Implementing Unified Access to Scientific Data from .NET Platform

Sergey B. Berezin
Assistant Professor,
Moscow State University,
Computational Mathematics and Cybernetics, Leninskie gory,
1 MSU, 119992, Moscow, Russia.
s_berezin@cs.msu.su

Dmitriy V. Voitsekhovskiy
Postgraduate,
Moscow State University,
Computational Mathematics and Cybernetics, Leninskie gory,
1 MSU, 119992, Moscow, Russia.
idmitry@inbox.ru

Vilen M. Paskonov
Professor,
Moscow State University,
Computational Mathematics and Cybernetics, Leninskie gory,
1 MSU, 119992, Moscow, Russia.
paskonov@cs.msu.su

ABSTRACT

Scientific data differ from common relational data in many aspects: scientific data may have a very complex structure, they are usually stored in files of various formats and individual data items can be very large. In this paper we present an extensible and efficient client-server system for accessing scientific data and its metadata. The architecture and major capabilities of our system will be described in the paper. The core of our approach is an extensible XML-based structure that annotates scientific data with rich metadata and maps every file or part of a file to a named strongly typed entity.

We do not introduce any new file formats and file transfer techniques, thus our approach doesn't require major changes to existing computational software. SOAP protocol and Web Services are used for accessing data sets and performing data requests. Filtering and caching enables an efficient access to large portions of data over network. Example of implemented filters are cropping and thinning of 2D and 3D arrays.

Our system is fully extensible and allows adding new data types, new file formats and new filtering algorithms without changing its core algorithms. Now it is used for accessing results of computational fluid dynamics simulations, but we hope that it can be adapted to many branches of science. The client is implemented on the .NET platform; the server-side is currently running on the IBM Regatta SMP mainframe on AIX

Keywords

Scientific data access, data management, visualization, web services, SOAP.

1. INTRODUCTION

Scientists are overwhelmed today by amounts of data generated by experiments and simulations. According to the Scientific Data Management Center at the Lawrence Berkeley National Laboratory [Sdm05w] up to 80 percents of a scientist's time is spent on data manipulation and only 20 percents – on actual analysis. That's why there is an emerging need of more convenient tools for scientific data access and analysis. Tendencies of scientific data management in near future are listed in [Jim05a] along with vision of the next generation data analysis tool called "smart notebook". In this paper we make a small step to such a tool by introducing our approach which consists of two parts: a scientific data access system and a data visualization tool.

.NET Technologies 2006
FULL papers conference proceedings
ISBN 80-86943-06-2

A lot of systems for scientific data management and analysis were developed for many branches of science, from astronomy [Jim01a] to computational fluid dynamics problems on irregular meshes [No01a]. Our system origins from the field of computational fluid dynamics but we believe that it appears to be useful in other branches of science.

Most important features of scientific data management systems can be found in the survey [Rea00a]. In our approach we focus on following aspects:

Logical data management – a data management system abstracts from the physical data layout. The resulting view of the data is a uniform collection of data items.

Physical data management – a request for logical data items results in a transparent physical files access, filtering and caching.

Metadata management – metadata describes data themselves [Jef02a]. Metadata is an important part of scientific data set, because it helps a scientist to understand data better and it helps various tools to perform a data analysis and visualization more efficiently.

2. RELATED WORKS

The evolution of Web technologies along with cheaper and more powerful hardware and increased networks bandwidth has brought to life new approaches to scientific data management. The huge number of online repositories and data centers allows scientists to publish, to search, to display and to download data.

The NCSA's Scientific Data Service (SDS) [Sds97w] provides Web access to a wide range of scientific data, facilitating data sharing between science teams and the general public. The SDS is a CGI program that provides scientific data in the several well known file formats. SDS is extensible and modular, but it is a fairly time consuming task to make SDS understand a new file format.

The metadata in SDS contains the fixed number of attributes to search by: spatial, temporal, dataset name, archive center, parameter name, platform name, sensor name, etc. Users can interactively examine the contents of a file with their Web browser, view a thumbnail image of the data, and retrieve the file, or a desired subset of the file, in its original file format or in ASCII. SDS has no object-oriented features and lacks support for client-side data management and caching.

The OpenGIS scientific data server [Ogs97w] is created by joint efforts of NCSA and the USDAC Consortium. It provides geospatial data according to object model described by the OpenGIS Abstract Specification. This model hides format details for three different types of geospatial data. Access to the scientific data objects is performed through the OpenGIS API. The objects returned to client can be visualized or saved as files. But object became a isolated entity after it has been obtained and it holds no reference to source data set.

The Distributed Oceanographic Data System (OPeNDAP) [Dap04w] is intended to give researchers a transparent access to oceanographic data across the Internet. Communication model in OPeNDAP works with URL addresses of web servers that deliver data to the researcher. In fact, researcher's data analysis software acts as a sophisticated web browser. Each data set is accessed via URL. Calls of API functions are forwarded to referenced web servers. Depending on the request type, the server returns a textual description of the data set contents or the actual values of data variables in a binary form. Textual descriptions provide a client library with metadata information concerning the operations that can be applied to data and the way binary data is to be decoded. The OPeNDAP incorporates a data translation facility, so that data may be stored in formats defined by the data provider, yet may be accessed by the user in a manner identical to the access of local files. Thus, the system provides transparent access to scientific data, but still there is no support for client-side data management.

Originality of our approach is based on following features: (1) integrity of data sets during their entire lifecycle, (2) efficient client-side data management and (3) common object-oriented API based on SOAP and XML. Another feature of proposed system is its high extensibility resulted from .NET Framework dynamic nature.

3. ABOUT DATASET

3.1. Common Features of Scientific Data

Long time passed since the single standard and SQL have been developed for the relational data model. However, scientific data strongly differ from common relational data in several aspects. This makes existing data management paradigms unsuitable for scientific data [Jim05a]. There is still no unified model for accessing scientific data. In this paper we introduce a new approach to the scientific data access that seems to be pretty general.

Our logical data model was designed to reflect following common features of all scientific data:

- Scientific data may have a very complex structure and are usually stored in files of various specific formats; individual data items can be very large.
- Scientific data often depend on parameters (for example, on time) or can be viewed as a collection of parameter *slices*.
- Practically all results of scientific researches contain both data and metadata.

Metadata can be of two types. The first type of metadata is designated for human reading and contains information about simulation parameters, about authors and the origin of data and so on. This type of metadata allows associative search, categorization and better understanding of scientific data by external researchers.

The second type of metadata describes the type and the format of scientific data. It is most useful for different automated tools for data retrieval, filtering and analysis. For example, the information about the type helps the visualization system to suggest the most suitable visualization method and its parameters.

3.2. DataSet Object Model

DataSet is a key notion of our approach to the scientific data management. It can be thought as a self-describing entity containing references to actual data annotated with rich metadata. DataSet Object Model is shown on Fig.1.

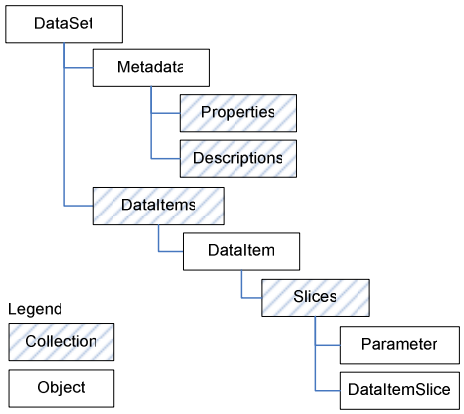

Figure. 1. DataSet Object Model.

The Metadata section holds descriptive information about DataSet and its data in a human readable form. The Properties collection contains information about simulations parameters, units of measurements, affiliations and authors of data. The Descriptions collection contains descriptions and annotations of any object in a DataSet. These collections can be used for searching and arranging DataSets.

The Metadata section also contains an address of a data source server (to which a data request should be sent) and the origin of a DataSet. The former allows copying and distributing the DataSet, keeping its functionality, and the latter allows checking for possible updates to the DataSet.

Every logical part of data set is represented by DataItem, which maps a portion of real data to a named strongly typed object. A DataItem can depend on one or more named parameters. Thus, a DataItem is a collection of so-called slices, which correspond to data for specific parameters values. The value of a DataItem for specified parameters is represented by an individual DataItemSlice object.

Each parameter has a name and a strongly defined type such as double, string etc. The example of parameters in computation fluid dynamics is time or the Reynolds number, in geophysics – coordinates of a data capture.

A DataItem can be either simple or composite. Simple DataItems hold references to a data piece that can be retrieved from a single location. We do not introduce new file formats, but instead we rely on existing well known formats such as netCDF or HDF [Fmt06w]. The usage of existing file formats has following advantages:

- We can easily assembly existing data in DataSets;
- We can use existing libraries to write or read DataItems of DataSets;
- We can extract parts of DataSet for processing with existing tools and utilities.

Composite DataItems are built by on the basis of one or more components (see Fig.2). Each component is the pair of a DataItem (possibly also composite) and an optional class name of the component. Class names help to distinguish components.

The following example introduces a constructor for computation fluid dynamics problems. Let's assume that the DataSet contains two DataItems: `uvw-values`, as a three-dimensional array of vectors, and `channel`, as a spatial grid. Combination of 3D vector array and data grid is a vector field. The constructor named `DataField` is used to create a composite DataItem `velocity`, representing the vector field. In such a way, the composite DataItem should be declared in the DataSet as an output of the `DataField` constructor depending on two components: the `uvw-values` with the class "values" and the `channel` with the class "grid" (see listing 1).

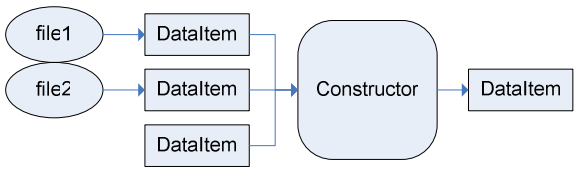

Figure 2. Composite DataItem construction.

Another example of important constructors is the constructor named `CompositeVectorArray` that allows creating new arrays by combining several arrays with smaller dimensions of items (see example listing 1), and vice versa.

A DataSet aggregates different data sources transparently for applications and makes it possible to view scientific data as a single collection of typed objects. This allows both logical and physical data independence.

The common standard for XML metadata descriptors and the DataSet XML schema definition were developed and now they are used in data repository for simulations in the field of CFD. We believe that this structure will be suitable for many fields of science, from computational fluid dynamics to biological systems modeling.

3.3. DataSet Example

The following DataSet XML document describes results of the numerical modeling of an unsteady flow

of viscous incompressible fluid in a flat channel. The results of the modeling consist of four files with a scalar array (three for the fluid velocity components and one for the pressure) for each moment of time.

```
<dataset id="…" dataSource="http://..."
   origin="http://..." type="CFD" …>
  <metadata>
    <property name="Re"
        description="Reynolds number"
        type="double" value="140.0" />
    …
    <description>Incompressible viscous flow
in a 3D channel</description>
    <description id="velocity">Velocity
vector field</description>
  </metadata>
  <structure>
    <dataItem id="uvw-values"
              type="Vector3dArray3d" >
     <composite
        constructor="CompositeVectorArray">
      <component id="u-values" />
      <component id="v-values" />
      <component id="w-values" />
     </composite>
    </dataItem>
    <dataItem id="velocity"
              type="VectorField3d" >
     <composite
        constructor="DataField">
      <component id="uvw-values"
           class="values"/>
      <component id="channel" class="grid"/>
     </composite>
    </dataItem>
    <dataItemTemplate id="u-values"
      type="ScalarArray2d"
      sourceType="netCDF" />
  </structure>
  <data>
    <dataItem id="channel"
     type="NonUniformGrid3d"
     sourceName="grid.dat"
     sourceType="plain text" />
    <parameter name="time" type="double">
      <slice value="0.00000">
        <dataItem id="u-values"
             sourceName="u_0000.cdf" />
        …
      </slice>
      …
    </parameter>
  </data>
</dataset>
```

Listing 1. Example of DataSet XML document.

The `structure` section specifies composite DataItems and templates for simple DataItems. The `dataItemTemplate` element is used to simplify DataItems declarations, especially parameterized. If the template is defined for certain `id` then attributes of the DataItem with the same `id` in `data` section will be considered as defined by default and may be either omitted or redefined with new values.

In the `data` section there are simple DataItems defined and arranged in slices by parameters values. In our example DataItems are defined for every moment of time and correspond to each component of the velocity vector and pressure. The DataItem `channel` represents the mentioned above spatial grid, that does not depend on time.

4. IMPLEMENTATION DETAILS

4.1. Architecture Overview

The server-side is currently running on the SMP mainframe IBM Regatta on AIX and implements two Web services. The first Web service performs administrative functions and provides access to DataSets. Search by metadata values is possible. The second Web service serves requests for DataItems and performs filtering. Complementary data request caching is used to maximize the speed of the service. The client-side of the system is implemented on the .NET platform as class libraries. Global view of our system is shown on Fig. 3.

The central class of the libraries is a DataSet. It is developed according to the DataSet object model (see Fig.1) and can be constructed on the basis of an XML document that represents DataSet entity. The DataSet class contains metadata and a collection of named objects of the DataItem class.

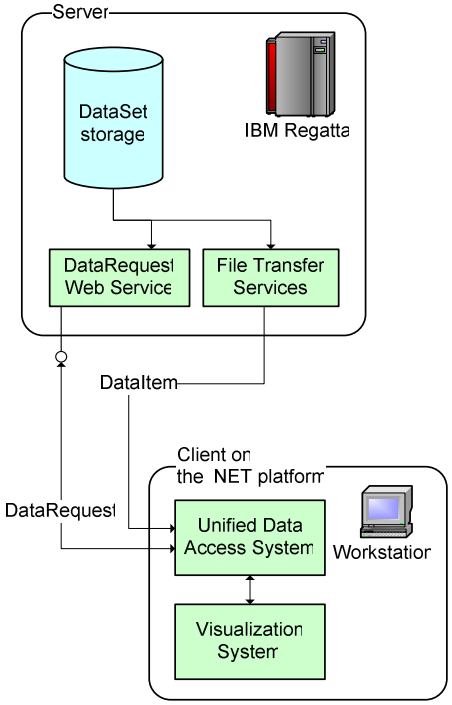

Figure 3. Architecture overview.

The structure of the DataItem class is represented by a tree, the nodes of which correspond to parameters and leaves – to DataItemSlice objects. The DataItem class offers convenient methods for data indexing by a set of parameters, returning an object of the DataItemSlice class for specified parameters values.

The DataItemSlice methods provide a direct access to data and return typed data object.

When data object is requested the system automatically forms and sends a data request to a data source address that is declared in metadata of the DataSet. Thus, client applications work with data transparently and without caring where and how they are stored.

The following code accesses array of velocity vectors at the moment 0.0 from the DataSet described in section 2.3. It can be seen in the DataSet that the required array is located on a server as three different files, but the physical representation of the data does not matter for the client at all.

```
// Creating object of class DataSet using
// XML-representation of DataSet
DataSet dataset = new DataSet(xmlDoc);
// Fetching DataItem by its name
DataItem velocity =
     dataset.DataItems["uvw-values"];
// Creating parameter corresponding to time
CompositeParameter param =
     new CompositeParameter(
     new ParameterValue("time", 0.0d) );
// Fetching DataItemSlice for the parameter.
// It is an instance of DataItem for
// specified parameter value.
DataItemSlice dataVelocity =
     velocity[param];
// Getting required data
Vector3dArray2d data =
  dataVelocity.GetData() as Vector3dArray2d;
```

Listing 2. Getting required data in C#.

4.2. Data Filtering

In most cases an application may request filtering of the data, i.e. their additional processing. For instance, a visualization program does not need such detailed grid data as they are usually computed in numerical experiments. Therefore thinning filter will be useful in this situation, because the resulting data after filtering will have exactly as many points as necessary for its correct visualization.

Another example of filtering is cropping. Let us assume that a scientist wants to study part of the data in detail. There is no need for a full local copy of existing data in that case, therefore cropping filter will return only required data.

Data filtering can be performed either by the client-side of the system or by the server-side. It occurs absolutely transparently for applications that work with the system: the decision where filtering will take place is taken by the system itself.

Thus, besides specific data handling for specific problem field, filters increase the efficiency of the system and reduce network traffic.

The following example expands the previous one given in listing 2 and illustrates how an application may request data with additional filtering. If there is no need for such a detailed velocity vectors array as it stored in files, a thinning filter may be applied to the data. The "Thinner" filter has parameters PercentageX, PercentageY and PercentageZ – those are fractions of points for each axis, which shall remain after filtering, and we make them equal to 5%. Code in C# is shown below:

```
// Creating object of class DataSet using
// XML-representation of DataSet
DataSet dataset = new DataSet(xmlDoc);
// Fetching DataItem by its name
DataItem velocity =
     dataset.DataItems["uvw-values"];
// Creating parameter corresponding to time
CompositeParameter param =
     new CompositeParameter(
     new ParameterValue("time", 0.0d) );
// Fetching DataItemSlice for the parameter.
// It is an instance of DataItem for
// the specified parameter value
DataItemSlice dataVelocity =
     velocity[param];

// Creating filter "Thinner" for required
// data type and setting up its parameters
Filter filter = FilterFactory.GetFilter(
        "Thinner", // filter class name
        dataVelocity.TypeDescriptor);
FilterServices.SetFilterParameters(filter,
   new FilterParameter[] {
   new FilterParameter("PercentageX", 0.05),
   new FilterParameter("PercentageY", 0.05),
   new FilterParameter("PercentageZ", 0.05)
        } );

// Getting required data
Vector3dArray2d data =
     dataVelocity.GetData(filter)
     as Vector3dArray2d;
```

Listing 3. Getting filtered data in C#.

Here an application gets the required filter, requesting it from the `FilterFactory` object by the filter's class name and the type of data, to which it shall be applied. Use of class factories is one of the keys which enable the system's high extensibility.

4.3. Performing DataRequest

DataRequest contains DataItem reference and filters, which shall be applied to this DataItem. DataRequest's content provides all information required to load the data. Any DataItem reference belongs to one of the three types. The first type, named dataSource, is designed for server's handling, which can locally (for the server) load requested data according to the reference. The second type, named dataRef, is used for remote loading of data that already are available on server as one file or directory. Besides the data type, dataRef contains the transfer protocol type and URL. The third type of a DataItem assumes that data are stored inline in

DataRequest. It is designated for transferring small pieces of data and improves the overall efficiency of request processing.

When an application requests data, the client libraries form DataRequest for specified DataItem from DataSet. DataRequest is passed by SOAP protocol to DRS Web service (see Fig.3), which loads requested data and tries to apply specified filters.

Only part of the filters might be applied, because some of filters may be either absent on server or inapplicable for particular data types. After filtering is completed, the server makes filtered data shared for the client, removes applied filters from DataRequest, and replaces all dataSource elements with dataRefs referring to the data or with inline data (see Fig.4).

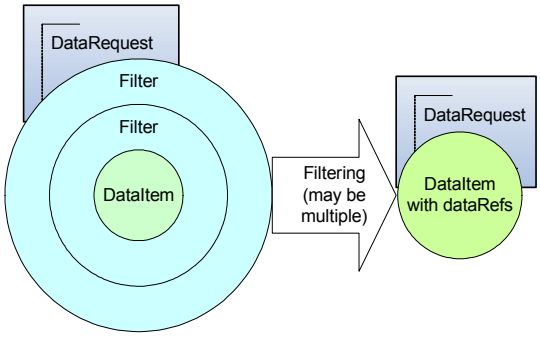

Figure 4. DataRequest lifetime.

The final DataRequest is sent back to the client. In this stage it contains either inline data or dataRefs, i.e. what and how a client must load, and a list of unapplied filters (which may be empty). As the data are downloading to a local computer, remote dataRefs become local references. After that, the client-side of the system parses the data and applies the filters which have been failed at the server. Data parsing is performed by special data source objects. The system can use various data source objects for each pair of a data source type and a type of data, which shall be loaded. All information that is necessary for a data loader is contained in dataRef.

We neither introduce a new file transfer technique nor restrict the choice of the existing one. The data transfer type is specified in dataRef by the server depending on its capabilities or any other term (for example, a security policy). Currently this is a transfer by FTP that is used, i.e. the server returns an address of FTP endpoint and a path to the needed file. One more file transfer possibility is the usage of WS-Attachments extension. This option is simple and interoperable, but it requires an extra bandwidth and may not be applicable due to a security policy on some systems.

In all suitable cases both the server and the client make caching of the request's result to decrease request handling time. Data request processing diagram is shown below.

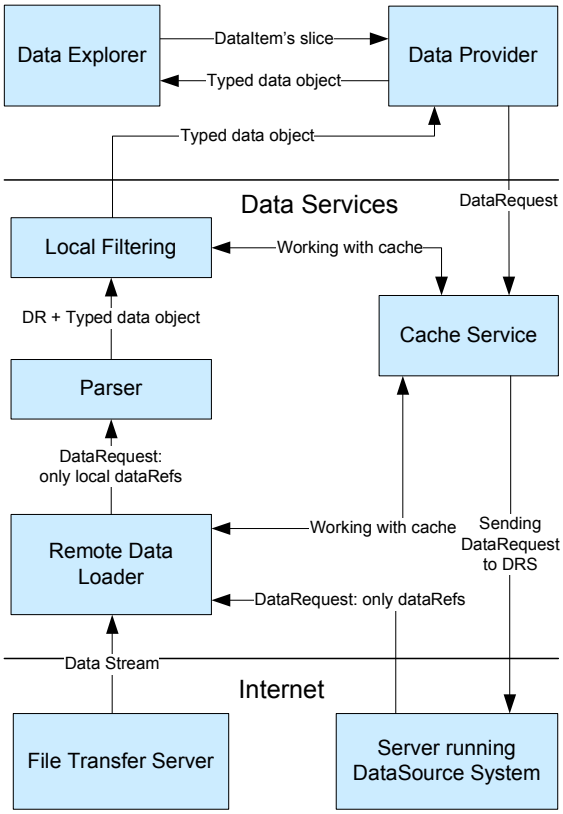

Figure 5. DataRequest performing schema.

Contents of DataRequests, which are generated by code on listing 3, are shown on listing 4:

```
<soap:Envelope … >
 <soap:Body>
  <dataRequest
        dataSource="…" dataSet="guid" … >
   <filter name="Thinner">
    <parameters> … </parameters>
    <dataItem type="Vector3dArray3d">
     <composite
        constructor="CompositeVectorArray">
      <component> <!-- u-values -->
       <dataItem type="ScalarArray3d">
        <dataSource sourceName="u0000.cdf"
sourceType="netCDF" sourceParameters="u" />
       </dataItem>
      </component>
      <component> … </component> <!-- v -->
      <component> … </component> <!-- w -->
     </composite>
    </dataItem>
   </filter>
  </dataRequest>
 </soap:Body>
</soap:Envelope>
```

Listing 4. DataRequest that is sent to the server.

```
<soap:Envelope …>
 <soap:Body>
  <dataRequest
        dataSource="…" dataSet="guid" …>
```

```
    <dataItem type="Vector3dArray2d">
      <dataRef sourceType="binary"
        sourceParameters="">
       <ftp url="ftp://..." />
      </dataRef>
    </dataItem>
  </dataRequest>
 </soap:Body>
</soap:Envelope>
```

Listing 5. DataRequest received from server

4.4. Extensibility of the System

One of our goals is not to design system for handling CFD-related data, but to create extensible and adaptable framework for managing scientific data sets. Our system can be extended by new data types, new data sources and new filters.

New data type is just a CLR class with no other requirements. Additional interfaces such as IScalarArray2d or INonUniformGrid3d whose names speak for them are implemented when needed. For each data type special type descriptor can be defined in configuration file.

Data sources are used for loading data objects from files or for composing new data objects from existing ones (example is constructing vector array from few scalar arrays). Thus, new data source has to be developed for each new file format or for new composite data type. Data sources are also listed in configuration file.

Data filters transform data objects according to filter's parameters. New data filters should implement two main functionalities: filter should be able to embed itself in XML data request for server processing and be able to perform actual client side filtering if it is not supported on server. Filters are also defined in configuration file.

For each new type of objects CLR class name and strong assembly name is specified in configuration file. On system start-up configuration file is examined. Assemblies are loaded on demand and objects are tied together in runtime using reflection and dynamic type information.

The system's architecture also allows every module having special code optimizations. For example, a filter can be optimized for work with a certain data type (from any module) and vice versa.

5. VISUALIZATION

Atop the data access system described above we build a visualization system for graphical exploration and analysis of data. A sample screenshot is shown below.

Our visualization system is built around the concept of workspace – a combination of DataSets and DataViews. It is important to mention that Workspace contains only references to DataSets, so Workspace is a very compact data structure that can be easily transferred from the researcher's workstation to his or her notebook providing a familiar work environment at any location. The structure of DataSet is shown in the left window on the screenshot. There you can see a list of DataItems and their parameters.

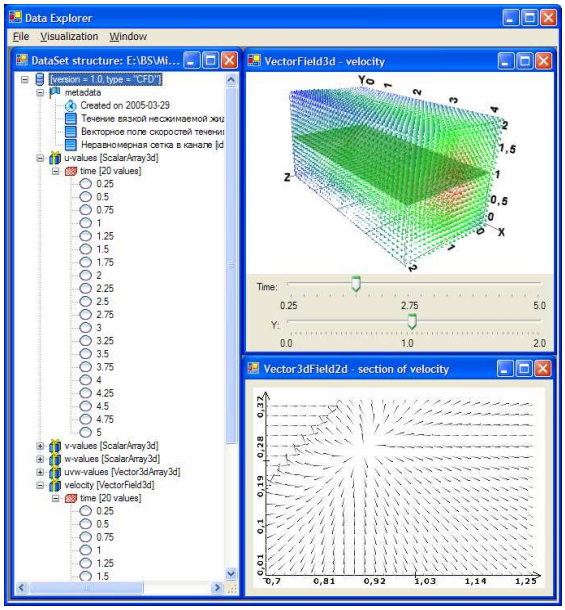

Figure 6. Visualization system screenshot

DataView is a visual object formed by a pair of a data object and a visualization algorithm. On the screenshot you can see one primary DataView (in the right-top window) and one dependent DataView (in the right-bottom window).

The primary DataViews take one of DataItems as its data object. The visualization algorithm can be chosen by the user from a list of options that is formed according to a DataItem type. Options could be sorted additionally according to the problem description found in metadata (i.e. physical oriented visualization algorithms will be on top for CFD problems).

If a DataItem depends on some parameters, the user is given a choice either to create a DataView for individual parameter slices or to display the entire DataItem with extra dimensions added by parameters. For example, a scalar 2D data field in coordinates (u,v) dependent on time can be displayed as an animation of a 2D surface in time or as a 3D scalar field in coordinated (u,v,t).

The secondary DataViews is created by applying one of visualization tools to an existing DataView, primary or secondary. The visualization tool is an object that can be applied to a specified type of DataView, has its own visual representation and results in a new data object. The green plane in the

right-top window is a section tool that extracts 2D subset from a 3D vector. The section plane can be moved up or down using control below. On the screenshot values of the 2D vector field subset are shown as a 2D marker field.

So, the Workspace can be thought as hierarchy of DataViews with DataSets as roots. Interacting with controls changes data in the DataView and this change is propagated automatically to all dependent DataViews. Although the data flow paradigm is not new in the field of scientific visualization [Vis96a] our visualization system allows graphical constructions of new visualization tools instantly from a visual representation of data. We believe that this will enable scientists to get new insights into data on the fly.

6. SUMMARY

The scientific data access system presented in this paper has following advantages:

- It gives an object oriented view to scientific data, which means that the client can retrieve metadata and data as strongly typed objects with caching and filtering.
- It allows creating a single family of data analysis tools, because almost any set of scientific data can be represented as a DataSet.
- It provides an indexing and associative search of data by their attributes and parameters hiding their physical location.
- It is highly extensible and provides interfaces for adding new data types, new types of data storage and new filters. This makes our system applicable to almost every branch of science.
- It is designed to interact with existing data storage formats and there is no need to abandon the existing computational or simulation software.

7. FUTURE WORK

We plan to extent our approach in three ways:

- Implement data management abilities – currently our system is a data access system with no ability to modify DataItems or DataSets.
- Extend a set of visualization tools by extending our visualization software and providing interfaces for our data access system from the existing powerful visualization software such as AVS
- Implement in-memory cache on client computer. Weak references are not suitable for this task because when amount of data exceed hundreds of megabytes weak reference became invalid shortly despite that there are still a lot of free memory.
- Implement server-side software on .NET Platform with reusing significant part of client-side code for data filtering and parsing.
- Design and implement second version API using language integrated queries and features found in LINQ [Lnq06w].

8. ACKNOWLEGEMENTS

This project was supported by RFBR grant 05-07-90378, 04-01-00332 and by the Student Laboratory of Microsoft Technologies at the Moscow State University.

9. REFERENCES

[Vis96a] William J. Schroeder, Kenneth M. Martin, William E. Lorensen. The Design and Implementation of an Object-Oriented Toolkit for 3D Graphics and Visualization. IEEE Visualization '96

[Sdm05w] Scientific Data Management Center at Lawrence Berkeley National Laboratory at http://sdm.lbl.gov/sdmcenter

[Jim05a] Jim Gray; David T. Liu; Maria A. Nieto-Santisteban; Alexander S. Szalay; Gerd Heber; David DeWitt. Scientific Data Management in the Coming Decade. Microsoft Research Technical Report MSR-TR-2005-10

[Jim01a] Jim Gray; Alexander Szalay; Ani Thakar; Peter Z. Zunszt; Tanu Malik; Jordan Raddick.Christopher Stoughton; Jan van den Berg. The SDSS SkyServer - Public Access to the Sloan Digital Sky Server Data. Microsoft Research Technical Report MSR-TR-2001-104

[No01a] J. No, R. Thakur, D. Kaushik, L. Freitag, and A. Choudhary. "A Scientific Data Management System for Irregular Applications", in *Proc. of the Eighth International Workshop on Solving Irregular Problems in Parallel (Irregular 2001)*, April 2001

[Rea00a] Reagan Moore. Data Management Systems for Scientific Applications. IFIP Conference Proceedings; Vol. 188. pp. 273 – 284, 2000.

[Jef02a] K. G. Jeffery, A. Asserson, A. S. Lopatenko, Comparative Study of Metadata for Scientific Information: The place of CERIF in CRISs and Scientific Repositories. Gaining Insight from Research Information, 6th International Conference on Current Reseach Information Systems, August 29-31, 2002 in Kassel, German

[Sds97w] NCSA Scientific Data Server http://hdf.ncsa.uiuc.edu/horizon/DataServer/sds_design.html

[Ogs97w] Open Geospatial Consortium http://www.opengeospatial.org/

[Dap04w] OPeNDAP: Open Source Project for a Network Data Access Protocol. http://www.opendap.org

[Fmt06w] Scientific Data Format Information FAQ http://fits.cv.nrao.edu/traffic/scidataformats/faq.html

[Lnq06w] The LINQ Project. http://msdn.microsoft.com/netframework/future/linq/