Hydrol. Earth Syst. Sci. Discuss., 4, 75–89, 2007
www.hydrol-earth-syst-sci-discuss.net/4/75/2007/

Hydrology and Earth System Sciences Discussions

The architecture and prototype implementation of the Model Environment system

G. Donchyts[1,*], D. Treebushny[1], A. Primachenko[1], N. Shlyahtun[1], and M. Zheleznyak[1]

[1]Ukrainian Center of Environmental and Water Projects, Kiev, Ukraine
[*]now at: WL/Delft Hydraulics, Delft, The Netherlands

Received: 16 October 2006 – Accepted: 16 November 2006 – Published: 18 January 2007

Correspondence to: G. Donchyts (gennadiy.donchyts@gmail.com)

Abstract

An approach that simplifies software development of the model based decision support systems for environmental management has been introduced. The approach is based on definition and management of metadata and data related to computational model without losing data semantics and proposed methods of integration of the new modules into the information system and their management. An architecture of the integrated modelling system is presented. The proposed architecture has been implemented as a prototype of integrated modelling system using. NET/Gtk# and is currently being used to re-design European Decision Support System for Nuclear Emergency Management RODOS (http://www.rodos.fzk.de) using Java/Swing.

1 Introduction

In environmental applications a mathematical model is understood as a numerical engine which provides calculations by processing different types of data related to some physical quantities used as a model variables, parameters, results etc.

In a complex Modelling System the most crucial part is the development of a Software Framework (Wikipedia, Software Framework). Indeed, the quality of the Software Framework defines flexibility, extensibility and scalability of the Modelling System. The most important concepts which should be implemented within such a framework are:

- Data and Metadata
- Integration of Modules providing some numerical simulations
- Well-defined Data Management, Graphical User Interface and GIS components
- Use of the EAA patterns (Fowler, 2002)

The first approximation of seamless modules integration is exchanging variables data between different modules within one program. An important step towards standardization of the model linking was done in the project HarmonIT funded by European Commission. The project is directed onto the development of Open Modelling Interface (OpenMI) – a software standard that enables OpenMI compliant models to exchange data at run time (Moore et al., 2004). It has to be noted that OpenMI can be used to link several modules that exchange their data during runtime, but it introduces restrictions which prevents it to be the central library of an Integrated Modelling Framework. One of the main limitations is the lack of detailed semantic information in the runtime data. By looking at the ExchangeItem we can identify location (where) and a physical quantity (what) which are required or can be provided by a model to describe data but we can not say how values of exchanged item are defined. Thus, initial conditions, boundary conditions, model parameters, river cross-section properties have the same meaning; moreover, it is possible to define them as a *VectorSet* or a *ValueSet* only. In the reality it is necessary to have much more information like how functional dependency is defined, if there is a time dependency or not, how to describe data using more than one quantity, grouping of data etc. Below are examples of typical input data for a one-dimensional river flow model which are difficult to describe in terms of OpenMI standard:

Initial Conditions (group)
$f_1(x),\ t = t_0$
$f_2(x),\ t = t_0$
Boundary Conditions (group)
$y(t),\ x = x_0$ - *water level as a function of time at the boundary location* x_0
$q(h),\ x = x_1$ - *water discharge as a function of water depth at the location* x_1
Model Parameters (group)
start time of the simulation
end time of the simulation
$g = 9.81\,\mathrm{m}^2/s$

On the other hand, by introducing a Framework, which can describe all possible types of input / output data in a generic way while preserving their meta information, the task of integrating modules becomes much simpler. In this case models are self describing. Among the important features which is required is that the Framework should allow a flexible way of models linking and exchange of their data at the design and run-time.

The first practical application of the proposed approach is the re-designed RODOS DSS – Real-time Online Decisions Support System for Nuclear Emergency Management (Bartzis et al., 2000). The RODOS system is a comprehensive integrated modelling system which integrates many of the environmental models and also provides real time monitoring data processing with the assimilation of these data to increase the predictive power of the models (Rojas-Palma et al., 2003). Previous version of the RODOS was designed for HP-UX environment and the main task of the system re-engineering is to transform the system into the multy-platform environment without principal changes in the source code of the models.

2 The architecture

The following technologies and approaches were used during design (see Fig. 1):

– N-tier and Plug-in based architecture.

– ORM (Object Relational Mapping) solution for system data management which allows to use different DBMS without making changes to the software code.

– GIS subsystem (using OpenGIS-compatible implementations).

The proposed system contains the following layers separated by functionality:

– ***Common*** common utility and common system data class libraries

– ***Core*** main/core system classes

– ***Data*** data access layer, implementation of generic data access to the system data

– ***Models*** defines IModel interface to be implemented by 3rd party models

– ***GUI*** graphical user interface

– ***GIS*** contains classes providing functionality to work with geospatial data

The system implements a plug-in based architecture. A plug-in is a library which can be easily embedded into the system only by putting it into the specific folder. Currently the system uses plug-in(s) to implement the data access functionality, integrate external models into the system and to provide different user interface views for data visualization. Each plug-in within the system can contain one or several Models (numerical kernels providing calculations) alongside with Views needed to represent data of the model (DataItems, Elements).

The core library of the system contains a set of classes which are used system-wide to access various system components. Main user document named Project embeds a set of Tasks. The Task class acts as a manager of a specific instances of Model. A Task can be considered as a scenario to run a defined computational model on a specified region.

3 Data and metadata (common library)

In the current work a unified definition of the model data is suggested. The simplest “atomic” data class DataItem is defined as a base entity for all types of data used in the system: parameters, input data, sources/sinks, initial/boundary conditions, results of the simulation etc. The data classes can be easily extended by new data types if needed (polymorphysm). The real values are provided by the classes derived form the DataItem and never by the DataItem itself. In this way all semantics of the data is preserved. Figure 2 shows main data entities of the system and defined data types.

The following objects are defined as common for the system:

- Quantity - physical quantity (e.g. concentration, pressure, velocity, density, ...)
- Unit - unit of measure for a certain quantity (m, m^2, Bq/m^3, ...)
- Element - used to define spatial element which can be used for data location (abstract element or specific grid cell, south boundary etc.).
- ElementCollection - model calculation grid defined as a collection of elements.
- DataItem –main base data class defined as a value of a certain quantity; can also have time, element and other properties (e.g. data role, text description, substance etc.).
- Series - used to represent any vector function
- TimeSeries - specific type of Series where first argument is always time
- ElementSeries - specific type of Series where first argument is always *Element*

Here is an example which describes an irregular rectangular model grid (1) with a set of vertices V and a set of cells C treated as Elements in the system (2), see also Figs. 3, 4. The set G defines corresponding ElementCollection.

$$\begin{aligned} G &= (V, C) \\ V &= \{v = (x, y)\} \\ C &= \left\{c = (v_1, v_2, v_3, v_4)\,,\ v_j \in V\right\} \end{aligned} \tag{1}$$

Two functions representing some physical quantities are defined on the specified sets of Elements:

$$\begin{aligned} F_1 &: V \rightarrow R \\ F_2 &: C \rightarrow R \end{aligned} \tag{2}$$

4 Integration of external models

The typical example of model integration is shown on the Fig. 5. The approach presented on a diagram can be recommended for usage in most cases when a model is implemented using native library. There are two main requirements for a model wrapper library:

- It should be implemented as a *Plugin*
- It should implement *IModel* interface

The first requirement makes it possible to plug-in the new models into the system. The second requirement ensures that model will be managed using well-defined interface. It is not required to implement *IModel* class from scratch, instead the framework provides default implementation as an abstract Model class which is recommended to be used as a basis for all new model wrapper classes. When system creates a new Task or loads a Task from a database the Model type is used to select a corresponding implementation (model wrapper) class. After that Task component asks Model to initialize its data. Depending on if a Model is loaded from a database or is newly created the Task component provides an existing set of the DataItems or creates a new empty data set. During this process the Model is responsible for correct initialization of all data. After Model/Task data are initialized a user can start simulation. When Model produces some results they can be immediately visualized in the corresponding data views and/or stored in a database etc.

5 GUI design

The proposed design of the graphical interface includes main controls shown on the Fig. 6. The system follows the *MVC (Model-View-Controller)* approach – the one that separates an application's data (model), user interface (view), and logic (controller) into three distinct components so that modifications to one component can be made with minimal impact to the others. The approach was implemented for the demo version of the Hydrological Dispersion Module of RODOS (Raskob et al., 2004) and then to the software reengineering of the whole RODOS system.

6 Conclusions

Development of the decision support systems for the environmental and emergency management as usual required integration under GUI different mathematical models coupled with data bases and GIS. Very often the legacy models /codes developed much earlier than the DSS is designed should be incorporated into the new software systems. An approach of the efficient design and implementation of such software system architecture has been presented. The core of the methodology is based on a detailed definition and management of the metadata and data related to the computational models without losing their semantics. This is achieved by implementation of the DataItem concept. The proposed approach can be extended to interact with existing OpenMI-compliant models by development of the component which wiil provide a bridge between IModel (current paper) and ILinkableComponent (OpenMI) interfaces. The methods of the integration of the new models into the information system is proposed that is based on the plug-in architecture, separation of the model management and its data editing/viewing. The GUI design was provided in the frame of MVC (Model-View-Controller) approach that separates an application's data (model), user interface (view), and logic (controller) into three distinct components. The prototype system based on the presented architecture was developed using NET/Gtk#. A first implementation of the approach for the multiple-module model based software system is providing, taking into account the prototype development experience, for the re-engineering of real-time online decisions support system for off-site nuclear emergency management within 6 Framework EC RTD program me project EURANOS (http://www.euranos.fzk.de) using Java/Swing. So the proposed architecture can be implemented in different object oriented software environment.

References

Bartzis, J., Ehrhardt, J., French, S., Lochard, J., Morrey, M., Papamichail, K. N., Sinkko, K., and Sohier, A.: RODOS: decision support for nuclear emergencies, in: Recent Developments and Applications in Decision Making, edited by: Zanakis, S. H., Doukidis, G., Zopounidis, C., Kluwer Academic Publishers, 379–394, Dordrecht, The Netherlands, 2000.

Booch, G.: Object-Oriented Analysis and Design with Applications (2nd Edition), Addison-Wesley Professional; 2 edition , ISBN:0805353402, (30 September, 1993).

Fowler, M.: Patterns of Enterprise Application Architecture, Addison-Wesley Professional, 2002.

Moore, R., Tindall, I., and Fortune, D.: Update on the HarmonIT project - The OpenMI Standard for model linking, Proc. Hydroinformatics, Singapore, 2, 1811–1818, 2004.

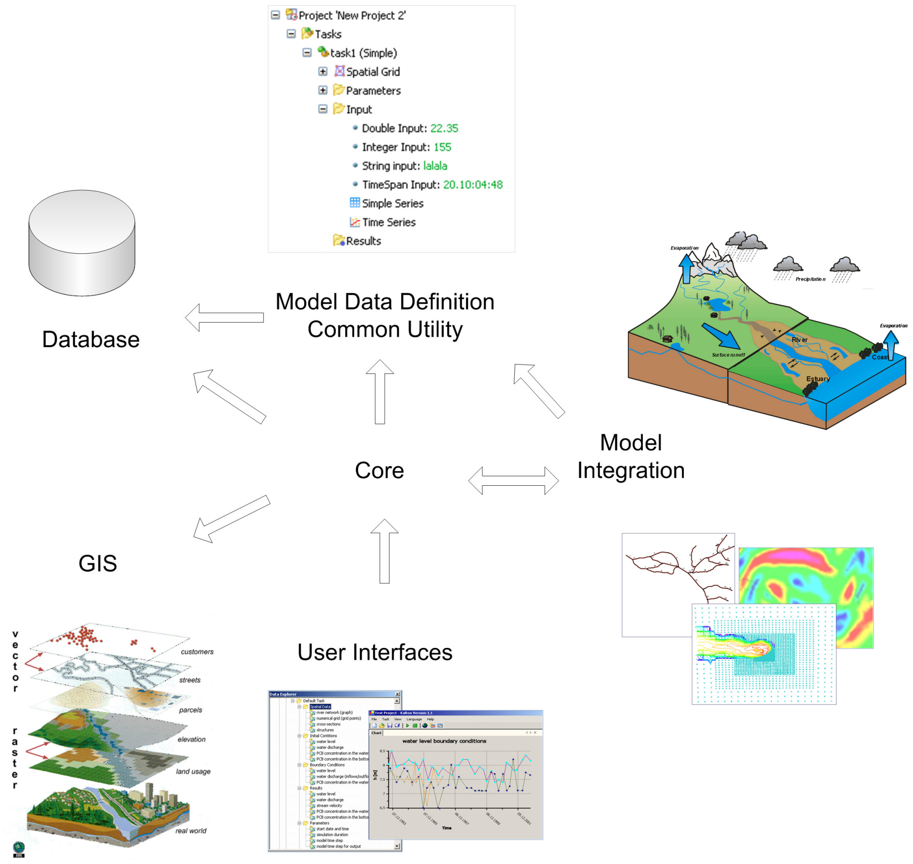
Raskob, W., Heling, R., and Zheleznyak, M.: Is there a need for hydrological modelling in decision support systems for nuclear emergencies, Radiation Protection Dosimetry, 109, no 1–2, 111–114, 2004.

Rojas-Palma, C., Madsen, H., Gering, F., Puch, R., Turcanu, C., Astrup, P., Muller, H., Richter, K., Zheleznyak, M., Treebushny, D., Kolomeev, M., Kamaev, D., and Wynn, H.: Data assimilation in the decision support system RODOS – Radiation Protection Dosimetry, 104, 31–40, 2003.

Wikipedia: Model View Controller pattern definition, http://en.wikipedia.org/wiki/Model-view-controller.

Wikipedia: Object Relational Mapping approach definition, http://en.wikipedia.org/wiki/Object-relational_mapping .

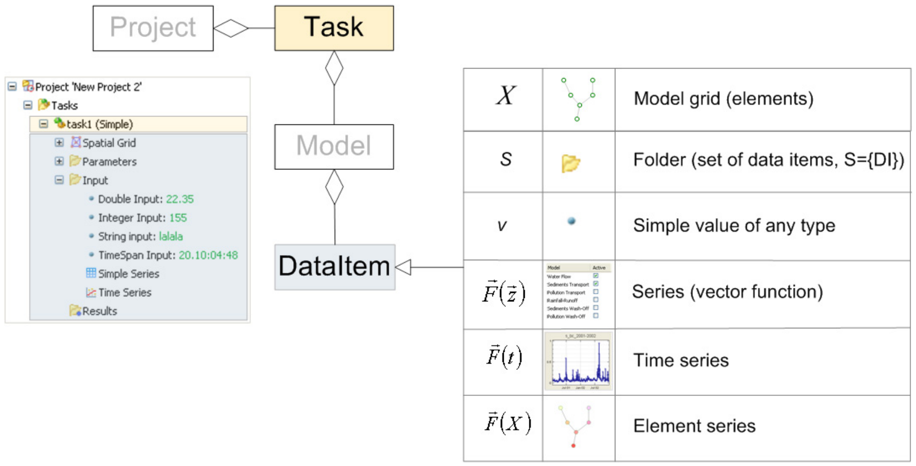
Wikipedia: Software Framework, http://en.wikipedia.org/wiki/Software_framework.



Project 'New Project 2'
Tasks
task1 (Simple)
Spatial Grid
Parameters
Input
Double Input: 22.35
Integer Input: 155
String input: lalala
TimeSpan Input: 20.10:04:48
Simple Series
Time Series
Results

Fig. 1. An architecture of the Modelling System.

Title Page
Abstract | Introduction
Conclusions | References
Tables | Figures
◀I | ▶I
◀ | ▶
Back | Close
Full Screen / Esc
Printer-friendly Version
Interactive Discussion



Project 'New Project 2'
- Tasks
  - task1 (Simple)
    - Spatial Grid
    - Parameters
    - Input
      - Double Input: 22.35
      - Integer Input: 155
      - String input: lalala
      - TimeSpan Input: 20.10:04:48
      - Simple Series
      - Time Series
    - Results

X	Model grid (elements)
S	Folder (set of data items, S={DI})
v	Simple value of any type
$\vec{F}(\vec{z})$	Series (vector function)
$\vec{F}(t)$	Time series
$\vec{F}(X)$	Element series

Fig. 2. Main data types.

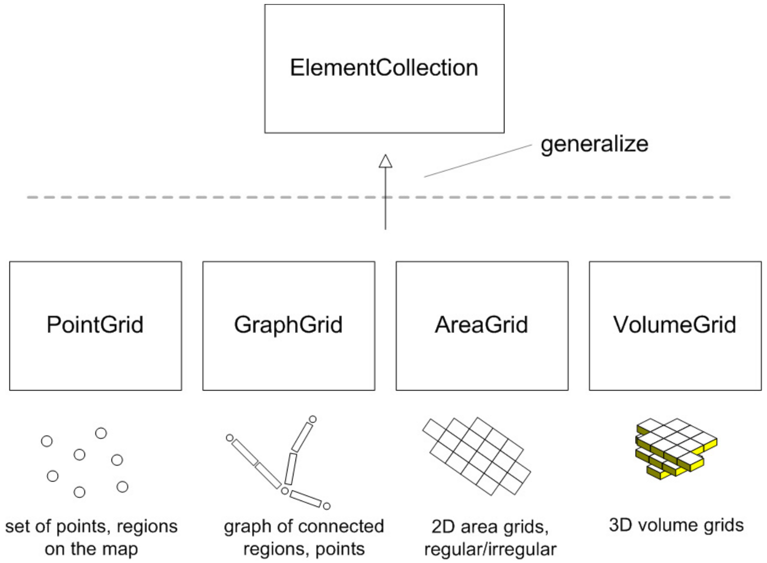

ElementCollection
generalize
PointGrid
GraphGrid
AreaGrid
VolumeGrid
set of points, regions on the map
graph of connected regions, points
2D area grids, regular/irregular
3D volume grids

Fig. 3. Examples of the ElementCollection implementations.

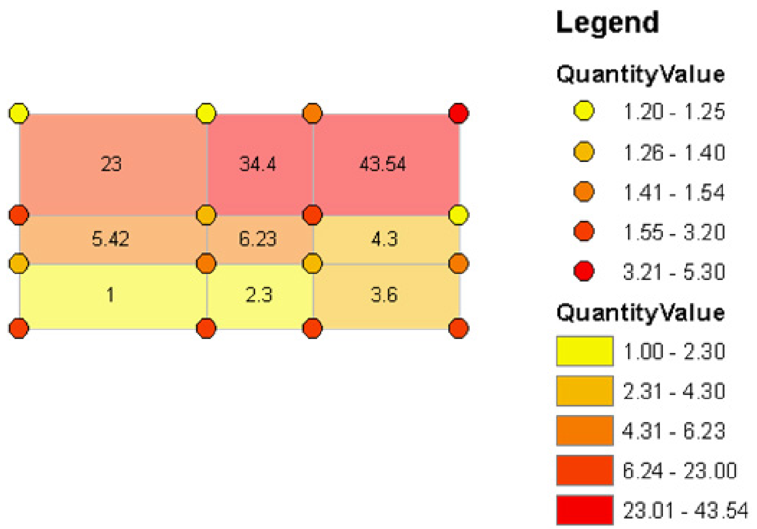


Fig. 4. Example of irregular rectangular grid, sets of vertices and cells; values of two quantities on these sets.

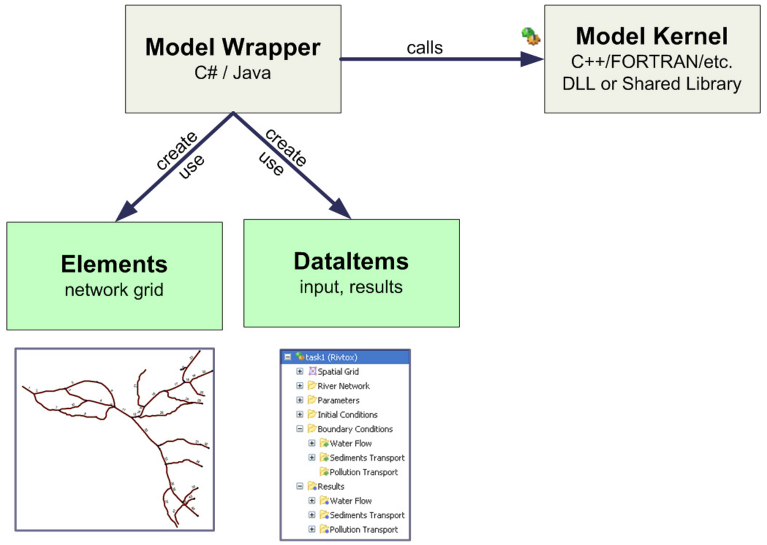
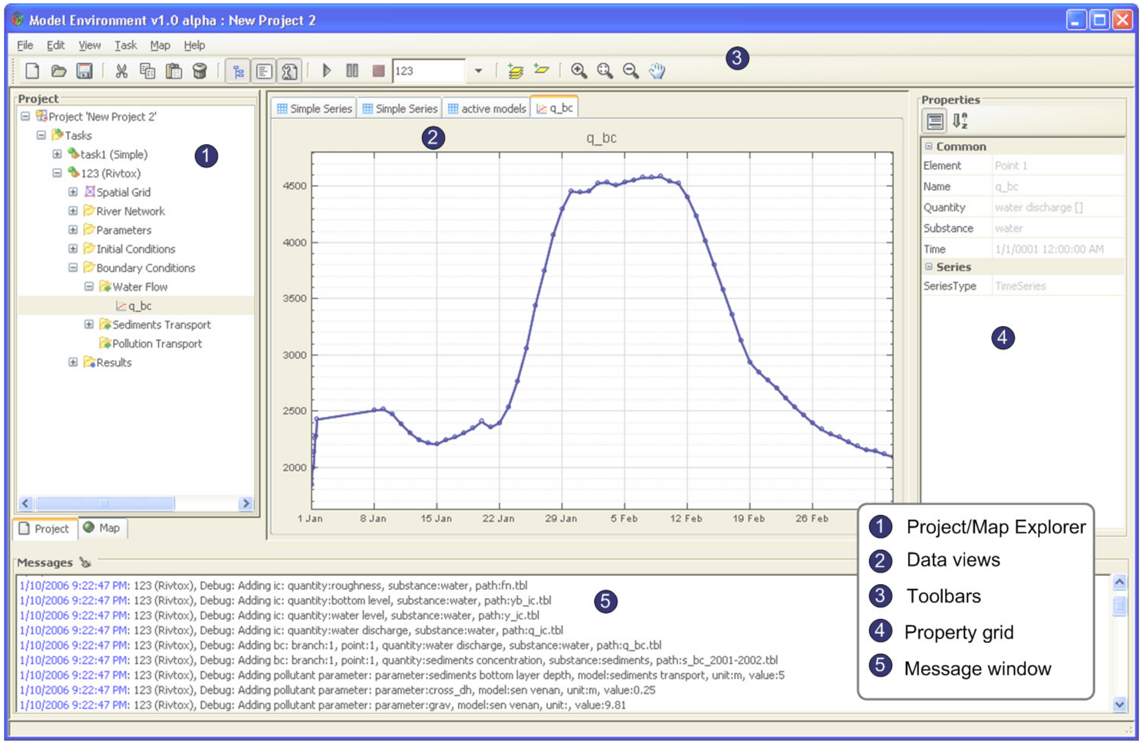


Fig. 5. Model wrappers and kernels interaction (integration of external models).



```
1/10/2006 9:22:47 PM: 123 (Rivtox), Debug: Adding ic: quantity:roughness, substance:water, path:fn.tbl
1/10/2006 9:22:47 PM: 123 (Rivtox), Debug: Adding ic: quantity:bottom level, substance:water, path:yb_ic.tbl
1/10/2006 9:22:47 PM: 123 (Rivtox), Debug: Adding ic: quantity:water level, substance:water, path:y_ic.tbl
1/10/2006 9:22:47 PM: 123 (Rivtox), Debug: Adding ic: quantity:water discharge, substance:water, path:q_ic.tbl
1/10/2006 9:22:47 PM: 123 (Rivtox), Debug: Adding bc: branch:1, point:1, quantity:water discharge, substance:water, path:q_bc.tbl
1/10/2006 9:22:47 PM: 123 (Rivtox), Debug: Adding bc: branch:1, point:1, quantity:sediments concentration, substance:sediments, path:s_bc_2001-2002.tbl
1/10/2006 9:22:47 PM: 123 (Rivtox), Debug: Adding pollutant parameter: parameter:sediments bottom layer depth, model:sediments transport, unit:m, value:5
1/10/2006 9:22:47 PM: 123 (Rivtox), Debug: Adding pollutant parameter: parameter:cross_dh, model:sen venan, unit:m, value:0.25
1/10/2006 9:22:47 PM: 123 (Rivtox), Debug: Adding pollutant parameter: parameter:grav, model:sen venan, unit:, value:9.81
```

Fig. 6. GUI design.