s):

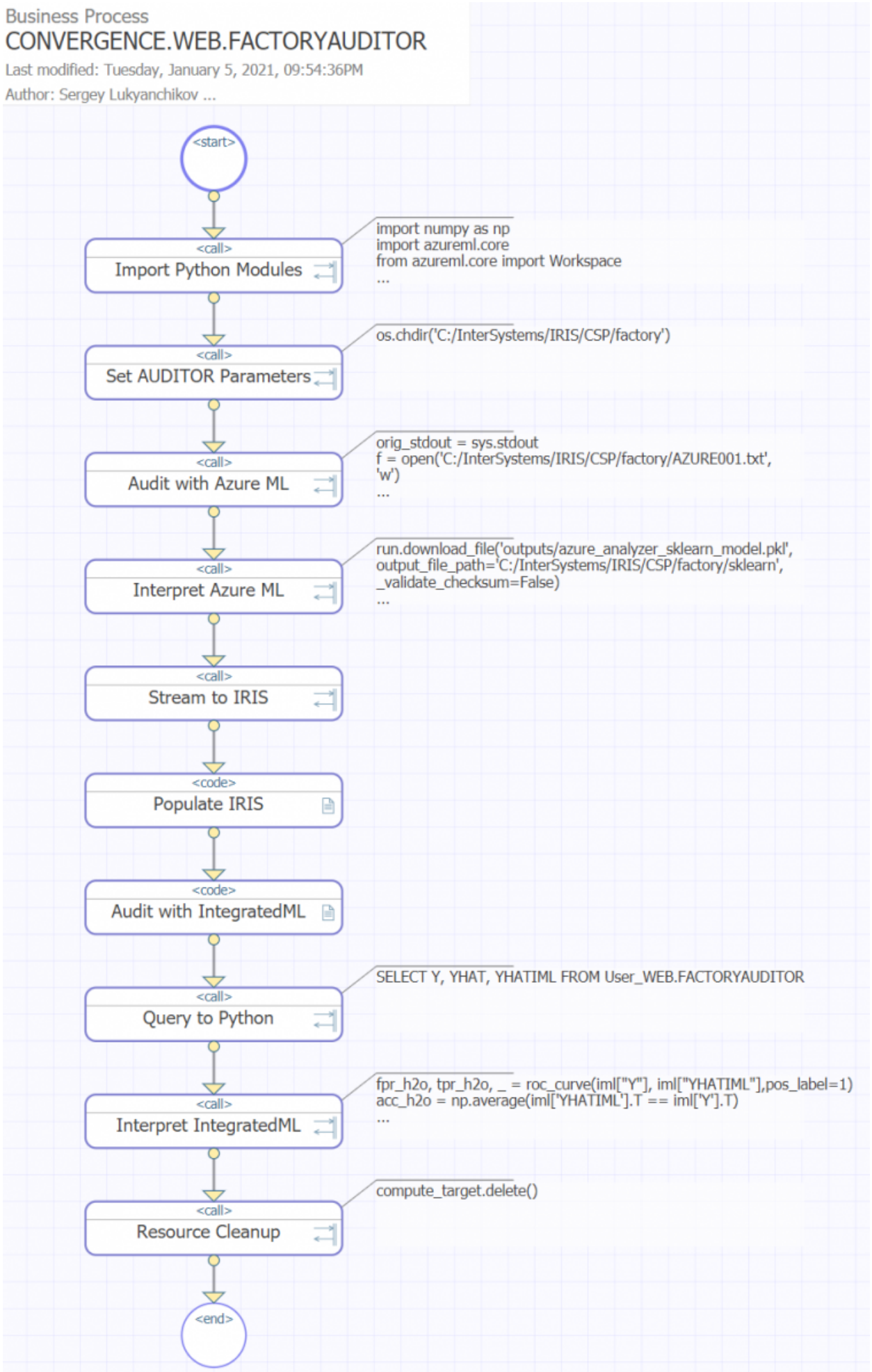
Business Process
CONVERGENCE.WEB.FACTORYAUDITOR
Last modified: Tuesday, January 5, 2021, 09:54:36PM
Author: Sergey Lukyanchikov ...
<start>
<call>
Import Python Modules
import numpy as np
import azureml.core
from azureml.core import Workspace
...
<call>
Set AUDITOR Parameters
os.chdir('C:/InterSystems/IRIS/CSP/factory')
<call>
Audit with Azure ML
orig_stdout = sys.stdout
f = open('C:/InterSystems/IRIS/CSP/factory/AZURE001.txt',
'w')
...
<call>
Interpret Azure ML
run.download_file('outputs/azure_analyzer_sklearn_model.pkl',
output_file_path='C:/InterSystems/IRIS/CSP/factory/sklearn',
_validate_checksum=False)
...
<call>
Stream to IRIS
<code>
Populate IRIS
<code>
Audit with IntegratedML
<call>
Query to Python
SELECT Y, YHAT, YHATIML FROM User_WEB.FACTORYAUDITOR
<call>
Interpret IntegratedML
fpr_h2o, tpr_h2o, _ = roc_curve(iml["Y"], iml["YHATIML"],pos_label=1)
acc_h2o = np.average(iml['YHATIML'].T == iml['Y'].T)
...
<call>
Resource Cleanup
compute_target.delete()
<end>

Figure 5 “ Receiver ” business process

(a) Import Python Modules – triggers a call to Python interface to transfer it a set of instructions to import Python modules that are required for further actions;

(b) Set AUDITOR Parameters – triggers a call to Python interface to transfer it a set of instructions to assign default values to the variables required for further actions;

(c) Audit with Azure ML – (we will be skipping any further reference to Python interface triggering) hands “ audit assignment ” to Azure ML;

(d) Interpret Azure ML – gets the data transmitted to Azure ML by the sender business process, into the local Python session together with the “ audit ” results by Azure ML (also, creates a visualization of the “ audit ” results in the Python session);

(e) Stream to IRIS – extracts the data transmitted to Azure ML by the sender business process, together with the “ audit ” results by Azure ML, from the local Python session into a business process variable in IRIS;

(f) Populate IRIS – writes the data transmitted to Azure ML by the sender business process, together with the “ audit ” results by Azure ML, from the business process variable in IRIS to a table in IRIS;

(g) Audit with IntegratedML – “ audits ” the data received from Azure ML, together with the “ audit ” results by Azure ML, written into IRIS in the previous action, using IntegratedML accelerator (in this particular case it handles H2O auto-ML framework);

(h) Query to Python – transfers the data and the “ audit ” results by IntegratedML into the Python session;

(i) Interpret IntegratedML – in the Python session, creates a visualization of the “ audit ” results by IntegratedML;

(j) Resource Cleanup – deletes from Azure the computational infrastructure created in the previous actions.

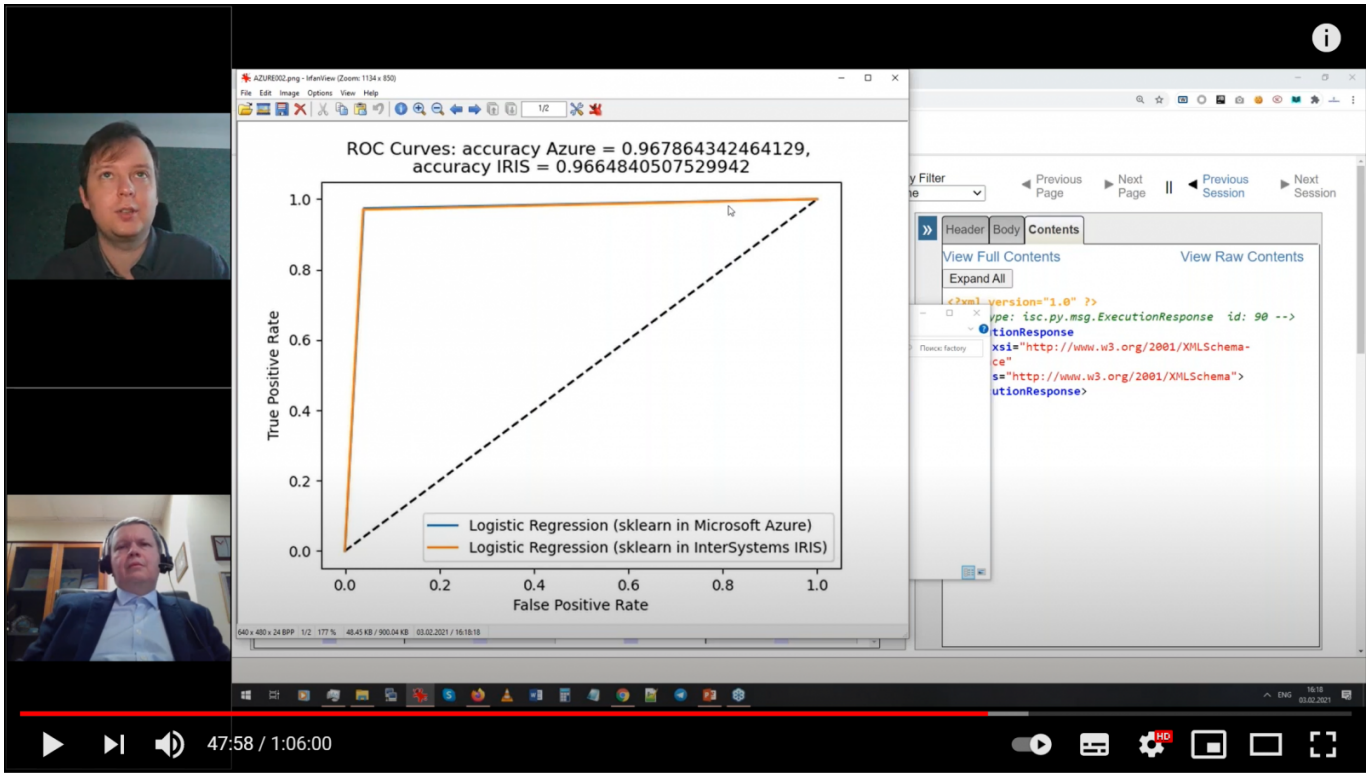

Figure 6 Visualization of Azure ML “ audit ” results

Figure 7 Visualization of IntegratedML "audit" results

How distributed AI is implemented in general on InterSystems IRIS platform

InterSystems IRIS platform distinguishes among three fundamental approaches to distributed AI implementation:

· Direct exchange of AI artifacts with their local and central handling based on the rules and algorithms defined by the user

· AI artifact handling delegated to specialized frameworks (for example: TensorFlow, PyTorch) with exchange orchestration and various preparatory steps configured on local and the central instances of InterSystems IRIS by the user

· Both AI artifact exchange and their handling done via cloud providers (Azure, AWS, GCP) with local and the central instances just sending input data to a cloud provider and receiving back the end result from it

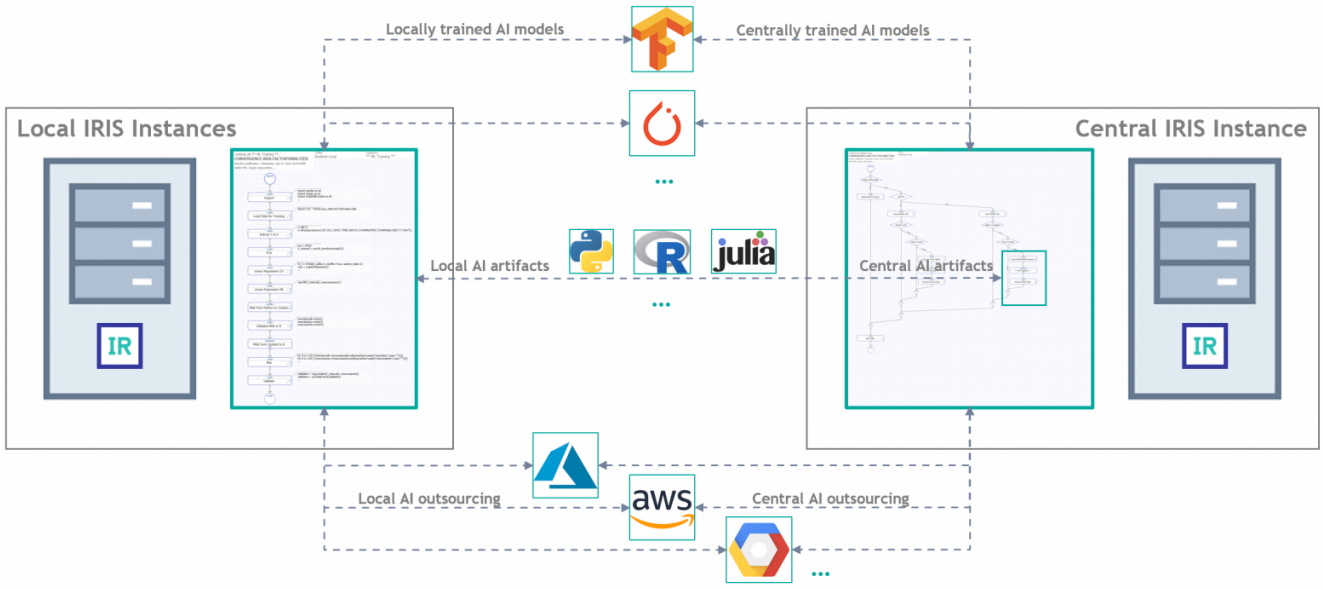

Figure 8 Fundamental approaches to distributed AI implementation on InterSystems IRIS platform

These fundamental approaches can be used modified/combined: in particular, in the concrete scenario described in the previous section of this article (“ audit ”), the third, “ cloud-centric ” , approach is used with a split of the “ auditor ” part into a cloud portion and a local portion executed on the data scientist side (acting as a “ central instance ”).

Theoretical and applied elements that are adding up to the “ distributed artificial intelligence ” discipline right now in this reality that we are living, have not yet taken a “ canonical form ” , which creates a huge potential for implementation innovations. Our team of experts follows closely the evolution of distributed AI as a discipline, and constructs accelerators for its implementation on InterSystems IRIS platform. We would be glad to share our content and help everyone who finds useful the domain discussed here to start prototyping distributed AI mechanisms. You can reach our AI/ML expert team using the following e-mail address – MLToolkit@intersystems.com.

#AI #Cloud #Convergent Analytics #Distributed Data Management #Machine Learning #InterSystems IRIS