International Environmental Modelling and Software Society (iEMSs)
2010 International Congress on Environmental Modelling and Software
Modelling for Environment's Sake, Fifth Biennial Meeting, Ottawa, Canada
David A. Swayne, Wanhong Yang, A. A. Voinov, A. Rizzoli, T. Filatova (Eds.)
http://www.iemss.org/iemss2010/index.php?n=Main.Proceedings

DeltaShell - an open modelling environment

Gennadii Donchyts[1], **Bert Jagers**[2]
[1] *Deltares, Delft, The Netherlands* gennadii.donchyts@deltares.nl
[2] *Deltares, Delft, The Netherlands* bert.jagers@deltares.nl

Abstract: Over the last two years an open modelling environment has been developed by the Dutch research institute Deltares. Main goal of the development was to create a solid software platform for running 1D, 2D and 3D environmental models. The system allows creating, storing and visualizing different types of environmental data, like time series, networks, GIS features, and other data types defined in the spatio-temporal domain. The architecture of the system, due to its modular and open design, can also support a wide range of other applications such as GIS data management, management and analysis of the hydrological data. The present version of the system makes extensive use of the open-source libraries such as SharpMap, GeoAPI, NetCDF, NHibernate, SQLite, PostSharp; some of those libraries were significantly extended and will be made freely available to the community.

Keywords: integrated modelling; environmental modelling; software framework; domain-driven design.

1. INTRODUCTION

Integrated environmental modelling involving multiple models becomes more and more popular. Multiple efforts have been made trying to develop frameworks and standards allowing linking various components provided by the different vendors and allowing them to exchange data between each other (OpenMI, Gregersen [2007], ESMF, Collins [2005]). An integrated modelling environment is another area of growing interest Ames [2009], Donchyts [2008], Kralisch [2005], Rahman [2005]. However in most cases such frameworks mainly target the integration aspects and rarely consider modelling the whole application domain, including all aspects associated with it. We believe that in order to create a good integrated modelling system it is necessary to perform a detailed object-oriented analysis of all domains associated with the system. Such domains in our case are:

- Model specific domains, e.g. hydrology, hydraulics, water quality, morphology
- Specialized libraries required to describe all mathematical, physical and other aspects of the data types used by the models, for example units of measure, model variables defined in a discrete way, integration and interpolation, etc.
- GIS domain, containing such OGC standards as geometry, feature, coverage specifications
- Data storage using high-quality, scalable and fast technologies, this may include relational databases and/or multi-dimensional file formats such as NetCDF, HDF.
- Graphical user interface including all required components and all related usability issues.
- Modern computer science software development techniques such as domain-driven design, aspect-oriented programming, use of Inversion of Control / DI principles, continuous integration, test-driven development, etc.

After application of the object-oriented analysis Booch [1991] and domain-driven design Evans [2003] methods to the above listed application domains a new modelling framework

and modelling environment called DeltaShell was developed. A few examples of the graphical user interface of the DeltaShell are shown on the two figures below. Figure 2 shows integration of the 1D river flow model SOBEK being developed in Deltares. Figure 1 demonstrates OpenMI plug-in, which allows use of OpenMI-compliant components in the DeltaShell.

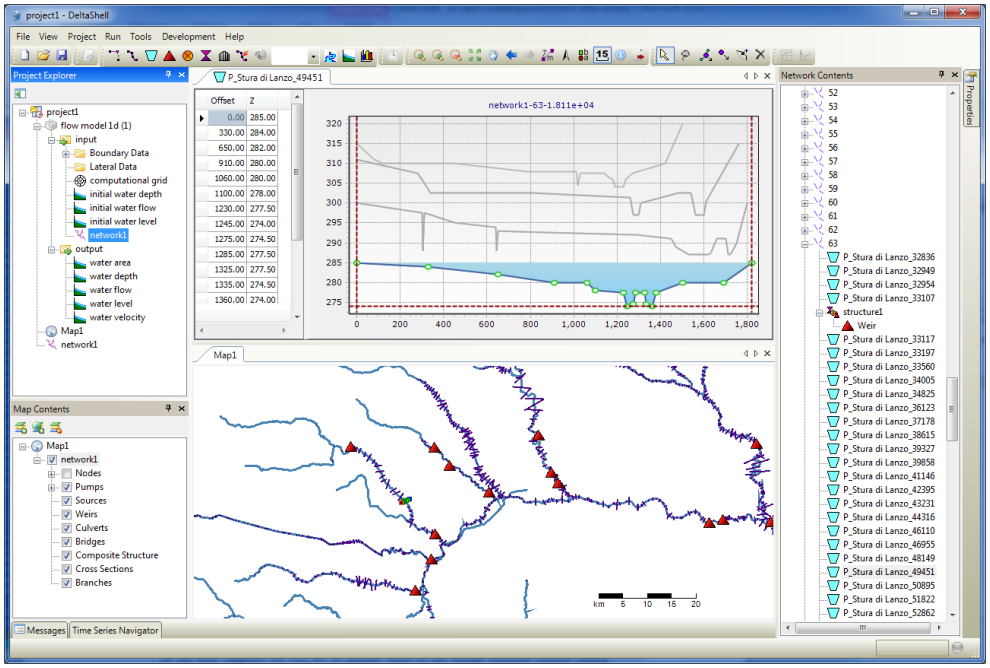

Figure 1 1D water flow model integrated into DeltaShell (Po River, Italy)

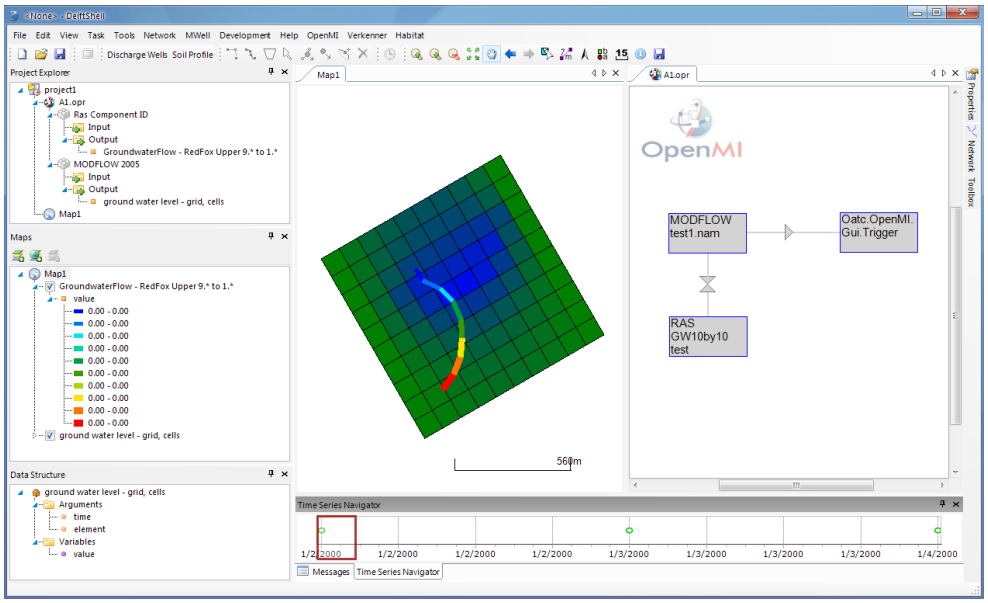

Figure 2 DeltaShell with a prototype of the OpenMI integration, hosting HEC-RAS and MODFLOW models

In the next chapters we will present some of the design solutions created during the development of the DeltaShell modelling environment. In the first chapter an overview of the DeltaShell architecture is presented. The following chapters focus on a number of more specialized topics such as possibilities to extend DeltaShell framework with plug-ins for new graphical user interface components, data types and models.

2. ARCHITECTURE

The architecture of the DeltaShell is presented on Figure 3. As can be seen from the figure, DeltaShell consists of a set of common class libraries, mainly representing different domains and software standards such as OGC-based geometries and features, hydrologic network objects, mathematical and physics libraries and general application framework class libraries (Core, Gui). These general framework class libraries define on a very high level how a typical console or graphical user interface application is constructed. They also define how different parts of the environment can be extended by means of plugins. This includes both non-gui plugins providing new data types or computational models to the system, but also graphical user interface components used to visualize different data.

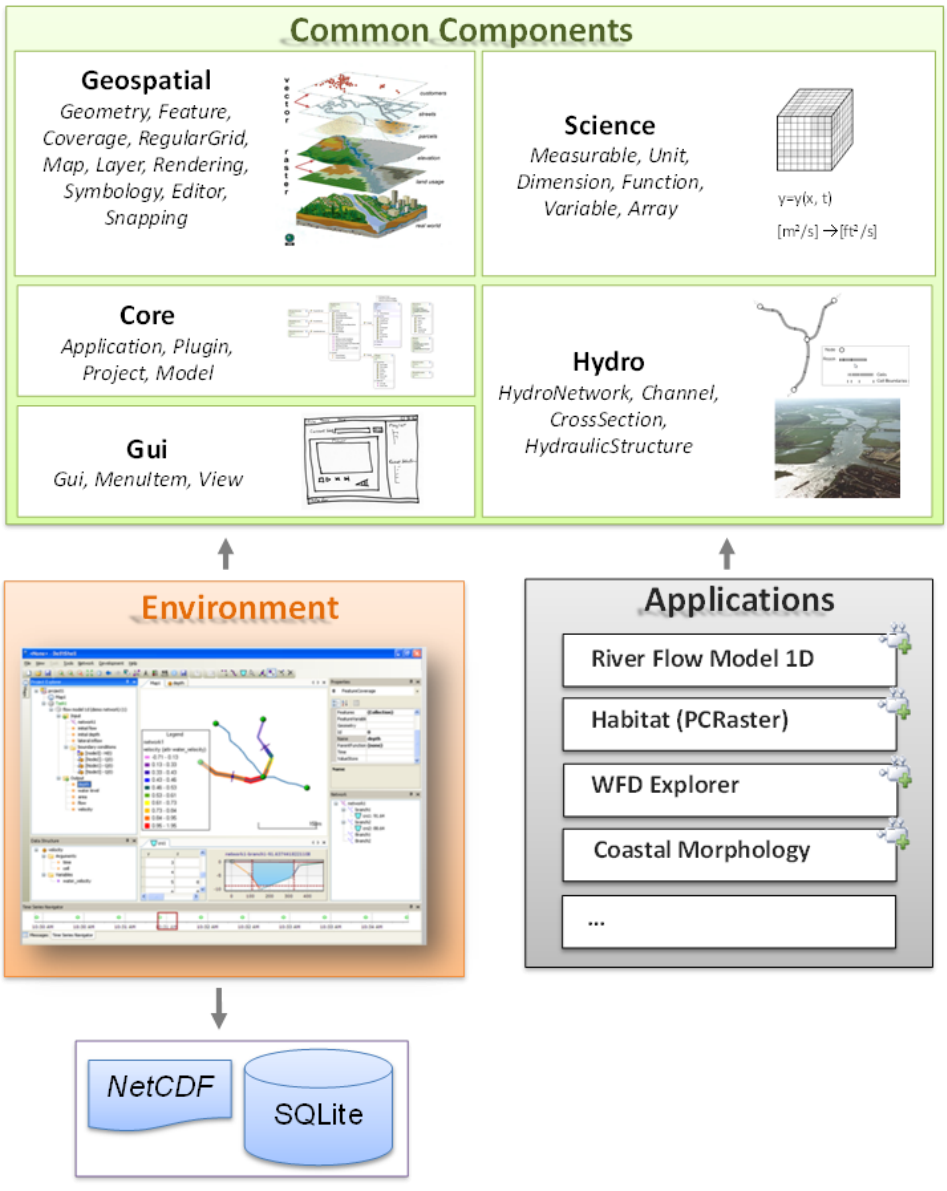

Figure 3 Overview of the DeltaShell architecture

On the next page the main components of the DeltaShell framework are presented (see Figure 4 and Figure 5). The red rectangles highlight parts which need to be implemented by the developer when developing a new plug-in. The following parts of the modelling framework can be extended by the third-party plug-ins:

- Data types

- 0D, 1D, 2D and 3D models of any type
- Data import and export components
- Graphical user interface components such as data editors, tool windows, menu items, toolbar items.

After a plug-in is implemented and deployed, all its custom menus and toolbars may be configured using a plug-in XML file.

DeltaShell uses a project class as single container of all data. The project is a working document of the user and is organized in a hierarchical way. A project may contain data items such as time series, raster grids, hydraulic networks, but also more general items such as models, maps, charts. Items provided by third-party plugins should also usually be stored in the project. All data items contained in the projects are by-default wrapped with a meta-class called DataItem (similar to ExchangeItem in OpenMI but much more generic). This is required in order to allow linking of the data used in the project but also to provide generic way to store data contained in the project.

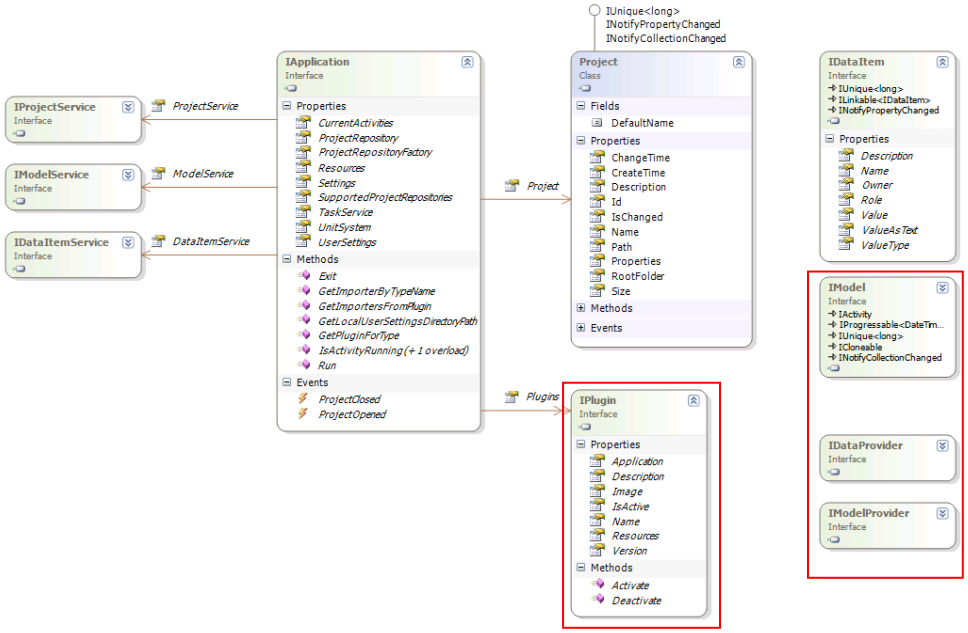

Figure 4 Core Framework UML class diagram

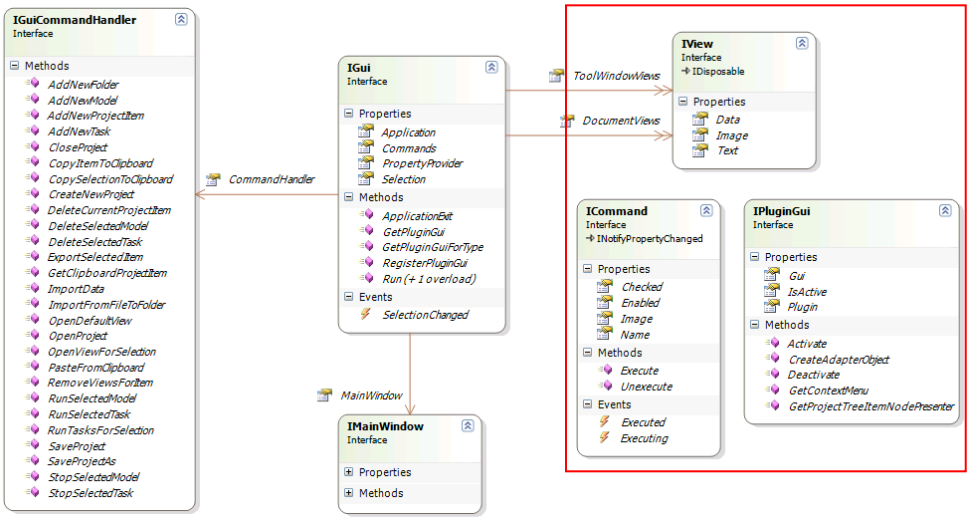

Figure 5 Gui Framework UML class diagram

3. CUSTOM DATA TYPES

In order to extend DeltaShell with a new data type it is required that developer at least implements an interface called IDataProvider. DeltaShell searches all implementations of that interface in all plug-ins during start-up and exposes all data types provided by them to the user. Data types exposed in such a way can be any custom entity defined by the user such as time series, rasters, tables, etc. DeltaShell also provides a set of default data types which can be shared between plug-ins. After data types become available in the system, they can be used in the project together with all other data types.

4. MODEL INTEGRATION

Very often models are implemented using programming languages such as FORTRAN, C++. DeltaShell allows integration of the external models on a lowest level, allowing exchange of data between model engine and modelling environment practically on every time step. The main advantage of this approach is that all model data can be edited, visualized and stored by the system within the system using generic components available as a part of the system or provided by developers of the plug-ins.

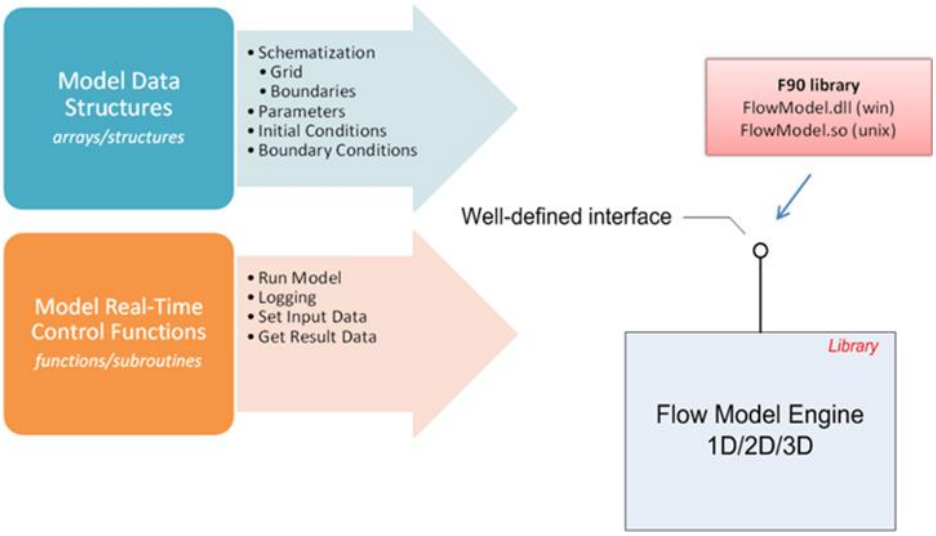

Figure 6 Application programming interface of the flow model engine

An example of a typical model integrated into the DeltaShell is shown on Figure 6. Principles to integrated models into DeltaShell are similar to those required by OpenMI software standard. So for those models which are already OpenMI-compliant an implementation of the DeltaShell plug-in would be a relatively easy procedure. The main difference in approaches between OpenMI and DeltaShell regarding model integration is that the latter requires that all input and output data items used by the model have to be exposed using either standard DeltaShell data structures or custom data structures (as described in the previous chapter).

5. SCIENTIFIC DATA MANAGEMENT

DeltaShell provides a class library used to define model data variables. The main interfaces used in the class library are shown on Figure 7. This class library is similar to the Common Data Model (CDM) developed by UCAR which allows defining and storing discrete variables defined using multi-dimensional data structures. However the present class library also models relations between different variables.

$$\vec{F} = (f_1, f_2 \ldots, f_n)(x_1, x_2 \ldots, x_m)$$

Component **variables**, multi-dimensional arrays — Argument **variables**, one-dimensional arrays

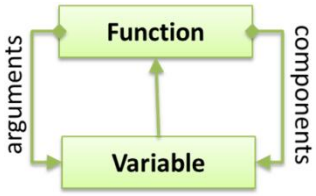

Figure 7 Functions and variables

This class library practically implements a vector function of a multiple arguments and multiple components. As can be seen on figure, a function is defined as an entity composed of its argument and component variables. At the same time a variable is defined as an entity derived from the function. This definition makes it look very similar to the mathematical definition of the vector function. This class library is used in many places of the system to define model variables used by the models. For example the following functions are used by a 1D water flow model:

- Time series such as: $Q = Q(t), y = y(t)$
- Rating curves: $Q = Q(H)$
- Model initial conditions: $Q=Q(s)$, where $s = (branch, offset)$ - location on a network
- Model results: $Q = Q(s, t)$

The next figure demonstrates how these classes can be used in order to define a two-dimensional function $f= f(x_1, x_2)$.

```
// construct f(x1, x2) and set values to 10.0 for x1 = 1.0 and x2 = 2.0
[Test]
public void SetValues()
{
    Variable<float> f = new Variable<float>("f");
    Variable<float> x1 = new Variable<float>("x1");
    Variable<float> x2 = new Variable<float>("x2");

    Function function = new Function("OneComponentTwoArguments Test");
    function.Components.Add(f);
    function.Arguments.Add(x1);
    function.Arguments.Add(x2);

    function.SetValues(new float[] { 10.0f },
        new VariableValueFilter(x1, 1.0f),
        new VariableValueFilter(x2, 2.0f)
    );
}
```

Figure 8 Defining 2d function using Functions class library

In order to store values of the functions an IFunctionStore interface is used. Currently we support the following implementations:

- MemoryFunctionStore
- NetCDFFunctionStore
- GdalFunctionStore

The first one simply keeps a set of multi-dimensional array in the memory for every argument and component variable of the function which is being used. The second one stores functions in the NetCDF files, wrapping UCAR Java implementation converted to .NET on a byte-code level. The third implementation is used to access raster data stored in GDAL file formats. Additionally to the simple data structures, the IFunction is also used as a basis for our custom implementation of coverages in the GIS class library. Currently the following coverages are supported (extending IFunction):

- ICoverage
- IRegularGridCoverage
- IDiscreteCurveCoverage
- IDiscreteGridPointCoverage

6. HYBRID DATA STORE

DeltaShell uses a mixed SQLite + NetCDF file format in order to store its project on the file system. Based on the type of data it decides where to store values of the objects used in the project. For example, most of the objects are stored in the relational database using NHibernate object-relational mapping library. But some of the data types (mainly those based on IFunction) are stored in NetCDF and only meta-information is stored in the relational database. In this case NetCDF file names always remain in sync with the corresponding records in the database. All storage logic is completely encapsulated in the data access plug-in, so a developer does not need to know anything about storage method. But he/she is required to provide a database mapping XML file once the plug-in introduces a new data type.

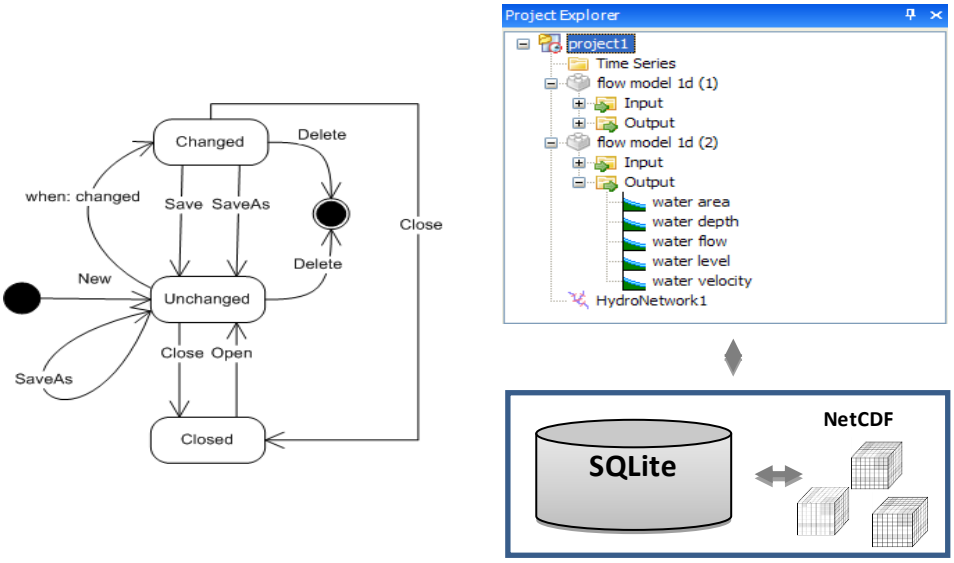

Figure 9 Project states and hybrid data store of the DeltaShell

7. GIS LIBRARIES

With increasing availability of detailed geospatial information, GIS becoming a crucial part of any integrated modelling system. In some cases it is used only as a data visualization tool, but it's increasingly also used to generate and store model schematizations used by numerical models, including all topological relations between different features of that schematization, for example ArcHydro, Maidment [2002]. However, ArcHydro is defined using ArcGIS as a platform which makes it hard to reuse directly as a part of an integrated modelling system.

The goal of the DeltaShell development in relation to modelling of the HydroNetwork was to provide a lightweight class library which is based only the open-source GIS class libraries and which can be reused by third-party developers working in object-oriented programming languages like C#/Java/C++.

DeltaShell is based on the following open-source GIS libraries:

- GeoAPI.NET – definition of OGC simple geometry specifications
- NetTopologySuite – implementation of OGC simple geometry specifications
- Proj.NET – coordinate system transformation library
- SharpMap – mapping library allowing to work with vector and raster layers

While these libraries were sufficient to implement a lightweight GIS subsystem, there were lacking some very important GIS concepts like Feature, Coverage, see OGC Abstract Specifications[1] for details.
As a part of DeltaShell development these libraries were significantly reworked and extended with the following features:

- IFeature – OGC simple feature specifications
- Coverages – OGC-like implementation, based in Function class discussed earlier
    - Regular grid coverages
    - Discrete curve coverages used to define data on 1D network
- Network library, used as a basis of the HydroNetwork
- Mapping functionality
    - Advanced map control supporting parallel layer rendering
    - Geometry and network editing functionality supporting snapping, topologies
    - Vector and raster layer symbology editors

8. CONCLUSIONS AND FINAL REMARKS

Main components and a few examples of the DeltaShell were presented. A very brief description of the major components was given. The goal of the authors was not to discuss every component of DeltaShell in detail but only to give a brief overview of the main functionality available there.
DeltaShell is currently used at Deltares as the basis for a new generation of graphical model user interfaces, and it is provided upon request as an open environment to others.

REFERENCES

Ames, D. P., Horsburgh, J., Goodall, J., Whiteaker, T., Tarboton, D., Maidment, D. Introducing the Open Source CUAHSI Hydrologic Information System Desktop Application (HIS Desktop). *18th World IMACS/MODSIM Congress, Australia* 2009.

Booch, G, Object-Oriented Design with Applications. *Addison-Wesley Professional*, 1991.

Collins, N., G. Theurich, C. DeLuca, M. Suarez, A. Trayanov, V. Balaji, P. Li, W. Yang, C. Hill, and A. da Silva, Design and Implementation of Components in the Earth System Modeling Framework. *International Journal of High Performance Computing Applications. Fall/Winter*, 2005.

Donchyts, G., Baart, F., Jagers, B., DelftShell - integrated modeling environment with elements of GIS, Data Management and OpenMI support, *American Geophysical Union*, 2008.

Evans, E. Domain-Driven Design: Tackling Complexity in the Heart of Software, *Addison-Wesley Professional*, 2003

Gregersen, J.B., Gijsbers, P.J.A. and Westen, S.J.P. OpenMI: Open modelling interface. *Journal ofHydroinformatics*, 9(3): 175-191, 2007.

Kralisch, S., P. Krause, O. David, Using the object modeling system for hydrological model development and application, *Advances in Geosciences*, 4, p. 75-81, 2005.

Maidment, D.R.,. Arc Hydro: GIS for Water Resources. *ESRI Press, Redlands, CA*, 2002.

Nativi, S., Caron, J., Domenico B. and Bigagli L., Unidata's Common Data Model mapping to the ISO 19123 Data Model, *Earth Science Informatics*, Volume 1, Number 2, 2008

Rahman, J. M., S. P. Perraud, H. Hotham, N. Murray, B. Leighton, A. Freebairn, G. Davis, and R. Bridgart. Evolution of TIME. In Zerger, A. and R. Argent, editors, *MODSIM 2005 International, Congress on Modelling and Simulation*, pages 697–703, 2005.

[1] The OpenGIS™ Abstract Specification, *http://www.opengeospatial.org/standards/as*.