"Mircea cel Batran" Naval Academy Scientific Bulletin, Volume XIV – 2011 – Issue 2
Published by "Mircea cel Batran" Naval Academy Press, Constanta, Romania

IMPLEMENTATION OF A GRAPHICAL ENVIRONMENT FOR SIMULATION OF MATHEMATICAL MODELS

NIkola VALCHANOV[1]
Anton ILIEV[2]
[1] Plovdiv University, Department of Applied Mathematics and Modeling, Bulgaria
[2] Plovdiv University, Department of Applied Mathematics and Modeling, Bulgaria

Abstract: *This paper follows through the implementation of a graphical environment for simulation of mathematical models. It deals with design and implementation issues of this type of applications. The paper offers a plug-in mechanism for dynamic use of pluggable computational libraries and performance optimization techniques for distribution of computations for various simulations between system nodes. The environment comprises of a graphical designer for building mathematical models using stock-flow diagrams, data extractor for converting the diagram to a parser readable string, mathematical core for running the simulations and a set of tools for adequate display of the simulation results.*
Keywords: *graphical simulation environment, stock-flow diagrams, mathematical model simulation, software architecture, distributed systems.*

INTODUCTION

The computer simulation tools have become an integral part of the abstract models research process because of the evolution of the information technologies. These tools allow us the extraction of behavioral information for a given conceptual model by using real–time modeling (simulation). The contemporary simulation environments [2, 3, 4] offer instruments for building abstract models, conduction of simulations and analysis of the extracted information. These types of systems are vastly used in both theoretical and practical problems. This paper describes the process of building a graphical environment for simulation of mathematical models by using information system architecture. It proves the existence of benefits from using the suggested information system model and its usefulness in various fields of the information technologies.

MATHEMATICAL FOUNDATIONS

The graphical environment for mathematical models simulation focuses on mathematical models, described by differential equations. Note an example of this type of differential equations is the so called system of Lotka–Volterra equations, which describe a model of the predator–prey biological system.
The simulation of such models is often done by application of iteration numerical methods. In this paper we will apply both well known in the literature Euler method and Runge–Kutta method for the initial value problem of first order ordinary differential equation of the type:

$$Y' = f(t, Y) \qquad Y(t_0) = Y_0$$

where Y is scalar.
Now we will give a brief description of the used numerical methods:

Euler method

Euler method is one of the simplest numerical methods for solving initial value problems of ordinary differential equations
The iteration's value Y_{n+1} in a given moment t_{n+1} is calculated based on the step h and the value Y_n retrieved by the previous iteration by $Y_{n+1} = Y_n + hf(t_n, Y_n)$

Common fourth order Runge–Kutta method

The common fourth order Runge–Kutta method (RK4) is one of the most commonly used methods of its kind. It is far more precise than the Euler method. The computation of the next iteration's value Y_{n+1} in a given moment t_{n+1} is calculated based on the step h and the previous value Y_n by:

$$Y_{n+1} = Y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad t_{n+1} = t_n + h$$

where

$$k_1 = hf(t_n, Y_n)$$
$$k_2 = hf\left(t_n + \frac{1}{2}h, Y_n + \frac{1}{2}k_1\right)$$
$$k_3 = hf\left(t_n + \frac{1}{2}h, Y_n + \frac{1}{2}k_2\right)$$
$$k_4 = hf\left(t_n + h, Y_n + k_3\right)$$

SYSTEM REQUIREMENTS

All graphical environments for mathematical models simulation offer tools that simplify the construction of models, their simulation and the analysis of the simulation results. The main challenges in this type of systems are:

- Finding **an adequate representation of the mathematical models** that is intuitive and easy to use by the end user,
- **Building a pluggable computational libraries mechanism**, that simplifies the integration of new numerical methods in the system and gives to the end user the possibility of choosing the best fitted method for each simulation,
- **Means for distributed execution of the simulations**, which allow the end users to continue their work while a given simulation process is still active,
- Choosing the **right tools for representation of the simulation results**, facilitating the model analysis process.

Adequate representation of the mathematical model

The method for visual representation of the mathematical models should be coordinated with the system specifications. One of the most intuitive methods for representation of mathematical models, described by differential equations, is the Stock–Flow diagram [7]. It is one of the simplest methods for definition and analysis of dynamic systems.

The idea of all Stock–Flow constructions is based on the assumption that every dynamic behavior is manifested when flows are accumulated into stocks.

Every Stock represent an entity accumulated in time by incoming flows and/or depleted by outgoing flows. The quantities in the stocks can be affected only by flows.

The flows change the quantities in the stocks in time. Usually there is a clear distinction between incoming and outgoing flows. The flow is defined by its debit which is recalculated on every time interval.

Pluggable computational libraries mechanism

One of the problems in mathematical model simulation systems is the integration of new numerical methods which is connected with the abundance of various numerical methods some better suited for a specific purpose than others.

The mechanisms for pluggable tools are a vastly used approach for managing application tools and allowing third party developers to produce custom pluggable instruments. These mechanisms index the available tools automatically and give the system a unified way to access them. The system structure does not change when a new tool is added and the tools themselves are based on an open specification common for the system.

This kind of indexing mechanism can be used for computational libraries management in the graphical simulation environment for mathematical models.

Means for distributed execution of the simulations

One of the performance criteria of computations driven systems is their ability to work in a cluster and distribute the computations amongst different system nodes.

The mathematical computations used in the numerical methods described earlier cannot be divided into synchronizable subproblems. Therefore the method of computations distribution can be described in a simple system use–case by the following way:

1. The application indexes all available nodes in the cluster, providing tools for simulation of mathematical models of that type;
2. The end user chooses a specific computational library for conducting the simulation process and starts the simulation;
3. The system finds the most suitable node offering the desired computational library creates an instance and provides it with an adequate input;
4. The terminal running the user interface is not blocked by CPU intensive computations while the tool simulates the model on the specific cluster node. This way the end user is allowed to continue his work on the model or start new simulations.

The right tools for representation of the simulation results

The tools for graphical representation of the simulation results play a crucial role in the mathematical model analysis process. These tools facilitate the identification of correlations between the members of the researched model.

The numerical methods, used in the graphical simulation environment for mathematical models provide information about the model participants per iteration. The natural way is displaying the obtained data by a tabular structure. At the same time the tabular view is not always the most productive way of data representation.

The graphical representation provides a different, visual look on the results that often gives a better perspective to the viewer. The numerical methods mentioned above allow us the construction of graphical description that follows the changes in a given model's member in time.

SYSTEM IMPLEMENTATION

Based on the system requirements we can divide the system implementation in the following way:

1. Building a module for simulation of mathematical models that takes an input string, containing the list of differential equations and initial values, which describe the model;

1.1. Building a parser that extracts the system of differential equations and initial values from the input string;

1.2. Building a mechanism that allows the use of pluggable computational libraries for model simulation;

1.3. Developing algorithms for mathematical model simulation using the Euler and RK4 methods and implementation of pluggable computational libraries for each one of them.

2. Designing a graphical component for building and editing of mathematical models by Stock–Flow diagrams;

3. Building the graphical component in the context of the information system architecture suggested in [8].

3.1. Establishing the communication between the graphical component and the model simulation module;

3.2. Building tools for tabular and graphical representation of simulation results;

3.3. Implementation of tools for storing mathematical models and their simulation results;

3.4. Implementation of distributed execution of the different simulations.

Building a module for mathematical model simulation

The internal dependencies between the different tasks, related to the system implementation, require the strict definition of a string format for representing differential equations and initial values for model's members.

Let us use the following syntax:

*Hares'=Hares*50/100*
*-(Hares*20/100)*
Hares=100

Once the syntax is fixed we can proceed to the implementation of a parser for extracting differential equations and initial values from the input string. The parser builds two associative arrays – one with initial values, and another with equations. The keys in both arrays are the names of the corresponding model's members.

In order to simplify the development and integration of new computational libraries in the module for simulation of mathematical models we need to build a dynamic loader. The module provides an interface that makes computational library integration possible.

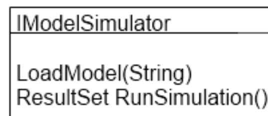

Figure 1. IModelSimulator interface

The module configuration contains the physical path to the folder with the pluggable computational libraries. When the module is accessed for the first time all files in that folder are indexed and the module tries to load them as assemblies. If successful the loaded assemblies are checked for compatibility and if compatible the module extracts information for the specific numerical method that each assembly implements. In the end the module obtains a list of all the available numerical methods.

When executing a simulation the client code feeds the mathematical model to the module using the input string and specifies the desired numerical method that should be used for the simulation. The assembly implementing the method is extracted from the prebuilt list. After the assembly identification the module searches it for IModelSimulator interface implementations. When multiple implementations are found the module works with the first. After the implementation is found a member is instantiated and the simulation is run.

Each mathematical library implements a specific numerical method. Two libraries are built for the implementation of the graphical simulation environment for mathematical models. The first one is connected with Euler method and the second - with RK4 method. Because of the similarities in the algorithmization of the mathematical apparatus of the two methods we will review only the differences.

The expected input parameters for the computational tools are:

- **Step** – used in the corresponding numerical method;
- **Time** – upper boundary marking the end of the simulation process;
- **Two associative arrays** – one with the initial values for the model’s members and another with their differential equations.

The storage structure of the result from the simulation process is also an associative array of lists containing the values from the simulation iterations for every model’s member.

The algorithm that implements the mentioned numerical methods can be described with the following block–diagram:

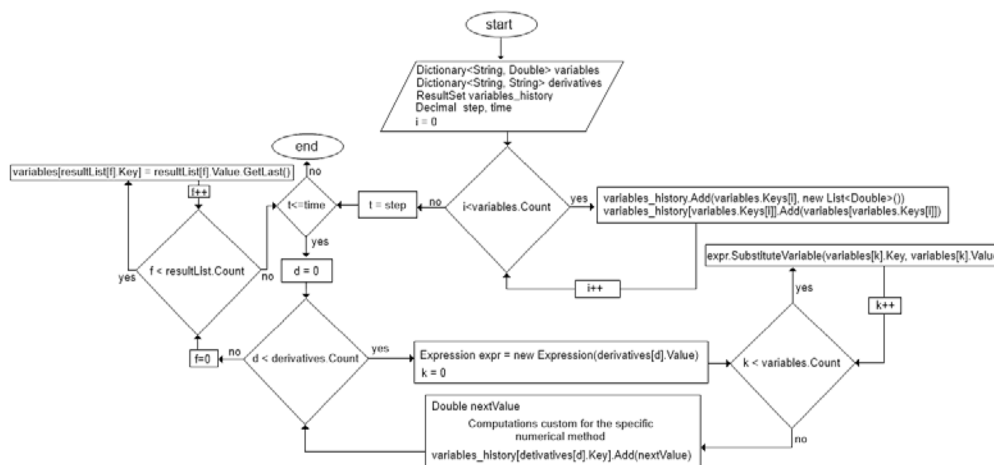

Figure 2. Algorithm used in the implemented computational libraries

After initial values extraction we create the data structure that will store the simulation result. We initialize the structure with elements – one for every model’s member. The keys of the elements of the associative arrays are the names of the model’s members. They are all initialized with an empty list of real numbers. The next stage of the algorithm execution is computing of the numerical method iterations until the end of the predefined interval.

When a specific model’s member is being processed, its differential equation is extracted. The values of the model’s members computed by the previous iteration are substituted in the equation, except the currently processed member.

When **Euler method** is applied the currently processed member is substituted by its value from the previous iteration. The result of the equation (assigned to the nextValue variable from the block–diagram) is stored in the result structure and it is replaced in the array of current values of the model’s members.

When **RK4 method** is applied we first calculate the four auxiliary values. The model’s member for the current iteration is then computed based on them. The calculated value is stored in the result structure and it is replaced in the array of current values of the model’s members.

Designing a graphical component for building of mathematical models

In order to create a graphical editor that provides tools for building Stock–Flow diagrams we first need to define architectural and functional requirements towards it.

Architectural requirements:

- Simplified integration of new element types in the graphical tool for building mathematical models;
- Facilitated element handling.

Functional requirements:

- Definition of the building blocks of the diagrams
- Managing diagram elements – add, select, move, edit element;
- Connecting diagram elements;
- Connecting outgoing/incoming flow elements to stock elements;
- Using elements for simulation results visualization;
- Resizing elements for simulation results visualization;
- Extracting diagram information in an adequate format.

The architectural requirements are followed during the system implementation in order to set strong foundations and simplify the support of the tool.
The tool provides a seamless way of working with the different diagram elements. This is achieved using interfaces for the different functionalities that the element types implement. These interfaces place an abstraction over the element types and facilitate their management.
This abstraction layer makes adding a new element type as easy as creating a new class which implements a set of interfaces based on the desired element type functionality.
In order to meet the functional requirements we need to specify the building blocks of the mathematical models diagrams.

Stock – element representing a member in the mathematical model. It can be connected to incoming and outgoing flows describing the process of accumulation;
Flow – element describing the rate of accumulation/depletion of a given entity in the stocks. The equations, describing the dynamic behavior of the model's members are defined by flows;
Coefficient – element used to encapsulate computational modules in the model. Specific blocks of given formulas can be encapsulated in coefficients which in turn facilitate the modifications of the mathematical models.
The set of interfaces includes the operations: **drawing; transformation; editing; connecting; resizing; extraction of element information in adequate format**.

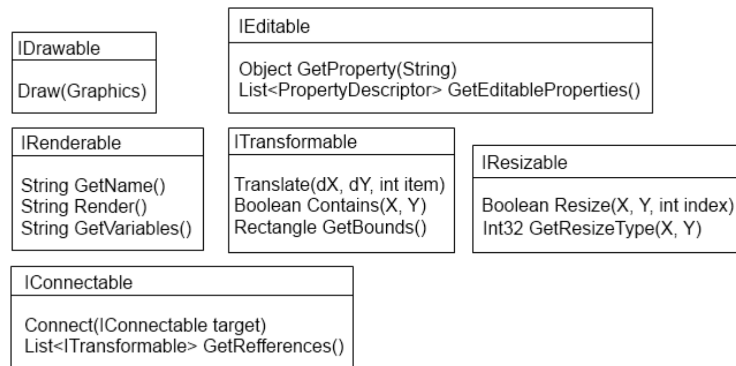

Figure 3. Set of interfaces for the different functionalities of the diagram elements

A collection of IDrawable elements is kept in the tool. When a specific operation needs to be executed on a given element the collection items are casted to ITransformable to identify the selected element. When the element is found it's casted to the interface corresponding to the operation that needs to be executed.
Connection of elements is done by using the IConnectable interface. Its method Connect connects two elements in an oriented way accepting the target element as a parameter. Once connected the start element stores a reference to the target element. This allows the starting element to use the target element in its expression.
The element types that implement the IEditable interface can be edited. The interface obligates its implementations to expose information about their editable properties using PropertyDescriptor instances storing property name, property type and providing means for extraction and manipulation of property values of any instance of this type. This way the IEditable implementations premise the implementation of editable forms with automatic user interface generation based on the editable properties of the currently edited IEditable instance.
When connecting a flow with a stock the stock stores a reference to the flow. The stock treats the flow as incoming or outgoing based on the way the flow has been connected.
In the implemented graphical simulation environment for mathematical models there are two main types of visualization tools for simulation data display – graphical and tabular. These tools implement all interfaces excluding the IRenderable. Although they are diagram elements, they do not hold model specific information and are ignored in the process of extraction of information for the mathematical model from the diagram elements.
The data visualization tools are the only elements that implement the IResizable interface. The information about the possible resize methods for a given item is available through GetResizeType(X, Y), which returns information about the available resize axes for a given point within the diagram element.

The IRenderable interface is used to implement the extraction of information about the mathematical model from the diagram elements. The algorithm that extracts the information starts in the stock elements. The Render() method is called for every stock element. The result is a string of two lines that define the initial value and the differential equation describing the model's member. The crawl of the model elements connected to the stock and the generation of the initial value line are done in the following few steps:

1. When the Render method in the stock is called it first builds up the initial value line. It is composed of the name of the stock and its initial value (both present in the stock element)
2. In order to build the differential equation the stock calls the Render() method of its incoming and outgoing flows. The equation is build in the following manner:
Model member' = (incoming flow expression)-(outgoing flow expression)
3. When the Render() method in the incoming/outgoing flow is called, the flow checks if its expression contains references to model variables other then the model's members. If such model variables exist, they are found in the collection of references that stores the elements used in the flow expression. The replacement of these elements is done by substituting their references in the expression with the result obtained from calling their Render() methods.
4. When the Render() method of an element of type coefficient is called, the coefficient checks if its expression contains references to model variables other then the model's members. If such model variables exist they are found in the collection of references that stores the elements used in the coefficient expression. The replacement of these elements is done by substituting their references in the expression with the result obtained from calling their Render() method.

After the algorithm's execution, the tool passes the obtained information to the module for simulation of mathematical models. After the execution of the simulation the module returns the set of all points for every model's member. The

obtained information is passed to the visualization tools that display it in tabular or graphical way.

Building the graphical component in the context of information system architecture

The information system architecture described in [8] offers standard three layer application architecture. The business logic contains a mechanism for module management. The system architecture comes with a set of implemented base instruments comprising of interfaces defining base structure types supported by the system, base visual components that use business layer tools that implement the supported system types and tools for distributed execution of system layers in a cluster. The expected benefits from using the suggested architecture are code reuse and means for distributed execution of model simulation.

According to the initial design of the graphical component we created classes for each item type of the diagram and interfaces for the different functionalities:

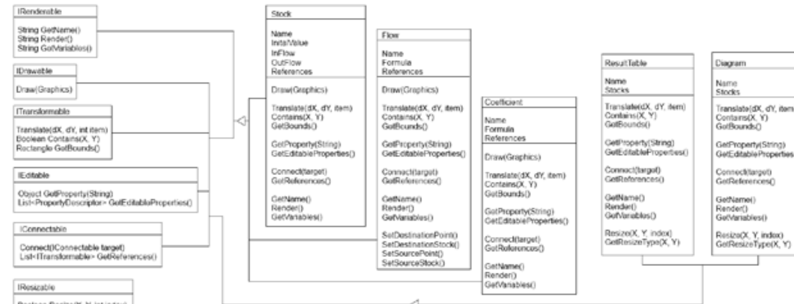

Figure 4. Class diagram of the classes and interfaces of the graphical component

The graphical component for building and editing mathematical models was integrated in an edit form so that information system architecture can be used for the graphical simulation environment for mathematical models. In order to be fully integrated as a custom editor the tool implemented the three main functionalities that information system model demanded – validation of the diagram, methods for extracting the diagram and the simulation results in a serialized manner suitable for storing in a data source and populating diagram based on serialized information extracted from the data source.

In order to facilitate communication between layers information systems use custom data structures that follow the data model. These structures are based on the concept of Entity–relationship models. For the representation of mathematical models and their simulations results we created the MathModel and ModelSimulationDataSet classes:

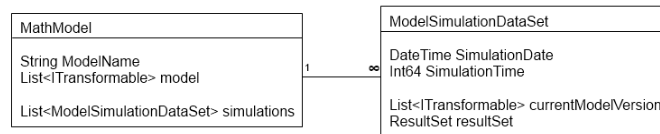

Figure 5. Entities for communication between layers

The MathModel class stores information about the current version of the mathematical model. It stores the name of the model as a string and the elements of the model in a collection of ITransformable items. The ModelSimulationDataSet class stores the result of a conducted simulation for a specific version of the model. The current version of the model could differ from the version of the simulation. The model versions for each simulation are stored only for consistency purposes.

In order to follow suggested information system architecture we extended its business logic layer with the MathModellingBL class. It implements the structural type master–details (IMasterDetailsBL interface). The master items type in this implementation is the MathModel class, the details are implemented as a collection of ModelSimulationDataSet instances.

In order to implement the communication with the data source when working with the MathModel and ModelSimulationDataSet classes we created the MathModellingDA class in the data access layer of the suggested information system architecture. It handles the communication with the data source and isolates its inner work from the business layer of the application.

Since we're strictly following the architecture suggested in [8] we could implement a distributed model of execution of the system layers. Every layer executes on a different system node in a cluster providing means for remote instantiation and access to its tools. The business layer implements the mechanism for module management within the system and provides access to the mathematical model simulation module.

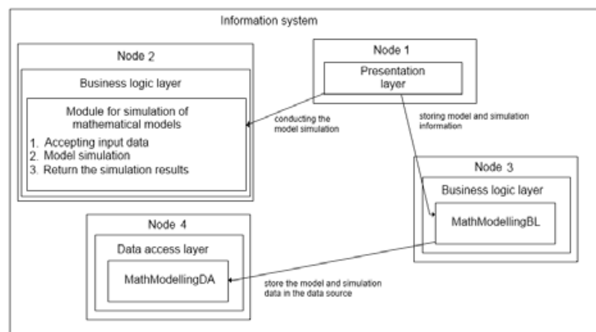

Figure 6. Distributed model of execution of the system layers

For the implementation of the graphical user interface we extended the existing presentation layer of the suggested information system model. We used the base visual components that work with the interfaces for the different structural types. The two main visual components in the system for building and simulation of mathematical models are a form for visualization of all stored mathematical models and information, obtained from their simulations and a visual component for editing mathematical models, containing the tool for building and editing of mathematical models.

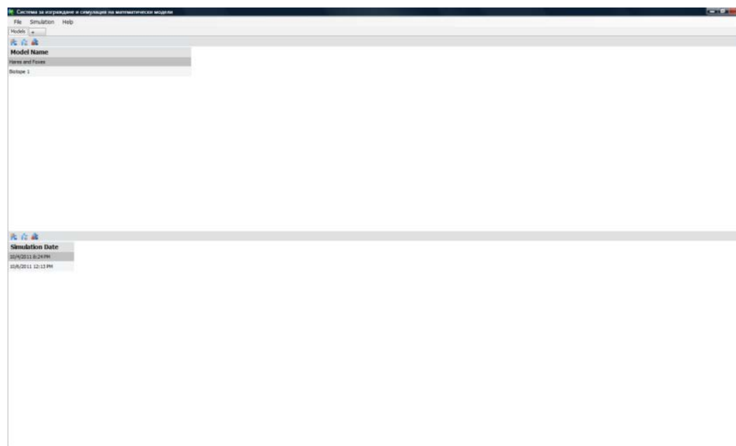

Figure 7. Form for visualization of all stored mathematical models

The functionality of the form for visualization of all stored mathematical models inherits from the base visual component for structural type master–details and works with the MathModellingBL class. Since most of the operations (record navigation, create, edit, delete, save, update) are common for all types of records within an information system the base visual components implement it and use the interfaces for the different structural types to extract the needed information from the business logic classes thus stimulating code reuse.

The edit form takes a Math Model instance and loads it in the visual tool for building mathematical models.

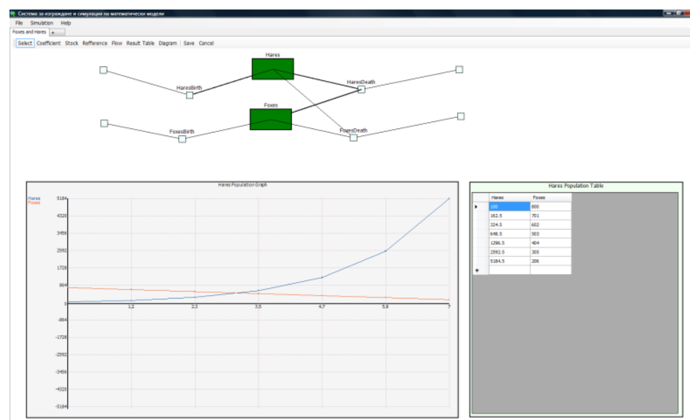

Figure 8. Form for editing mathematical models

The edit form frm Math Model Edit inherits from the base edit form visual component and implements extracting model information form controls into a Math Model instance, extracting model information from Math Model and populating controls and data validation.

CONCLUSION

The process of the implementation of graphical simulation environment for mathematical models was followed through. The result was a functional graphical simulation environment. The suggested design for the graphical component for building mathematical models is applicable in multi-purpose diagrammer component implementation.

The suggested application design included plug-in mechanisms for dynamic use of pluggable computational libraries. The mathematical foundations of the simulation problem were analyzed. The developed algorithms were used to create computational libraries for mathematical models simulation.

A model for distribution of the simulation process between system nodes was suggested. The use of information system architecture for the implementation of the graphical simulation environment for mathematical models allowed code reuse and saved time and resources during implementation.

REFERENCES

[1] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, 1994
[2] Vensim simulation software, http://www.vensim.com
[3] Stella simulation software, http://www.iseesystems.com
[4] AnyLogic simulation software, http://www.xjtek.com/
[5] Microsoft Corporation, Provider Model Design Pattern and Specification, http://msdn.microsoft.com/en-us/library/ms972319.aspx.
[6] J. Hart, Mathew B., Gilani S., Gillespie M., Olsen A., Visual Basic .NET Reflection Handbook
[7] G. Ossimitz, Stock-Flow-Thinking and Reading stock-flow-related Graphs: An Empirical Investigation in Dynamic Thinking Abilities, System Dynamics Conference, Palermo, Italy, 2002
[8] N. Valchanov, T. Terzieva, V. Shkurtov, A. Iliev, Approaches in building and supporting business information systems, Information technologies in business and management, 16-17 October 2009, Varna, Bulgaria, 100-106