Non-visual platform components for a system of polyvariant calculation of dynamic models of the production process

Sergey Medvedev[1], *Vitaly* Terleev[1,2,*], and *Olga* Vasilyeva[3]

[1]Agrophysical Research Institute, Grazhdansky pr., 14, St. Petersburg, 195220, Russia
[2]Peter the Great St.Petersburg Polytechnic University, Polytechnicheskaya, 29, St. Petersburg, 195251, Russia
[3]Moscow State University of Civil Engineering, 26, Yaroslavskoye shosse, Moscow, 129337, Russia

Abstract. Production process models have been used for many years in decision support systems in agriculture. They allow solving the problems of forecasting the yield, operational support, integration with GIS, calculating pedotransfer functions, etc. A high-performance, efficient platform has been developed to create powerful yet lightweight applications for a wide range of tasks. These tasks were solved in the RW.Ring platform, which was developed specifically for the new version of the APEX polyvariant calculation system instead of the outdated kernel of the old system, which contained many non-optimal solutions that impede the effective development of the system. While the new version of the polyvariant calculation system itself is currently under development, the platform itself already contains debugged working modules and can be used for a large number of similar applications.The platform's performance is confirmed by the successful development of the Schicksal statistical analysis program. In the future, the platform will develop in parallel with the new version of the APEX polyvariant analysis system, as well as other programs based on it. The RW.Ring platform can be recommended as a set of standard libraries designed for building any shells for a large number of models.

1 Introduction

Production process models have been used for many years in decision support systems in agriculture [1]. They allow solving the problems of forecasting the yield [2], operational support, integration with GIS [3], calculating pedotransfer functions, etc. However, in its pure form, the benefit from using the production process model in comparison with the vegetation experiments [4] can be felt only when using special media that automate typical scenarios for working with the production process model. Sometimes the polyvariant calculation [5] underlying this automation is supported by the model itself (WOFOST [6], ORYZA), sometimes it is not. In the second case, it is necessary that the general functionality required by the researcher must be removed from the model and placed in a

*Corresponding author: vitaly_terleev@mail.ru

specialized shell. In addition to the polyvariant calculation required for all calculations, the shell can support functions such as crop rotation, statistical analysis of model results, generation of daily weather meteorological data and loading them from external sources [7]. It is obvious that such a software product must integrate a large number of components and therefore needs a reliable developed platform that allows to effectively solve these problems.

Following tasks should be considered in the terms of the polyvariant calculation system solution, and requirements which can be solved at the platform level should be formulated.

Table 1. Requirements for the platform by shell tasks for production process models.

The task solved by the shell	Platform requirements
Search of the causes of failures in multiple model calculations	A logger that allows to log events occurring in different application domains through a single mechanism
Storing complex presets in multiplayer mode	A configurator that allows to save arbitrary data structures in configuration files
A big amount of computation on a background thread	Components for multithreading support
Support for a large number of dynamic models of the production process [8]	An assembly classifier that allows to load plugins both into the current application domain and into other domains
Dynamic loading of complexly connected components at program start	Loader automatically handling system component dependency graph
Ability to work both locally and over the network [9]	The presence of a network module that provides both remote procedure calls and a lower-level mechanism for exchanging streams and datagrams
Ability to use different technologies for developing user interface for different environments	Allocation of the level of services, which allows abstracting from specific technologies of user interface development

These tasks were solved in the RW.Ring platform, which was developed specifically for the new version of the APEX polyvariant calculation system instead of the outdated kernel of the old system, which contained many non-optimal solutions that impede the effective development of the system. While the new version of the polyvariant calculation system itself is currently under development [9], the platform itself already contains debugged working modules and can be used for a large number of similar applications [10-13]. This work is devoted to the description of these modules and their practical application in the problems of modeling the production process of cultivated plants [14-16].

2 Material and methods

The RW.Ring platform, designed to solve the above problems, is a set of .NET Framework libraries. Although .NET Core is a newer and more powerful system, it is currently too raw and contains a number of bugs that are missing in the .NET Framework. In addition, a number of used libraries are available only for the .NET Framework, and this system itself has lower system requirements. In addition, a stripped-down version of the .NET Framework Client Profile is used instead of the full version of the .NET Framework to further minimize system requirements. Even the network module in RW.Ring is made in such a way that it can work by itself without the full version of the .NET Framework.

The RW.Ring platform currently contains three libraries:

- Notung.dll. The main functionality of the platform and all its levels of abstraction are contained here;

- Notung.Windows.dll. This library contains imported Windows functions necessary for the platform to work;
- Notung.Helm.dll. This library contains the implementation of application services for the Windows Forms GUI development technology.

This architecture allows to develop application services implementation for newer GUI development technologies, such as WPF and UWP, as needed. The name Notung and some other terms of the platform are borrowed from the "Ring of the Nibelung" by R. Wagner.

The Notung.dll library contains namespaces that categorize the functionality of the library:

- Notung contains the most general data types that are difficult to categorize and are used everywhere;
- Notung.ComponentModel contains .NET component model extensions that can be used in conjunction with application services or independently;
- Notung.Configuration contains data types that implement the functionality of the configurator;
- Notung.Data contains data types that describe data structures used in the platform and algorithms for working with these data structures;
- Notung.Loader contains data types that implement loader functionality;
- Notung.Logging contains data types that implement the functionality of the logger;
- Notung.Services contains data types that describe application services and their associated abstractions;
- Notung.Threading contains components to support multithreading.

The Notung.Windows.dll library contains the only namespace Notung.Helm.Windows, which contains all data types that are wrappers over standard functions and objects of the Windows operating system that may be required to solve modeling problems.

The Notung.Helm.dll library also contains the following namespaces, which categorize the functionality of the library:

- Notung.Helm contains data types that allow Notung.dll services to interact with Windows Forms technology in general;
- Notung.Helm.Configuration contains components for working with the configurator in the context of Windows Forms;
- Notung.Helm.Controls contains reusable controls that you can put on Windows Forms;
- Notung.Helm.Dialogs contains dialogs and their presenters, through which Notung.dll services for Windows Forms are implemented.

This research discusses the most basic tools included in this platform. When developing them, principles were used that allow, whenever possible, to use them independently, without importing entire libraries. All the capabilities of the library can be divided into components and infrastructure to do this. Components include the first four functions from Table 1. While components have a minimal number of dependencies on each other, the infrastructure, on the contrary, collects them into a single system that increases the productivity of developing production process modeling environments many times over, compared to individual components.

3 Results and discussions

3.1 Logger

The software interface of the RW.Ring logger is made similar to the software interface of the popular log4net logger [17], but it itself contains a number of features that should be taken into account when using it. Below is a list of the main features of the RW.Ring logger that should be taken into account.

- The main logger instance is shared between application domains, which allows centralized logging of information from all plugins.
- If there is reliable information about the main thread of the application (a special interface is responsible for this), logging is performed in a separate thread, which ends after the main thread of the application.
- The logger does not require any additional actions for starting and stores its settings in the standard .NET Framework configuration file. It should be noted that these settings are much less than those of log4net, and much of what can be set in log4net through the configuration file [18] must be done programmatically in RW.Ring.

As with log4net, there is a LogManager class that contains a static GetLogger method that returns an instance of the ILog interface. However, this interface itself contains only one WriteLog method with three parameters:

- message - the message to be written to the log (string);
- level - the "level" of the message, which has a special enumerated type that can take one of the following values:
  - Debug - debug message;
  - Info - informational message;
  - Warning – warning;
  - Error – error;
  - Fatal - fatal error;
- data - additional data associated with the message.

It should be noted that the InfoLevel datatype is used by almost all RW.Ring modules and is therefore declared in the Notung namespace, not Notung.Logging.

All basic actions on the ILog interface, similar to the methods of the same interface in log4net, are implemented in the Notung.dll library as extension methods for this interface, which in the current version of RW.Ring are in the LogManager class.

Also, it should be noted that the LogManager class has a SetMainThreadInfo method that takes an instance of the IMainThreadInfo interface. This API is intended to support the operation of the logger in asynchronous mode. The IMainThreadInfo interface contains two read-only boolean properties: IsAlive and RealiableThreaing. The first of these properties returns a boolean value indicating whether the main thread of the application is currently running, the second returns true if the data source for the main thread is reliable. If the object that implements this interface is a wrapper over a real application thread, this property must return true. If the object that implements this interface is a stub that mimics information about the main thread, this property must return false. In this case, when calling the LogManager.SetMainThreadInfo method, the logger simply does not switch to asynchronous mode. The base implementation of the IMainThreadInfo class is the CurrentMainThreadInfo class, which displays the state of the thread in which this class was instantiated.

Another method of the LogManager class allows to add appenders - instances of the interface that are directly responsible for writing information to log files, to the console, etc.

At the first call to the logger, an appender is automatically created, which writes log files to a special directory in the user profile. To generate the path to this directory, the ApplicationInfo auxiliary class is used, which is used not only by the logger, but also, for example, by the configurator and therefore placed in the Notung namespace. Other appenders can be added to this appender that must implement the ILogAppender interface, the only method of which takes as a parameter an instance of the LoggingData structure, which is an array of read-only instances of the LoggingEvent structure, which has the following fields:

- Message - the text of the message that is written to the log;
- InfoLevel - "level" of the message that is written to the log;
- LoggingDate - date and time when the message was written to the log;
- Source - the name of the logging source passed to the LogManager.GetLogger method;
- Data - additional data passed to the ILog.WriteLog method.

In addition, this structure contains the context in which the logging took place. Since messages are collected into a single logger from all application domains, the context in which each message was originally created must be different for each of these messages. Since the context is a reference data type, there is no need for each message to contain its own copy of the context. Instead, all messages created in the same context contain links to the same context. The context itself is a dictionary that maps parameter names to their values. To work with the context, the LoggingContext class is intended, which contains the Global and Thread properties that return instances of the ILoggingContext interface. The Global property always returns the same instance for all contexts across all application domains, and Thread returns its own context instance for each thread. The ILoggingContext interface contains an indexer whose key is the name of the parameter and the value is the value of that parameter, and a Contains method that checks for the existence of a key with the given name in the context. The principle of the context is as follows: when a parameter is requested from an instance of the LoggingEvent structure, it is first searched for in the context of the stream in which the log entry was created, and if it is not there, then it is searched for in the global context of the application. Since computations in production process models are expedient to be multithreaded for better performance, the global context is thread-safe. The formatting string stored in the configuration file should be used to request parameters from the context.

3.2 Configurator

The standard .NET Framework configurator works well with scalar application-level settings, but it is unstable with user settings, and, moreover, does not allow storing complex data structures in configuration files. Therefore, in the RW.Ring platform, the standard configurator is used only to store scalar application-level settings, in particular, logger settings, which are not supposed to be changed during the operation of programs, and a separate configurator has been made for the settings that can be changed by the end user. The operating scheme of this configurator is shown in Figure 1.

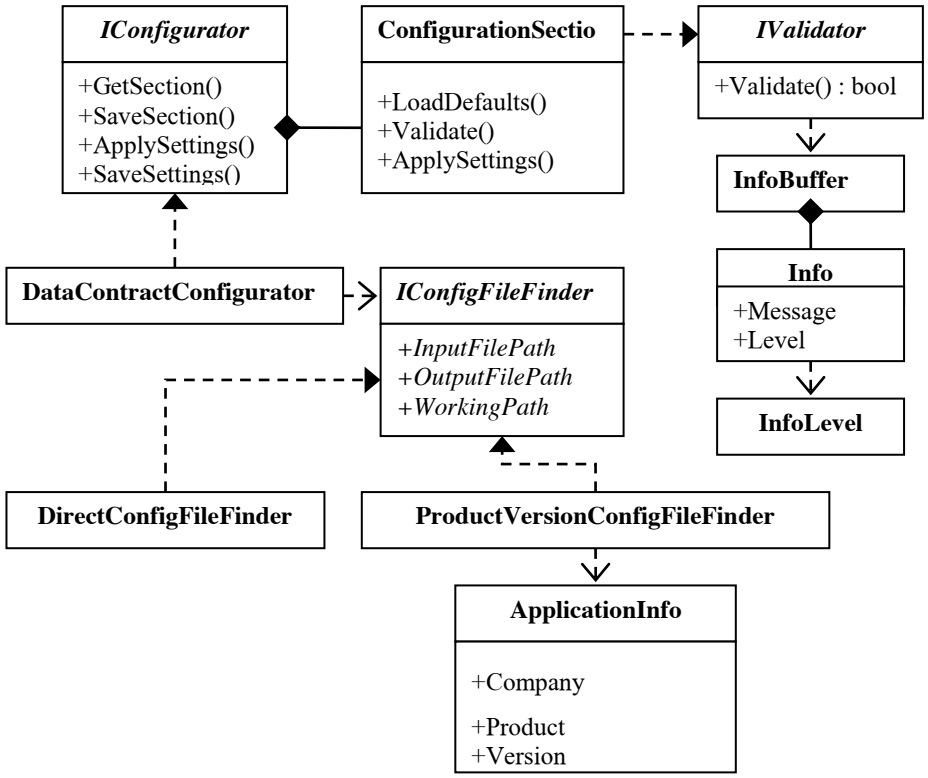

Fig. 1. Configurator operating scheme.

The configurator is an interface that provides access to various configuration sections. The main method of the configurator is the generic GetSection () method, which returns a configuration section of the requested type, read from the configuration file. You can replace a section after editing using the SaveSection method. When the ApplySettings method is called in the configurator, the virtual ApplySettings method will be called for all loaded sections. Calling the SaveSettings method will save all configuration sections to a configuration file.

The basic implementation of this interface is the DataContractConfigurator class, which, depending on whether the configuration section has the DataContract attribute, uses either DataContractSerializer or XmlSerializer to serialize this section to a file. All sections are saved in one configuration file, and their names must be unique during serialization., This class uses the IConfigFileFinder interface to search for a configuration file, which contains three properties: the path to the configuration file from which to read the settings, the path to the configuration file, where the settings should be saved, and the path to the directory for storing data that, for one reason or another, is inappropriate store in a config file. Such data can be either settings of the generator of daily weather meteorological data, or settings of adapters of production process models, and other binary data used by specific plugins.

There are two implementations of the IConfigFileFinder interface. A simple implementation of DirectConfigFileFinder contains a specific full path to the configuration file, and both properties that return the file path to read and save return that path. A more advanced implementation of ProductVersionConfigFileFinder uses the same ApplicationInfo class as the logger to dynamically generate the path to the configuration file based on the manufacturer of the current program, the name of the software product and

its version. If a configuration file exists for the current product version, both properties return the path to that file. Otherwise, the path to the configuration file for reading the settings is the path to the configuration file from the latest version where it exists, and the path to the configuration file to save the settings is the path to the configuration file of the current version.

Another mechanism that is absent in the standard configurator is the mechanism for checking settings. Each configuration section implements the IValidator interface, which contains a single Validate method that returns a Boolean value and takes an instance of the InfoBuffer class as a parameter, which in turn is a collection of Info messages. Each message contains a Message text and a message "level" of the same type that was used by the logger for this purpose.

3.3 Multithreading components

Unlike the logger and configurator, which are integral subsystems whose components interact closely with each other, the multithreading support components in RW.Ring are rather loosely coupled. They are either small extensions of the .NET Framework standard library, making it more convenient to use, or abstraction layers on top of which more complex elements of the RW.Ring infrastructure are built.

The most commonly used component from the first category is the SharedLock class, which is a wrapper over a class from the ReaderWriterLockSlim standard library and has a more convenient API than the standard class. The original solution used in the implementation of this class is the use of the using syntax, based on the IDisposable interface. The WriteLock, ReadLock, and UpgradeableLock methods of this class return instances of the IDisposable interface, which is implemented in such a way that the Dispose method will release the requested lock. This allows to use the using statement in the same way of the lock statement usage when using the standard library Monitor class, as shown in Listing 1.

Listing 1. An example of using the SharedLock class.

```
SharedLock myLock = new SharedLock();
using (myLock.WriteLock())
{
    DoSomething();
}
```

This class is used in all classes of the RW.Ring platform where shared locking is required, since its implementation is such that it has minimal overhead compared to the class from the standard library.

Another class that serves the same purpose is the ThreadField class. While the standard .NET Framework mechanisms allow to have own instance of a static class field per thread, there is no way to do this with instance fields by default. The generic ThreadField class is intended to implement this feature, the generic parameter of which must be a field type, which must have its own value in each thread. This value is available through the Instance property for reading and writing.

One of the drawbacks of the .NET Framework standard library is the lack of a centralized repository of threads currently active in a program. The ThreadTracker static class uses the IThreadCollection interface to overcome this. This class allows to register

threads with the RegisterThread method and remove them from the list of registered threads using the RemoveThread method, as well as define global handlers that will be applied to all registered threads using the AddThreadHandlers method. This mechanism allows for reduced connectivity between other components that use multithreading.

The IOperationWrapper and ISynchorizeProvider interfaces are basic tools that abstract the fetch method into synchronization objects, allowing all the necessary operations to be performed on the main thread of the application. One of the implementations of the IOperationWrapper interface is designed to perform specified operations in a single-threaded apartment, even if it was originally requested in a multi-threaded apartment. This may be necessary in situations where production process models or support functions such as a daily weather generator are implemented in unmanaged libraries to access these functions from background streams. The IProgressIndicator interface is designed to notify about the progress of operations performed on the background thread.

3.4 Assembly classifier

With the .NET technology stack, there is the following hierarchy in the structure of a running application: an application consists of application domains, an application domain consists of assemblies, and an assembly consists of modules. An application domain is a part of an application that can be completely unloaded without restarting the program (except for the main application domain). An assembly is an executable file or group of files that is uploaded to a domain as a whole. A module is a single executable file with or without an entry point, which is either a whole assembly or a part of it. However, many applications consist of only one application domain, and most assemblies consist of one module; however, typically, many assemblies are loaded into any domain when an application runs. In this regard, the classifier in RW.Ring is made specifically for assemblies.

The assembly classifier contains information about any assemblies loaded in the current application domain, as well as about plugins loaded with a special call. Another function of the classifier is the sharing of services between application domains, which is implemented in such a way that the connectivity between the classifier itself and the services being shared is minimal.

An assembly classifier is an interface that contains the following members:

- TrackingAssemblies - Assemblies to be tracked. These are all assemblies in the current domain that satisfy the filter condition. Represents a read-only collection of instances of the Assembly class from the standard library. It is assumed that this list contains assemblies of the current application, but not using third-party libraries other than RW.Ring;
- ExcludePrefixes is a filter condition by which assemblies are added to the TrackingAssemblies list when loaded into the application domain. It is a list of assembly name prefixes that do not need to be included in TrackingAssemblies. The implementation of this list uses a prefix symbol tree based on hash tables. This provides the fastest way to check if an assembly name starts with one of these prefixes, which is computationally O (N), where N is the length of the longest prefix. By default, this condition contains the .NET Framework standard library prefixes;
- Plugins - the list of plugins loaded by calling the LoadPlugins method;
- UnmanagedAssemblies - the list of assembly names that could not be loaded when loading plugins;
- PluginsDirectory is the name of the directory in which the LoadPlugins method will look for plugins;

- LoadPlugins - method that loads plugins. It has two parameters:
  - searchPattern - pattern for searching plugin files. In general, a plugin file is a small, lightweight xml file that tells you which assembly to load;
  - mode - which application domain to load plugin assemblies into. The parameter has a special enumerated type that can take one of the following values:
    - CurrentDomain - load assemblies into the current application domain;
    - SeparateDomain - load all plugin assemblies into one separate application domain;
    - DomainPerPlugin - load each plugin assembly into its own separate application domain;
- LoadDependencies - loads all dependencies of the specified assembly into the current application domain with a filter specified through ExcludePrefixes;
- ShareServices - sharing all services between application domains. This operation is automatically called when plugins are loaded into other application domains, but can be called separately for the specified domain.

The assembly classifier also provides an assembly initialization mechanism. For this operation LibraryInitializer attribute should be used, which is needed to mark the assembly and specify the class in its parameter, the static constructor must be called when the assembly is initialized. Assembly initializers on load will be called for all assemblies that appear in the TrackingAssemblies list.

The basic implementation of the IAssemblyClassifier interface is the AssemblyClassifier class, which contains a virtual GetPluginInfo method for reading the plugin file. By default, this is an xml file with a root plugin element and assembly and name attributes. The assembly attribute must contain the name of the assembly file located in the same directory as the plugin file, and the name attribute value can be arbitrary and is intended for displaying plugins in the user interface. The PluginInfo class object to read from this file also contains the assembly file name and the plugin name.

The class in which these services are located, must meet the following conditions to share services:

- The class must be marked with the AppDomainShare attribute;
- The class must be declared in the same assembly that implements the Notung.dll library functionality. This can be either the assembly Notung.dll itself or any assembly into which individual components are imported from there;
- The class must contain a static method Share, which takes a domain as a parameter, which is either created by the assembly classifier when loading plugins, or is explicitly called from the assembly classifier.

This component allows to create an extensible architecture and load plugins with minimal overhead. This allows for a flexible plug-in architecture

4 Results and discussions

At present, a program for statistical analysis Schicksal has been developed on the basis of the RW.Ring platform. This program uses both those components that are described in this research, as well as those that remained outside its scope. The program uses a configurator to store settings for displaying colors of various levels of significance in the user interface and a list of recent files that the user has worked with. The program uses a logger for logging calculations, since during statistical calculations erroneous situations are possible, and convenient access to the program logs allows you to analyze their causes. The program

uses multithreading components to perform lengthy statistical calculations in a background thread with a dialog that displays the progress of the operation. The program uses an assembly classifier to provide support for plugins that are used to import data from external sources. The software included in the package contains a single plug-in that allows you to import data from Excel based on OleDb technology. In this case, the plug-in assembly contains two different data imports from Excel. In the first one, the import is carried out from the Excel range as is, i.e. each column represents either a factor influencing statistically processed features, or some of the statistically processed features, and each row is a tuple of the values of all factors and features. In the second case, the imported Excel table is considered as a matrix with named rows and columns, and when importing, a table with three columns is created: the row name of the imported matrix, the column name of the imported matrix, the cell value at the intersection of the corresponding column and the corresponding column. However, the presence of a plug-in architecture as a whole allows its functionality to be extended without recompilation.

The program loads quickly and also responsive to user actions even on old computers, which indicates high performance of RW.Ring libraries and good bootloader optimization in particular. All standard dialogs are shown through RW.Ring services, which allows you to change the technology for developing a graphical interface with minimal effort. It should be noted that many of the components of the RW.Ring platform were finalized during the development of the Schicksal program. Dividing RW.Ring libraries into components and infrastructure allows to cover individual components with tests and thereby guarantee their stable operation. In general, these libraries improve the experience of using the .NET framework and make working with this system more reliable and comfortable. Since the platform is not written as a new development, but is the next generation of the core of the APEX polyvariant calculation system, it takes into account the experience of using technologies gained during the development of APEX and corrects unsuccessful solutions. This allows the developed platform to be used for a wide range of tasks, which are not limited to agricultural modeling, but also in the tasks of land improvement, construction of structures, and in general wherever modeling using ensemble calculations may be required.

5 Conclusion

A high-performance, efficient platform has been developed to create powerful yet lightweight applications for a wide range of tasks. The platform's performance is confirmed by the successful development of the Schicksal statistical analysis program. In the future, the platform will develop in parallel with the new version of the APEX polyvariant analysis system, as well as other programs based on it. The RW.Ring platform can be recommended as a set of standard libraries designed for building any shells for a large number of models.

References

1. R.A. Poluektov, S.M. Fintushal, I.V. Oparina, D.V. Shatskikh, V.V. Terleev, E.T. Zakharova, Archives of Agronomy and Soil Science **48(6)**, 609-635 (2002
2. I. Dunaieva, E. Barbotkina, V. Vecherkov, V. Popovych, V. Pashtetsky, V. Terleev, A. Nikonorov, L. Akimov, Advances in Intelligent Systems and Computing **1259**, 198-206 (2021
3. I. Dunaieva, W. Mirschel, V. Popovych, V. Pashtetsky, E. Golovastova, V. Vecherkov, A. Melnichuk, V. Terleev, A. Nikonorov, R. Ginevsky, V. Lazarev, A. Topaj, Advances in Intelligent Systems and Computing **983**, 236-246 (2019)
4. A. Topaj, W. Mirschel, Computers and Electronics in Agriculture **145**, 199-207 (2018)

5. S. Medvedev, A. Topaj, IFIP Advances in Information and Communication Technology **359**, 295-301 (2011)
6. A. Ceglar, R. van der Wijngaart, A. de Wit, R. Lecerf, H. Boogaard, L. Seguini, M. van den Berg, A. Toreti, M. Zampieri, D. Fumagalli, B. Barutha, Agricultural Systems **168**, 168-180 (2019)
7. S. Medvedev, A. Topaj, V. Badenko, V. Terleev, IFIP Advances in Information and Communication Technology **448**, 252-261 (2015)
8. V. Badenko, G. Badenko, A. Topaj, S. Medvedev, E. Zakharova, V. Terleev, Environments **4(4)**, 92 (2017)
9. S.A Medvedev, A.S. Cheryaev, Agrophysics **3**, 45-52 (2020)
10. N. Arefiev, M. Mikhalev, D. Zotov, K. Zotov, N. Vatin, O. Nikonova, O. Skvortsova, S. Pavlov, T. Chashina, T. Kuchurina, V. Terleev, V. Badenko, Y. Volkova, V. Salikov, K. Strelets, M. Petrochenko, A. Rechinsky, Procedia Engineering **117**, 32-38 (2015)
11. R.A. Poluektov, V.V. Terleev, Russian Meteorology and Hydrology **11**, 70-75 (2002)
12. R.A. Poluektov, I.V. Oparina, V.V. Terleev, Russian Meteorology and Hydrology **11**, 61-67 (2003)
13. R.A. Poluektov, V.V. Terleev, Russian Meteorology and Hydrology **12**, 73-77 (2005)
14. V.V. Terleev, A.G. Topaj, W. Mirschel, Russian Meteorology and Hydrology **40(4)**, 278-285 (2015)
15. V. Terleev, A. Nikonorov, I. Togo, Y. Volkova, V. Garmanov, D. Shishov, V. Pavlova, N. Semenova, W. Mirschel, Procedia Engineering **165**, 1776-1783 (2016)
16. V. Terleev, A. Nikonorov, V. Badenko, I. Guseva, Y. Volkova, O. Skvortsova, S. Pavlov, W. Mirschel, Advances in Civil Engineering **2016**, 8176728 (2016)
17. R. Gupta, D.K. Mishra, S. Tokekar, ICETET **2009**, 346-351 (2009)
18. *Log4net, "Logging Services TM", The Apache log4net project* (2019) https://logging.apache.org/log4net/