Effective Execution of Systems Dynamics Models

Alexey Mulyukin, Tatyana Kossovich
ITMO University
Saint-Petersburg, Russia
alexprey@ya.ru, tatyana_kossovic@mail.ru

Ivan Perl
ITMO University
Saint-Petersburg, Russia
ivan.perl@corp.ifmo.ru

***Abstract*—This article describes a new project – sdCloud, which was designed in purpose to create an effective cloud-based execution environment for System Dynamics models. The article contains overview of basic terms and conditions in System Dynamics, description of sdCloud project and investigations of issues, which has occurred in development process, including integration of existing System Dynamics tools, possibilities of optimization processes in System Dynamics modeling and its basic approaches.**

I. Introduction

Personal, portable, and mobile computers have become very powerful, but even so don't always have enough computational capacity for all simulation applications. Especially for Big Data applications, researchers often need faster execution and high bandwidth connections between the execution engine and data, regardless of where the data resides. It is exactly this kind of need that has given rise to cloud computing and cloud storage paradigms.

Cloud-based execution is particularly convenient for Big Data applications and research, as keeping both the data and the execution engine in the cloud reduces the modeler's involvement in data sharing and resource management. Cloud-based engines are also an archetypical match for System Dynamic models, allowing researchers to select the execution engine somewhat independently of the model creation tool. Especially in certain research domains, matching the engine technology to the data processing context has significant benefits. Furthermore, cloud-based executor is more accessible to typical users than engines which require local installation.

II. System Dynamics basics

System Dynamics is a methodology and mathematical modeling technique to frame, understand, and discuss complex issues and problems. Originally developed in the 1950s to help corporate managers improve their understanding of industrial processes, System Dynamics is currently being used throughout the public and private sector for policy analysis and design. Through this approach, there is a possibility to create simulations based on shrinked space and time to the target level allowed to carry out behavior investigations. Such investigations are useful in purpose to monitor any of the following aspects like causal relationship, feedback loops, delay of reactions, as well as the impact of the external environment. System Dynamics is mainly used for the development of long-term or strategic models.

The simplicity of the structures used in System Dynamics to represent the modeled systems, as well as a significant amount of information being processed leads to the fact that the calculation of the data amenable to automation systems. A feature of the long-term model is that large segments of time are used to model systems using a small step simulation. Thus, this leads to the generation of enormous amounts of data at the output of the modeled system. In addition to modeling just requires significant computing power to support complex long-term calculation model [1].

The system dynamic models are based on the stocks and flows. Stock, in the concept of System Dynamic - it is an element of the model, which describes the accumulated value in the model. Flow - it's rate of stocks changing. Stocks can have relationships between each other stocks via flows. Value of the flow is defined by his equation that can be a constant a function (function that does not have depend on any stocks of the model), or a function that depends on the external values of model. As this values for the flow equation can be used the value of any other flows, stock or external variable that is declared within the model. The flow can only change the value of stocks, which connected to this flow as sources or destinations. If stock is marked as the source for specific flow, then value of this stock are change in the negative direction, and for destination stock value are change in the positive direction [2], [3].

There are several file formats to determine dynamic system models. Most of them were specifically designed for particular software, like VEnsim and Stella. Small differences between realizations lead to a compatibility issue. And as a result XMILE specification was released in 2013 (XML Modeling Interchange LanguagE) by Steve Adler. The main feature was the point that whole model description is provided using XML. This format became the base one, mostly because its independence from programming point of view.

III. PySD library

Despite the fact that the XMILE specification was appeared recently, there are a lot of solutions, which executes System Dynamics models and supports XMILE specifications. And PySD library is one of these solutions [4].

PySD is open source Python library, which was designed by James Houghton [5]. This library supports several model formats, first of all – models described by XMILE. On picture below there are five main steps, which described its operation principle.

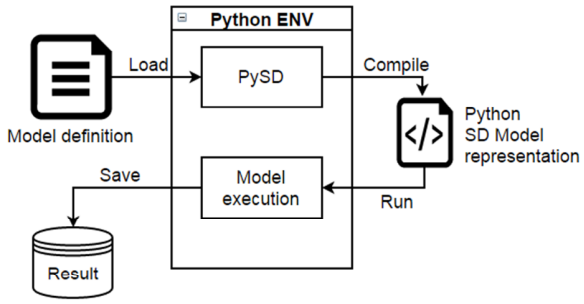

Fig. 1. Diagram of PySD execution steps

1) *Getting model description from file*
2) *Converting model into Python code*
3) *Compilation of Python code*
4) *Model executing*
5) *Saving results of execution into memory*

Model execution results might be converted into any format. This approach provides additional opportunities for analyzing the original model.

PySD library allows to change default conditions set by original model description. This functionality provides possibility to reduce time in case to rerun model with alternative settings because there is no need to recompile model file.

IV. SDCLOUD PROJECT

SdCloud solution is a new open source project dedicated to creating a cloud-based execution environment for System Dynamics models. Its goal is to provide model sharing and remote model execution and result generation possibilities to modelers and researches working in area of System Dynamics.

A. Rationale

The technological layer of System Dynamics modelling appears feature rich and mature. Commercial products (like those from ISEE, Ventana, AnyLogic, Simulistics, and others) provide rich modelling functionality covering most of the common use cases. Forio-Simulate features a cloud-based model execution and presentation environment for selected proprietary modelling languages.

There are also open source and free projects like Minsky, Mapsim, and Simantics SysDyn. Paralleling the migration of applications to web-based deployments in other domains, web-based modelling approaches have emerged, offering a kind of fusion modelling approach. InsightMaker and Systo, for example, allow model development in almost any Web browser supporting JavaScript.

Despite this highly evolved modelling ecosystem, there are some limitations. In particular, until the formalization of XMILE, it was difficult to separate model creation and model execution. This separation is particularly important in Big Data work, because the data itself is often coupled to a particular storage and processing technology. Thus the tool which might be most effective for developing a particular model is often not the most effective tool for executing the model on the relevant data store.

In discussions stimulated by System Dynamics society and others interested in model exchange, it became clear that a cloud-based deployment would be the most effective way to address the Big Data needs, and that cloud-based model translation, execution, and sharing would not only create synergies with existing System Dynamics tools, but also be a hospitable and open house for future development and research.

B. Goals

As with other cloud-based solutions, an important benefit of sdCloud is that the cloud service not only executes the model, but also manages all resource-related issues and is easily accessible from almost everywhere via almost any connected device.

While the key goal of this project is to build a cloud-based system, running on powerful, well-connected servers, other requirements, though, seemed necessary to reap the full benefit from a cloud modelling service. Thus, we expect sdCloud to provide following set of features to the end-user:

- The ability to upload a System Dynamics model in any format, including XMILE.
- The ability to download a previously uploaded model in any format, independently from the format in which it was uploaded.
- The ability to schedule model execution, track schedule and execution progress, and browse model execution results.
- The ability to share models and model execution results both within sdCloud and outside it.

The sdCloud project does not aim to build a complete solution from scratch. Instead, we hope to build a management and access layer on top of components created by others.

C. Architecture

SdCloud project includes 4 basic components:

1) *Web UI:* Web UI allows end-users accessing the system, upload models, schedule execution and monitor execution results.
2) *REST API:* Rest Api should provide required functionality for Web UI and also work as an access point to the system for 3d-products.
3) *Internal core libraries:* modules, which represents key features of SdCloud solution.
4) *Model execution components: modules, which provides model execution functionality.*

SdCloud solution architecture is schematically represented on diagram below.

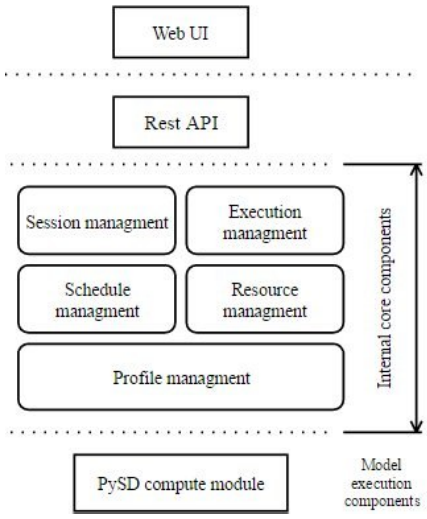

Fig. 2. SdCloud solution architecture

D. Model execution components

As a default module for executing System Dynamics models we decided to use PySD library due to the following list of its advantages:

- The conversion of the original System Dynamics model description file into executable Python code and its compilation in purpose of performance improvements for further reruns.
- The possibility to execute System Dynamics model in the background.
- The possibility to store and analyze results of model execution.
- PySD is open source project, supported by System Dynamics community.

E. Internal core components

It was decided to use .Net platform for developing core components for sdCloud solutions, due to its advantages in complex solutions development. And here we faced an issue of integration PySD library into solution developed using other programming languages. In our case we faced the issue to integrate PySD library into .Net solution, written by C# language.

V. Approaches to integrate Python libraries into .Net solution

There are two possible ways to solve the issue of integration Python library into .Net solution – process approach and IronPython library.

A. Process approach

This approach is relatively versatile as it does not require additional development libraries. In this case all the commands executed by Python are strings. If you choose this approach, it does not matter exactly how the executable instructions are transferred to the program - it can be a dedicated file containing instructions or constants listed directly in the program.

To get started with Python we have to create a process of «Python.exe» and specify the appropriate input parameters as command line arguments. However, this is a separate process, from which you can obtain the output data from the launched Python modules, which in turn can be processed by .Net [7].

This approach has its advantages and disadvantages. Among the advantages are the following points.

1) *Independence from the version of Python:* Using processes abstracts the installed version of Python in terms of program development for the .Net platform for the launch «Python.exe» process only requires the use of the correct path to the executable. This makes it possible not to change the code in .Net if you change Python version.
2) *No need for additional .Net libraries:* For the development of this approach does not require the presence of additional .Net libraries, since all the necessary components for the processes inherent in .Net platform core since version 3.5.
3) *Run «pure» Python scripts:* This approach as a whole is the work with Python command line in the hidden from the external user form. Thus all the work on encapsulation run Python console takes responsibility main program, and the executable commands have the same syntax as if it were run manually. This feature can facilitate verification of correctness Python scripts in case of errors.

On other hand, the usage of this approach has several disadvantages.

1) *Python configuration issues:* The main program cannot control the errors associated with incorrect installation and Python settings on the host.
2) *Lack of typing for input and output data:* When using the process all the work with Python modules occurs through the transfer of string instruction and receiving the string data from the output stream, which also complicates the control of the progress of the program. The main program should include mechanisms for output treatment in the strongly typed data for further work.

B. IronPython

IronPython is an implementation of the Python language itself, written entirely in C # and designed for the .Net platform. To work with modules written in Python, the code being developed in C #, you need to connect a set of relevant libraries [8]. In doing so, run Python modules can be both a part of the project as a script and compiled together with the whole project, so be it external files.

ScriptEngine and ScriptScope: Two main object provided for interaction with the Python. The first object is responsible for the execution of Python scripts and importing additional Python libraries. The second, in turn, is a scope of interaction and stores all the variables, allowing them being changed easily [9].

For this approach, you can also select a number of advantages and disadvantages. Benefits include the following points.

1) *"On the fly" Python executable module:* The executable module is generated during the execution of the main program in C # via ScriptEngine object capabilities. In case of errors in Python script, an exception to the detailed description of a particular error. This feature simplifies script debugging during development process.
2) *Controlled work with Python modules arguments:* All the parameters for run-ins are stored in the object ScriptScope, their values can be changed during the execution of the main program, whilst settings - added or removed.
3) *Import of additional Python libraries:* This feature allows you to load additional Python libraries to run complex scripts, and enables ScriptEngine object.
4) *Python interaction with .Net:* Iron Python supports usage of .Net assemblies.

Among the disadvantages of using Iron Python are the following:

1) *Different versions of Iron Python and different versions of .Net platform:* As in Python, there are 2 supported at the moment version of Iron Python language - 2.0 and 3.0 respectively. It is not possible to simultaneously support startup script, written in different versions of Python.
2) *Limited functionality and possible errors in Iron Python libraries for .Net.*

C. Common approach to integrate PySD library

Two approaches described above have their advantages and disadvantages. But both are oriented to launch Python scripts directly from .Net runtime. So, it makes code support more complicated. The following approach allows to separate runtimes in order to simplify development.

This approach may be applied not only in .Net solutions and it is also very simple in usage. It doesn't depend on Python support in .Net, and main feature is that Python and .Net runtimes are absolutely separated. This might be done by creating simple REST API service, which launch PySD library. This service is taken place as a wrapper, and it hides all direct actions with PySD library. The Fig. 3 below shows its architecture.

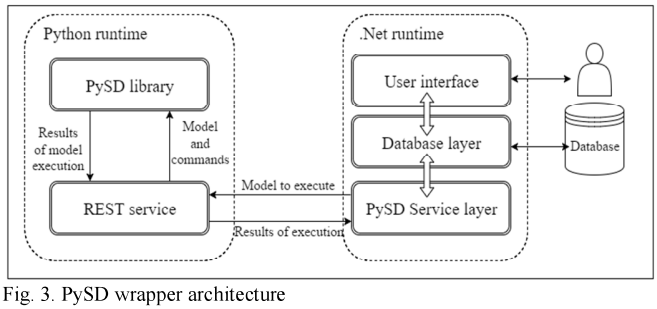

Fig. 3. PySD wrapper architecture

This architecture has several advantages.

1) *Environment independence:* Python and .Net executes in separated runtimes. No direct calls to PySD library from .Net runtime.
2) *Extensibility:* REST service for calling PySD functions may be easily modified due to any changes in PySD. At the same time, its API specification may not be changed, so you don't need to change anything in main solution source code.
3) *Efficiency:* This architecture covers no extra resource consumption for running Python directly from .Net environment. So, it makes this approach the fastest one.
4) *Less limits:* There are no functional limits of PySD usage and Python functions as well.

Due to its advantages this approach is the best of all described above.

VI. Optimization Processes in System Dynamics

After first we had got the first working prototype of SdCloud project, it was decided to discover possible ways of its further optimization. In the first iteration of our investigate we take a CPU time resource as primary criteria for optimization process. To start with it is needed to determine existing optimization approaches in System Dynamics.

A. Possibilities of optimizing the process of modelling in System Dynamics

Existing algorithms of System Dynamics models are sequential. Because of what computing resources are not optimally used, and as a consequence the complete simulation time considerably higher than when using the parallel approach.

There are basically two fundamentally different approaches to optimize the modeling process. The first of them - is an instrumental approach of optimization by changing the tools and technologies used in the development of software solutions for System Dynamics models. The second approach - the development of specific distributed algorithm that allows efficient use of computational resources.

B. Instrumental approach

The first optimization approach is instrumental one. On the basis of the above, the structure of System Dynamics models may be considered. We have to pay attention to the definition of a function, which defines a flow. That is to say, that the flow function has no side effects for the entire system, which corresponds to the pure function definition from programmatically point of view.

Pure functions are one of the main concepts in functional programming. Consequently, the concept of formation System Dynamics models may be implemented using functional programming languages. The key advantage of such languages to procedural or object-oriented programming languages is

that it become possible to optimize the program working process at runtime due to different aspects, such as clean functions. For example, pure functions may be performed independently from each other, because of they don't carry side effects and cannot effect each other performance. Functions, which require complex calculations, may be turned into caching, thus would lead to the usage of value from the runtime environment cache instead of full value of allocation, while recalling the calculation. So, it is apparent that the correct description of the structure transformation and System Dynamics models into executable code using functional programming language, the model execution may be optimized by the runtime, without the development of specialized distributed algorithm.

C. *Distrubuted algorithms approach*

Let's take a look at the structure of dynamic systems models as well as models describing XMILE format [4]. You may notice that every equation used to calculate flow values, is a pure function, which has no side effects to the whole system. Such functions are one of the main concepts of functional programming. Algorithms have been developed for functional programming concepts can be optimized at runtime using approaches such as parallel execution of pure functions, caching computing for complex functions and operations. Thus, with proper transformation of describing structure we can achieve optimizations of process without the development of specialized distributed algorithm, just by execution environment.

Distributed algorithm for calculation of System Dynamics models should provide the ability of parallel computing of model components. This will require a more detailed look at the structure and identify two main types of links between components: the components without feedback and components with feedback. Feedback - a property that defines the dependency between components. If the two component models (A and B) are connected by flow F, where A is the source of F stream, and B - drain, and thus the value of the flow F depends only on the value of the source, then it can be assumed that the model components A and B have no feedback. If the value of the flow depends on the drain, then - components have feedback.

First of all, you should pay attention to the model, in which the components have no feedback. Based on the definitions above, we can see that the value of the flow F, as the source of A is independent of the value that is in drain in B. If you add extra sources in this system (Ai), which will be connected to the same drain (B) with flows without feedback (Fi), then it can be said that the input streams (Fi) are not dependent on each other. Consequently, the values of these streams, as well as the values of drains can be calculated independently from each other in parallel. The value of the resulting drain (B) can be calculated as the sum of pre-calculated independent streams (Fi). The Fig. 4 below shows the model without feedback scheme.

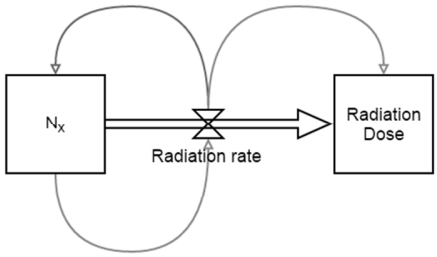

Fig. 4. Model without feedback scheme

$$Radiation\ rate_x = \frac{N_x}{2 * R^2 * T_{1/2}} \rightarrow F(N_x)$$

Where $T_{\frac{1}{2}}$ – half-life period, and R – distance between radioactive object and target object and it's a constant value.

$$Radiation\ dose' = Radiation\ dose * \sum_{x=0}^{n} Radiation\ rate_x \rightarrow F(Radiation\ dose, F(N_x))$$

Furthermore, it should pay special attention to the model that contains the components with feedback. The problem of distributed algorithm development for such models is that the simulated system has a high degree of connectivity. Suppose that there are several sources (Ai), which are connected flow (Fi) with a drain (B). The value of each selected stream depends on the drain current value (B). Thus, the calculation of the flow, as well as the associated source for the next time interval cannot be performed until a value for the drain at current time period is calculated. If we consider that the resulting value of the drain is the sum of all incoming flows, it can be said that all input streams (Fi), as well as the origins of (Ai), are dependent on other streams. Thus, an independent flow values calculation cannot be performed. In a parallel calculation of such flows occur competing access to the drain value (B). The Fig.5 shows a diagram of drains connections for model with feedback.

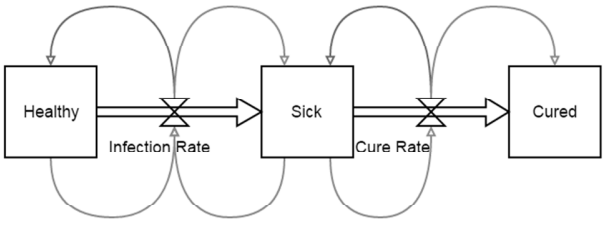

Fig. 5. Diagram of drains connections for model with feedback

$$Infection\ Rate = Sick * \frac{Cured}{Total * Sick\ frequence} \rightarrow F(Sick, Cured)$$

$$Cure\ rate = \frac{Sick}{Curing\ duration} \rightarrow F(Sick)$$

Where $Sick\ frequence$ and $Curing\ duration$, it's a constant value.

$$Sick' = Sick + Infection\ rate \rightarrow F(Sick, Infection\ Rate) \rightarrow F(Sick, F(Sick, Cured))$$

$$Cured' = Cured + Cure\ rate \rightarrow F(Cured, Cure\ rate) \rightarrow F(Cured, F(Sick))$$

In the case when computation engine use parallel methods for calculate values in flows there is a concurrency to accessing for values of drain B. In order for the resulting data of the simulated system still are correct, it's required to avoid concurrency values access. When concurrency are arrive to computation engine of System Dynamics models and it's not handled, then may be a situation in which two flows would reserve the same value of the drain. This will not correct calculated values of flows equation at this period. The consequence of this is the waterfall effect for all of the resulting values of the simulated system. Thus, we come to what is the correct calculation of the parallel methods with feedback flows is not fully possible.

In a real system it's very rare to find when the model is the components that are connected to each other only by streams without feedback. Naturally, there are components in the models as with feedback, and without it. Thus, it is the task of analyzing the components within an existing communications models. You can imagine the whole system as a unidirectional graph, where edges indicate the direction of streams and thus the connection between the components for this task. This point, which depends on the maximum number of components is strongly coupled system, removing that can obtain two independent systems are called sections or cut point. To find this point, you can use the algorithm for finding the minimum cut graph Stoer-Wagner algorithm. By applying this algorithm to the model's graph may be obtained independent parts of the system, which can be calculated in parallel.

D. Problems of disturbed algorithms development

As previously mentioned, the System Dynamics model is performed on large segments of the duration of time with a small step towards the whole area of the simulated time. Existing modeling algorithms use a sequential approach with the simulation time intervals with a fixed step. This approach does not allow to process model independently. But this approach can be adjusted that will allow without any interference with the basic algorithm to model the system at different time intervals in parallel. Assume that the System Dynamics model to be processed in the time interval t1 - t2 with step dt. Then, increasing the simulation step up to 4dt, you can get a boundary point of the simulated system on the whole required length of time. For these boundary points will be obtained data on the values of each drain and the stream used in the model. Using them we can model the system on each time interval obtained from the already predetermined step. Because the initial condition is known for each period of time, then we can assume that each of these periods of time are not dependent on each other, hence each received time period can be simulated in parallel.

VII. ERLSD COMPUTATION CORE

In a way to optimize our solution we decided to investigate disturbed algorithms approach in relation to the sdCloud project. So, we started to develop our own model execution component, written in Erlang. While Python has the advantage of being familiar to researchers in many areas, allowing them to directly modify the generated code if necessary, Erlang promises efficiency benefits for computationally intensive simulations.

Erlang execution environment provides the next key-features:

- Not require for manual handling of system resource allocation and management (allocate new threads, processes), this operation takes a lot of time.
- All Erlang processes executes in virtual machine, that provides a simple way for manage using resources.
- Erlang has integrated functionality for design and developing distributed applications.
- Hot code upgrade provide possibility for quick changing the model compiled representation without any downtime on environment.

The following diagram shows the difference in average execution time depending on same number of operations.

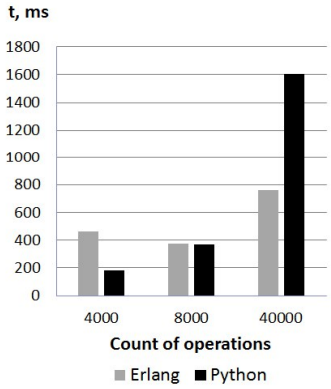

Fig. 6. Time dependency graph

According to diagram from above, it can be stated, that usage of Erlang might increase performance for models, which requires big number of operations.

Also, models expressed in Erlang should be able to easily exploit Erlang's advanced optimization and parallelization features.

This module is still under development now, but we expect that models expressed in Erlang will be more naturally able to exploit the cloud's flexible resources allocation. Therefore, SdCloud solution will use both model execution components – PySD and ErlSD, depending on model specification and complexity.

VIII. CONCLUSION

Summing up, the sdCloud project is an answer to the big data processing and data access challenges facing the System Dynamics community. A high performance cloud-based environment can make working with system dynamics easier, faster and closer to the model's users. At the same time, sdCloud project is in the process of constant work on improvements in order to optimize execution process of System Dynamics models.

REFERENCES

[1] J. Sterman, *Business dynamics: systems thinking and modeling for a complex world*, Cambridge: 2002.

[2] J. W. Forrester, *The Beginning of System Dynamics*, Cambridge: MIT, 1989.

[3] M. J. Radzicki and R. A. Taylor, *Introduction to System Dynamics (version 1.0)*, U.S.: Sustainable Solutions, Inc., 1997.

[4] J. Houghton, M. Siegel, "Advanced data analytics for System Dynamics models using PySD", *33rd International Conference of the System Dynamics Society,* vol. 2, 2015, pp. 1436-1463.

[5] GitHub, Source code of PySD library, Web: https://github.com/JamesPHoughton/PySD/

[6] Oasis official site, specification document: "XMILE: An XML Interchange Language for System Dynamics", Web: http://docs.oasis-open.org/XMILE/XMILE/v1.0/errata01/XMILE-v1.0-errata01-complete.pdf

[7] Kennedy M. "Python for C# Developers", *CODE Magazine. Add to your Toolbox – Learn Ruby and Python*, 2014.

[8] Foord M., "Muirhead C. IronPython in Action", *Manning Publications*, 2009

[9] Pierson H. "Introducing Iron Python", *CODE Magazine. Discover the new .NET languages*, 2008.