Model Architecture for Automatic Translation and Migration of Legacy Applications to Cloud Computing Environments

(Scientific Report – Activity 1.3)

Andrei Ionut DAMIAN
Cloudifier SRL
Bucharest, Romania
damian@cloudifier.net

Nicolae TAPUS
Computer Science & Engineering Department
University POLITEHNICA of Bucharest
Bucharest, Romania
ntapus@cs.pub.ro

***Abstract*—On-demand computing, SaaS, PaaS, and in general Cloud Computing is currently the main approach by which both academic and commercial domains are delivering systems and content. Nevertheless there still remains a huge segment of legacy systems and application ranging from accounting and management information systems to scientific software based on classic desktop or simple client-server architectures. Although in the past years more and more companies and organizations have invested important budgets in translating legacy apps to online cloud-enabled environment there still remains an important segment of applications that for reasons have not been translated. This paper proposes an innovative pipeline model architecture for automated translation and migration of legacy application to cloud-enabled environment with a minimal software development costs.**

Keywords — automatic programming; cloud computing; migration; machine-learning; automatic translation

I. INTRODUCTION

Automated translation of legacy applications has always been an important research area for the computer science community with high economic impact especially in the area of small and medium enterprises. With each arrival of a new programming languages, frameworks or formal modelling languages computer scientist pursued the research and development of various code translators that would automatically migrate from legacy application source code to the new programming language. Most successful research and production projects over the years have been in the areas of automated code generation from formal modelling languages [1] [2] [3] [4] and in the area of automated code translation [5] [6] [7]. Nevertheless the process of source code migration has been always accompanied by re-engineering processes that were supposed to optimize the internal mechanics of the newly migrated software systems. Yet another objective of the software translation has been and still is the migration of the legacy application from a deprecated environment to a more modern, robust and reliable one – such as migrating from desktop based applications to client-server and then to Cloud Computing environments. However on multiple occasions due to the missing of legacy source code the actual migration has been forced by employing terminals or virtual desktops.

In our paper we propose an innovative model architecture designed for the challenges posed particularly by the application user interface and data flow re-engineering of legacy applications where the source code is **not** available. Our model will employ machine learning techniques and in doing so it will be able to "observe" and infer the functional behavior of the legacy application. At the end of the inferential process our pipeline model will be able to generate an intermediate scripted definition of the observed legacy application. Finally, the

ID Proiect: P_38_543, Nr. Ctr. 98/09.09.2016, MySMIS: 104349, Apel - POC-A1-A1.2.1-C-2015,

inferred and scripted definitions, defined by our own Intermediate Translation Definition Language (ITDL in short) script, will be used by the last layer of our pipeline model to generate target platform code.

II. Related Work

A. *Current state-of-the-art on image semantic segmentation*

One of the first areas that our work is related to consists in the current state-of-the-art in image semantic segmentation. For our model pipeline to accomplish the desired results is necessary for the machine learning layer to have the ability of correctly identifying the visible user interface controls of the observed legacy application and their locations. Finally our machine learning model will be able to infer the overall scene layout of each user interface screens.

State-of-the-art and beyond research topics such as image classification [8] and image semantic segmentation [9] provide new opportunities for various machine learning related tasks ranging from recognition of a cat in an image, its actual position in the scene and the definition of the whole scene attributes such as all objects with dimensions and positions.

B. *Advances in machine-learning inference performance*

Recent advances in computer processing, storage and communication power – and particularly in the area of GPU based numeric/scientific massive parallel processing [10] brought a tremendous improvement to existing or ongoing approaches such as deep convolutional neural models. Currently GPU technologies such as NVIDIA Pascal architecture offer at acceptable price levels for high-performance computing power able to deliver over 10 TFLOPS of parallel computing on thousands of numeric computational cores. Prestigious image recognitions competitions such as ImageNet [11] competition demonstrate am amazing rate of improving of over last years both related to the usage of Deep Learning and particularly deep convolutional neural networks. The employment of GPU based massive parallel computing training of the machine learning models has been a winning approach in image recognition competitions - from winner of 2010 competition that obtained a accuracy of 72% up to the winner of 2015 competition with a accuracy of over 96%.

III. Architedture Model

The generic architecture of our proposed pipeline presented in Figure (1) is based on a three main layers: (a) ***Layer A*** - a basic data preprocessing layer consisting of a image/video stream capturing engine that will be responsible of "observing" the legacy system in order to feed input information to the following layer (b) ***Layer B*** - the machine learning predictive layer responsible of analyzing the input stream frame-by-frame and reconstructing via ITDL the full user-interface scene layout and inferring models and controls structure; (c) ***Layer C*** - based on machine learning layer ITDL output the final layer will generate source code in a language of choice (dictionary based rules) for the inferred user interface views with their own models and controllers.

Target source code OUTPUT of Layer C
Code generation layer (*Layer C*)
ITDL script Output (Intermediate output) of Layer B
ITDL composition layer (*Layer B.2*)
Machine Learning layer with Softmax output layer (*Layer B.1*)
Image frame (Intermediate output) output of Layer A
Pre-processing layer (*Layer A)*
Input video stream (INPUT)

Figure 1 Model Architecture Layers

A. *From video stream to ITDL to source code*

The main proposed innovation of our paper in the area of automated programming is the introduction of a machine learning pipeline that will be able to accomplish real time user interface analysis and reconstruction for the case of legacy system porting to Cloud computing environments.

The proposed machine learning layer within our pipeline model architecture is responsible for the user interface scene inference as previously mentioned. The predictive machine learning model must analyze a stream of images delivered (as a movie/stream or application real runtime environment) from an actual interaction between a user and the legacy system. The actual processing consist in reading a stream of high-definition screens at over $3 * 10^6$ predictor features per screen and at least 10 frames per second, potentially for several hours. Finally the machine learning model is able to infer the screens, generating in real time outputs based on ITDL script language. Individual objects within the analyzed video stream are classified based on Softmax classification equation (1) where θ^T is the transpose vector of learned feature weights for a particular type of application user interface control class (out of a total o *N* trained classes).

$$h_{softmax}\left(x_i^{(j)} \middle| \theta, j \in M, i \in N\right) = \frac{e^{\theta^T x_i^{(j)}}}{\sum_k^N e^{\theta^T x_k^{(j)}}} \qquad (1)$$

Due to the use of the $h_{softmax}$ hypothesis function within the machine learning layer of the pipeline model we will be able to determine the actual “interface control probability” that a certain identified user interface control belongs to a certain class of controls such as button, label, and other user interface controls.

For the individual object position inference and the overall scene decomposition-assembly objective a straight-forward shallow sliding-windows mechanism will be used. In terms of complexity the pipeline model will use a variable sliding window range based on the provided pipeline training data. Nevertheless, the adaptation of the pipeline model to an online learning environment will provide a continuous stream of training cases (such as new controls shapes and behaviors) that will allow the pipeline model to self-adapt.

The actual screen definition of the user interface screens including model and controller information is based on a JSON/dictionary-style ITDL script as shown in *Output 1*.

Output 1 *ITDL Output (generic)*

```
Result = {
        <string element name> : {
              “TYPE”: { [<value> :<percentage>,] }
              “LABEL”: <value>,
               “COORD”: { <values> }
               “SIZE”: { <values> }
               “ACTION”: {<value> }
              [ <string sub-element name> {
                     [<definition>] },
                     .....]
}
```

B. *Translation model algorithm*

The generic process that describes the overall workflow of our pipeline model is presented in the following algorithm:

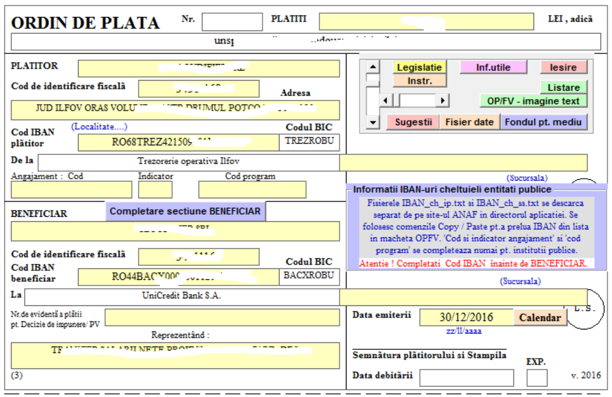
Algorithm 1 *Pipeline Model Overall Algorithm*

```
S0: Training phase
    S0.1: If pre-trained Load pre-trained B.2
          Else
    S0.2: Preprocess training dataset with Layer A
    S0.3: Train Layer B.1 machine learning model
S1: Layer A connects to the legacy app video stream
S2: For each video frame observed by Layer A
    S2.1: Preprocess data within Layer A
    S2.2: Layer B.1 receives data and predicts scene
    S2.3: Layer B.2 received predictions and accumulates
    S2.4: If Layer B.2 received new scene from prev
          S2.4.1: Generate previous scene ITDL
          S2.4.2: Send scene ITDL to Layer C
          S2.4.3: Layer C generates scene source code
S3: Online training phase
    S3.1: If online labeling available
          S3.1.1: Train-update model B.1
          S3.1.2: Add sample to training repository
          S3.1.3: Save pre-trained machine learning
          model B.1
End
Return
```

C. *Use case*

A presentation of a real-life legacy application use case and the actual ITDL output of the model will better clarify the actual process described in *Algorithm 1*. For this purpose we will use a real life windows desktop application presented below in *Figure 2*. For obvious confidentiality reasons in *Figure 2* the details/data within some of the fields has been obfuscated.

Our proposed test use case is based on short “analysis” time frame that our pipeline model has conducted on the legacy application interface screens. The first stage of our pipeline model assumes our machine learning layer is allready pre-trained on a training dataset specialized in application user interface controls. *Layer A* of our pipeline model will capture the application screen as described by *Figure 2* and will apply image segment extraction based on various sliding-windows (step *S2.1* of our pipeline model algorithm). For each extracted window the Softmax classifier will infer the top options for that particular segment in terms of user interface control class. Given a higher-than-average probability inference for a certain sliding-window we can then assign various properties such as dimension and location followed by the inference of the control content (such as user interface button caption or option grid texts)


ORDIN DE PLATA Nr. PLATITI [redacted] LEI , adică

unsp [redacted]

PLATITOR [redacted]

Cod de identificare fiscală [redacted] Adresa

JUD ILFOV ORAS VOLU [redacted] DRUMUL POTCO [redacted]

(Localitate....)

Cod IBAN plătitor RO68TREZ421509 [redacted] Codul BIC TREZROBU

De la Trezorerie operativa Ilfov

(Sucursala)

Angajament : Cod Indicator Cod program

BENEFICIAR

[redacted]

Cod de identificare fiscală [redacted]

Cod IBAN beneficiar RO44BACX000 [redacted] Codul BIC BACXROBU

La UniCredit Bank S.A.

(Sucursala)

Nr.de evidentă a plătii pt. Decizie de impunere/ PV

Reprezentând :

TF [redacted]

(3)

Informatii IBAN-uri cheltuieli entitati publice

Fisierele IBAN_ch_ip.txt si IBAN_ch_ss.txt se descarca separat de pe site-ul ANAF in directorul aplicatiei. Se folosesc comenzile Copy / Paste pt.a prelua IBAN din lista in macheta OPFV. 'Cod si indicator angajament' si 'cod program' se completeaza numai pt. institutii publice.

Atentie ! Completati Cod IBAN înainte de BENEFICIAR.

Data emiterii 30/12/2016 zz/ll/aaaa

L.S.

Semnătura plătitorului si Stampila

EXP.

Data debitării

v. 2016

Figure 2 Desktop legacy application

In the following script output (*Output 2*) we present the inferred scene properties for the above use case:

Output 2 *Use Case ITDL Output*

```
Result = {
"lblORDIN" : {
                "TYPE": {
                                "CAPTION" : 0.97
                                },
                "LABEL": "ORDIN DE PLATA",
                "COORD": { X: 5, Y: 5},
                "SIZE": {X: 150, Y: 20},
                "ACTION": "None",
                "BOUNDDATA": "None",
                "CHILDREN": "None"
                                },
"lblNr" : {
                "TYPE": {
                        "AUTOEDIT" : 0.55,
                        "EDIT": 0.41,
                },
                "LABEL": "Nr.",
                "COORD": { X: 165, Y: 4},
                "SIZE": {X: 40, Y: 20},
                "ACTION": "None",
                "BOUNDDATA": "lblNr_Data",
                "CHILDREN": "None"
                                },
"lblPLATITI" : {
                "TYPE": {
                        "EDIT": 0.89
                        },
                "LABEL": "PLATITI:",
                "COORD": { X: 215, Y: 4},
                "SIZE": {X: 180, Y: 20},
                "ACTION": "on_lblPLATITI_Change",
                "BOUNDDATA": "None",
                "CHILDREN": "None"
                                },
"lblLEIadica" : {
                "TYPE": {
                         "LABEL":0.91
                        },
                "LABEL": "LEI, adica",
                "COORD": { X: 395, Y: 5},
                "SIZE": {X: 35, Y: 20},
                "ACTION": "None",
                "BOUNDDATA": "None",
                "CHILDREN": "None"
                                },
"gidUnnameGridArea":
{
                "TYPE": {
                        "VIEWGRID",:0.99
                        },
                "LABEL": "None",
                "COORD": { X: 5, Y: 35},
                "SIZE": {X: 450, Y: 350},
                "ACTION": "None",
                "BOUNDDATA": "None",
                "CHILDREN": {
        "lblPLATITOR" : {
                "TYPE": {
                        "COMBOEDIT" : 0.59,
                        "AUTOEDIT": 0.30,
                        "EDIT": 0.10
                        },
                "LABEL": "PLATITOR:",
                "COORD": { X: 3, Y: 3},
                    “SIZE”: {X: 185, Y: 20},
                    “ACTION”: “on_lblPLATITOR_Change“
                    “BOUNDDATA”: “lblPLATITOR_Data”,
                                        },

. . .
}
```

As presented in *Output 1* our pipeline model generates a full scene inference for the desktop based application with the screen captured in Figure 2. Based on the actual user interaction with the legacy system the pipeline generates a maximum likelihood probability for each user interface control class. The inferred user interface control class is presented together with with other highly likely potential candidates if such exists.

IV. Implementation considerations

As previously mentioned the machine learning layer of our pipeline model will fully rely on a shallow model architecture. The main reason behind the employment of a shallow machine learning model instead of a classic deep convolutional neural network for image recognition is based on the sparsity of training data for each individual user interface element at the moment of system experimentation. In order to train a deep convolutional network for full scene inference as described by [9] [8] [12] we would need either a pre-trained model or a massive training dataset. Due to the nature of our target scenes that we have to infer we will not be able to use pre-trained models on pictures/photographs such as well-known convolutional models such as AlexNet [8] or Inception [13]. Currently well known methods are available for dataset enrichment such as sample generation and labeling based on SVMs [14] or other simple regression techniques. However in our case the individual user interface control class sample observations are just too few to apply any kind of regression or noise-based dataset enrichment.

As a result, a small variable dimension training set can be actually provided as described by the sample data in *Table* 1 and *Equation* 2, where $X^{(m)}$ is the feature vector for the mth training sample from a given training dataset of M samples and a maximal sample dimension of N_{max} ($X \in R^{MxN_{max}}$), *cls* is the actual control class label (for example “B” for interface button and “L” for visual label). Note that each training sample $X^{(j)}$ belongs to a certain class $C^{(cls)}$ and each class has a particular dimensional space.

Nr	**Nr F**	**F 1**	**F 2**	**…**	**…**	**…**	**…**	**…**	**…**	**cls**
…	…	…	…	.	…	.	…	…	.	…
K	N_K	$x_1^{(K)}$	$x_2^{(K)}$	…	$x_{N_K}^{(K)}$	…	*na*	*na*	.	B
…	…	…	…	.	…	.	…	…	.	…
J	N_J	$x_1^{(J)}$	$x_2^{(J)}$	…	$x_{N_K}^{(J)}$	.	$x_{N_J}^{(J)}$	*na*	.	L
…	…	…	…	.	…	.	…	…	.	…

Table 1 - Sample training data

$$X^{(j)} = \left\{ x_i^{(j)} \middle| \, i \in [1..N_m] \, , j \in [1..M] \right\} \qquad (2)$$

Basically, the pipeline model training dataset will contain a multitude of sub-datasets, each sub-dataset with its own dimension space. The preprocessing phase of *Layer A* within our pipeline model is responsible of scaling the samples to the actual sliding window dimensional space.

Due to the potential high parallelizable nature of our layers within the pipeline architecture – with emphasis particularly on *Layer B.1*, *Layer B.2* and *Layer C* – we employ a map-reduce approach for each of the mentioned layers. At the level of the machine learning layer model we are able to generate inferences in parallel for multiple observed sliding-windows within the video streams. As a result, the proposed pipeline model is gracefully scaling the entire computations required by machine learning model on available resources. Using either OpenCL [15] or CUDA [10] we deploy kernels on currently affordable GPU infrastructure that will compute inferences at the level of sliding-window for each and all known and labeled controls.

Using the proposed feedback-loop described by *S3.1.1*, *S3.1.2* and *S3.1.3* or *Algorithm 1* we can feed correct labels back to the already trained model in order to online-update the model parameters and thus improve scene inference accuracy. Finally, based on a Cloud Computing environment the pipeline model should be able to continuously update the trained weights based on model averaging from sequential or parallel scene inference jobs and create a powerful pre-trained machine learning model.

V. Conclusions

Our proposed pipeline model architecture is able to accomplish an important automated process with high value for a multitude of entities and in particular for small and medium enterprises. The proposed impact for the small medium

ID Proiect: P_38_543, Nr. Ctr. 98/09.09.2016, MySMIS: 104349, Apel - POC-A1-A1.2.1-C-2015,

enterprises community, that is the driving engine of global economy, is high to its cost-saving nature. Basically, our proposed automated process, employed by the pipeline model, saves more than 50% of the translation and migration costs related to transforming a legacy application to a Cloud based one.

Working in a real time and on-line environment the pipeline model is able to serve multiple clients based on the highly scalable cloud computing processing architecture. Employment of online machine learning training and inference highly leverages our pipeline model with the power of auto-adaptivity. Nevertheless, our pipeline model ongoing research process will continually transform the proposed architecture to adopt new technologies, languages and cloud based execution environments.

References

[1] S. Viswanathan and P. Samuel, "Automatic code generation using unified modeling language activity and sequence models," *IET Software*, vol. 10, no. 6, 2016.

[2] R. Campos-Rebelo and F. Pereira, "From IOPT Petri nets to C: An automatic code generator tool," in *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, 2011.

[3] S. Diswal, P. W. V. Tran-Jørgensen and P. G. Larsen, "Automated Generation of C# and .NET Code Contracts from VDM-SL Models," in *14th Overture Workshop: Towards Analytic Tool Chains : Technical report ECE - TR - 28*, 2016.

[4] B. Milosavljević, M. Vidaković and Z. Konjović, "Automatic code generation for database-oriented web applications," in *PPPJ '02/IRE '02 Proceedings of the inaugural conference on the Principles and Practice of programming*, 2002.

[5] M. Trudel, M. Oriol, C. A. Furia and M. Nordio, "Automated Translation of Java Source Code to Eiffel," in *49th International Conference, TOOLS 2011, Zurich, Switzerland, June 28-30, 2011. Proceedings*, 2011.

[6] G. Paulsen, J. Feinberg and X. Cai, "Matlab2cpp: A Matlab-to-C++ code translator," in *System of Systems Engineering Conference (SoSE), 2016 11th*, 2016.

[7] S. Chadha, A. Byalik and E. Tilevich, "Facilitating the development of cross-platform software via automated code synthesis from web-based programming resources," *Computer Languages, Systems & Structures*, vol. 48, 2017.

[8] A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet Classification with Deep Convolutional Networks," *Advances in neural information processing systems 1097-1105*, 2012.

[9] J. Long, E. Shelhamer and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *IEEE Pattern Analysis and Machine Intelligence ISSN: 0162-8828*, no. May 2016, 2016.

[10] J. Ghorpade, J. Parande and M. Kulkarni, "GPGPU PROCESSING IN CUDA ARCHITECTURE," *Advanced Computing: An International Journal (ACIJ)*, vol. 3, 2012.

[11] ImageNet, "ImageNet Large Scale Visual Recognition Competition (ILSVRC)," in *www.image-net.org/challenges/LSVRC/*.

[12] A. Karpathy and L. Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions (arXiv:1412.2306)," *Computer Vision and Pattern Recognition*, 2015.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions," *Computer Vision and Pattern Recognition*, 2014.

[14] R. Mao, H. Zhu and L. Zhang, "A New Method to Assist Small Data Set Neural Network Learning," in *Intelligent Systems Design and Applications, 2006. ISDA '06. Sixth International Conference on*, 2006.

[15] K. Wang, J. Nurmi and T. Ahonen, "Accelerating Computation on an Android Phone with OpenCL Parallelism and Optimizing Workload Distribution between a Phone and a Cloud Service," in *Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), 2016 Intl IEEE Conferences*, 2016.

cloudifier .NET
ID Proiect: P_38_543, Nr. Ctr. 98/09.09.2016, MySMIS: 104349, Apel - POC-A1-A1.2.1-C-2015,