Efficient transfer and object-relational mapping of data for client-server applications through the Internet

P. Voborník

***Abstract*—**This article summarizes current approaches for the transmission of the data for the client-server applications which communicate via the public internet. On their basis an entirely new approach is proposed that ensures to minimize server load and maximize user comfort when working with a client application. This method encapsulates all operations into two class libraries, one of which operates on the server side and the second is integrated into the client application. Data are submitted to the client application in a clear object-oriented form, including references between them, only minimized and uncomplicated text strings are sent via the internet and these data are automatically stored on the server in a classic relational database. In addition to the data transfer this method also solves object-relational mapping. Using the method presented in the article also brings comfort for applications developers and they can thus concentrate more on the development of application logic.

***Keywords*—**Data interchange, client-server, serialization, Silverlight, web services, object-relational mapping, DataSet, OOP, object database, WCF.

I. Introduction

THE administration interface of Universal Testing Environment[1] provides the administrators with a user-friendly way of the central database data editing on the remote server. This communication takes place over the public Internet network between a client application in Silverlight[2] and a server application in ASP.NET (see Fig. 1). Data are stored in a classic relational Firebird[3] database on the server. The client application then works with the data as objects.

The requirement aroused for a simple way of passing data between a web server and a client application. Ideal would be to unify this procedure to cover all the needs not only of this application, but also so that it would be applicable universally in other projects.

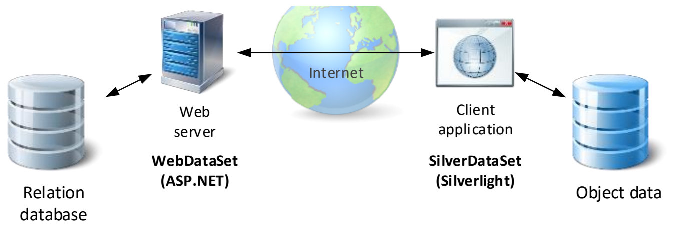

Fig. 1 – Scheme of data transfer between DataSet on the server and DataSet on the client side

The ideal candidate for a such two-way exchange of relational data would be the **DataSet**. *The DataSet object in .NET represents an in-memory cache of data and is the critical object that gets passed between the middle-tier and the Web Services layer. DataSet objects easily and efficiently serialize themselves into and out of XML. This means that data, as well as the related schema information, can be moved between tiers in a loosely coupled manner.* [1, p. 212]

However, the DataSet class is not supported in Silverlight, because of the minimize the installation package of Silverlight version of the .NET Framework, and its addition to Silverlight is not planned[4]. For this reason, independent project was created, which tried to create a DataSet for Silverlight. This project should encompass all of its functionality and enabled a data exchange with the Web server by using serialized data compatible with the original .NET DataSet[5]. This approach was based on a translation of serialized data to a dynamically created objects directly by techniques MSIL[6]. Although these objects fully cooperated with standard data components, it was very difficult to supplement and work with them, because their dynamically created classes did not exist at the time of application design. This project the authors have not completed

This research has been supported by Specific research project of University of Hradec Kralove, Faculty of Science in 2015.

P. Voborník is with the Department of Informatics, Faculty of Science, University of Hradec Králové, Rokitanského 62, Hradec Králové, 500 03, Czech Republic (e-mail: petr.vobornik@uhk.cz, orcid: 0000-0003-1841-3455).

[1] Universal Testing Environment is an electronic online testing system designed for the creation, operation and administration of the tests, independently or in cooperation with LMS [20].

[2] *Silverlight is a software plugin for development lavishly furnished internet applications that run within a web browser. It is developed by Microsoft, executed using the plugin which is a smaller version of the .NET framework and written in various languages supported by .NET (e.g. C# or Visual Basic).* [21]

[3] *Firebird is a relational database offering many ANSI SQL standard features, offers excellent concurrency, high performance, and powerful language support for stored procedures and triggers.* [22, p. 97]

[4] MSDN Blogs – ADO.NET team blog – DataSet and Silverlight: http://blogs.msdn.com/b/adonet/archive/2009/05/26/dataset-and-silverlight.aspx

[5] ADO.Net DataSet for Silverlight Applications, see www.silverlightdataset.net

[6] MSIL – Microsoft Intermediate Language, simply IL – the low-level assembly language with simple syntax based on numeric codes to which are translated (compiled) .NET programs before conversion into machine code [23, p. 3]

(e.g. dysfunctional relational relationships between tables) and they prematurely terminated it.

WCF[7] architecture enables transferring such objects whose classes are defined on the server side. During development of the client application, the integrated development environment can determine the structure of these classes from the WSDL[8] document and automatically create and later upgrade their Silverlight version. But there is a fundamental disadvantage in WCF, that during serialization[9] it cannot keep references between objects. Each sub-object, which is referred to the other serialized object, is also serialized every time, repeatedly. For example, when a list of results from the testing will be sent via WCF and each result will also reference the tested user object, then the complete data of 100 users will be sent for the 100 different results, even if all results were linked to the same user. Then 100 independent objects of *User* class with the identical values are created by deserialization[10] on the client side. This fact increases bulkiness of transmitted data and also greatly complicates their object representation and processing.

The **RESTful**[11] services transmit data in XML or JSON[12]. All requests and queries for such data are part of the URL. This URL includes, for example, filter conditions, the authentication key etc. Part of the request sent via HTTP protocol can also be parameters unspecified in the URL, but attached by the POST, PUT and DELETE methods. These methods are used for modifying the data (POST = inserting a new record or collection, PUT = changing data of existing record or collection and DELETE = deleting a record or collection). [2] RESTful services are provided usually by servers for which it is desirable to support the development of independent applications that work with their data (e.g. Amazon[13]).

Exchanging data between applications of different developers is most frequently performed via an XML (or JSON) document currently. Its structure is usually defined by an author of one of the communicating applications and other authors adapt to this structure. **OData**[14] is one of the protocols that standardize this data structure and also the form of querying and handling with this data. Applications that support OData data exchange are then able to communicate without major adjustments of inputs or outputs. Online applications supporting OData are queried via the URL in a standard manner (similar to the RESTful services) including the above-mentioned problems with serialization of data.

A new solution was designed and subsequently created due to the aforementioned complications in transferring of object data – **Silverlight DataSet** [3], [4].

II. Silverlight DataSet

For the purpose of data exchange two libraries of classes were created. The first is for the Silverlight platform on the client side and facilitates the compilation of data requirements, deserialize incoming data, their administration and builds the list of changes. The other library receives requests on the server, serializes the required data and processes lists of changes (see Fig. 1).

The Silverlight part defines a class `DataObjectBase` that provides the basic features for individual data records. All the classes representing the tables in a relational database on the server must be derived from this class.

Linking of classes with records in tables and class properties with columns of these tables is done directly in the code of the class. Each such class must have an attribute[15] in which basic parameters for linking with the table can be defined, i.e. especially its database name. Similarly, the properties of this class. Properties may also have attributes that determine which columns of the table are bound with them. Attributes can also contain other flags, e.g. whether the value is read only, whether it is compulsory (`not null`) or delayed, and maximum length of text strings (see Code 1). Properties that do not have this attribute are ignored by DataSet and a used optionally by the developer.

```
[DataObjectAttribute("A_USERS")]
public class User : DataObjectBase
{
  private string surname;
  [DataObjectProperty("SURNAME", true, 30)]
  public string Surname
  {
    get { return surname; }
    set { surname = value;
          ValueChanged("Surname"); }
  }

  private int height;
  [DataObjectProperty("HEIGHT")]
  public int Height
  {
    get { return height; }
    set { height = value;
          ValueChanged("Height"); }
  }
  ...
}
```

Code 1 – Sample code of class definition linked to the database table using attributes

Code 1 shows the mapping method using attributes [5, p. 449]. The `User` class is linked to the table `A_USERS`, the property `Surname` is bound with the `SURNAME` column and the property `Height` is bound with the `HEIGHT` column. The attribute constructor also specifies that the name is

[7] WCF – Windows Communication Foundation

[8] WSDL – Web Services Description Language

[9] *Serialization saves the state of an object to the selected storage* (e.g. as a text string) *and deserialization from it retroactively reconstructs the original object* [24]

[10] *Deserialization* retroactively reconstructs the original object from its serialized form (e.g. text string or XML)

[11] REST – REpresentational State Transfer

[12] JSON – JavaScript Object Notation, www.json.org

[13] Amazon web services – aws.amazon.com

[14] OData – Open Data Protocol, www.oauth.net

[15] Custom attributes allow you to declaratively annotate the code constructs, thereby enabling special features, e.g. it can be queried at runtime by reflection and dynamically interpreted by a different code [25, p. 435]

a compulsory item (2nd parameter) and its maximum length is 30 characters (3rd parameter).[16] Calling the `ValueChanged` method in the `set`-part of property code allows for data continuity of the property with a visual component for editing, which can react accordingly to a possible change of property value (see [6]). This method also records all changes to the data, for their subsequent submit back to the server and their storing to the central database.

The DataSet performs compiling of the list of all these classes through reflection (see [7, p. 489]) and it is not necessary to draw them up somewhere else again. A transmit of `Assembly`[17] (in which the data classes are defined) to the main mediating class is sufficient during creating it. These classes are found on the basis of these attributes automatically, and they are mapped and stored in structured lists, so that the DataSet could work with them without further delay.

All data are downloaded in their original relational format and classes, that process these data on the client side, must adjust them appropriately. References to other tables (classes) are basically realized only through linking values (foreign keys) and do not by object references, but this functionality is not difficult to be programmed through the *LINQ*[18] *to Object* (see [8, p. 49]).

III. Loading of the Data

When an application needs some data (e.g. if the user wants to view some overview), a request for them is passed to the DataSet. This may be either object oriented or in the form of a text string, or an as XML element of specific structure (see [9] and [10]). Sequence diagram in Fig. 3 shows how a request for data is processed asynchronously.

The request may contain links to multiple tables simultaneously. A filter for the required data is compared always with filters of already downloaded data for each of these tables. If the DataSet evaluates that all the data have already been downloaded earlier, it directly notifies the application to continue. Otherwise, the DataSet builds a requirement into a text string which is passed back to the application. The application sends the request to the server (e.g. via web service), where it is passed to the web part of the DataSet. The DataSet deserializes it into a structured data object and returns it to the server application. The application processes the request in a way that it loads required data from the database to the classic *DataSet* object, which is part of the .NET Framework [11, p. 309]. At this point, the application can check also the authentication of a user, that requires the data; whether the user has the appropriate permissions to read them or not. The *DataSet* with loaded data is passed back to the `WebDataSet` class that serializes these data into a text string. The WebDataSet returns this string to the web part of the application to send back to the client.

The client application receives the string with data and transmits them into its part of the Silverlight DataSet. The DataSet processes them (deserializes them to the appropriate classes, adds records to the objects lists and restores relations between objects) and then notifies the application that everything is ready. The application can then start working with the data.

All data sent over the internet is sent as a text string throughout the entire process [12]. Yet the application works only with object data. Complete process of serialization, deserialization and management of the data is catered by the appropriate library of the DataSet. Database connection and sending and receiving of data is provided by the application. DataSet is independent of the environment in which it is used and it maximally simplifies all routine work with the data.

A. Filters

Filters are an important part of the DataSet. The main assumption is the fact that only the data that is needed now are always downloaded from the server. In order to not have to always download the complete data of whole table, the filters can determine, what part of the table (horizontal) is currently being needed.

The filters are based on a text strings. Developer (author of the application that using the DataSet) defines format of these strings. Because he also implements a direct database connection, evaluating of these filters is only in his jurisdiction. DataSet on the client side recognizes itself only two cases: when the same filter is used repeatedly, or if all data from whole table has been downloaded completely. Both cases are a sign that for the requirement for data is no longer need to download anything new. The application author must ensure a comparison of other types of filters.

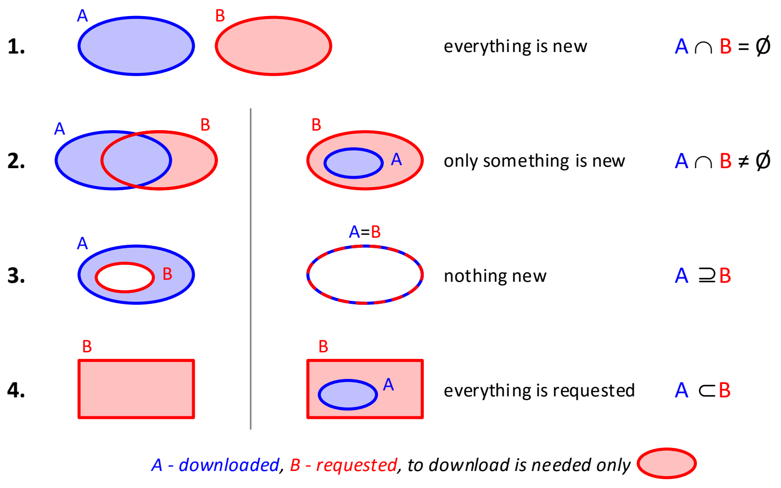

Fig. 2 – Possible types of results of comparison of data filters of already downloaded data and required data

[16] A similar way of the definition of links is also used e.g. XPO from DevExpress (www.devexpress.com/products/NET/ORM)

[17] Assembly is the logical unit that contains compiled code targeted at the .NET Framework [23, p. 17]

[18] LINQ – Language INtegrated Query

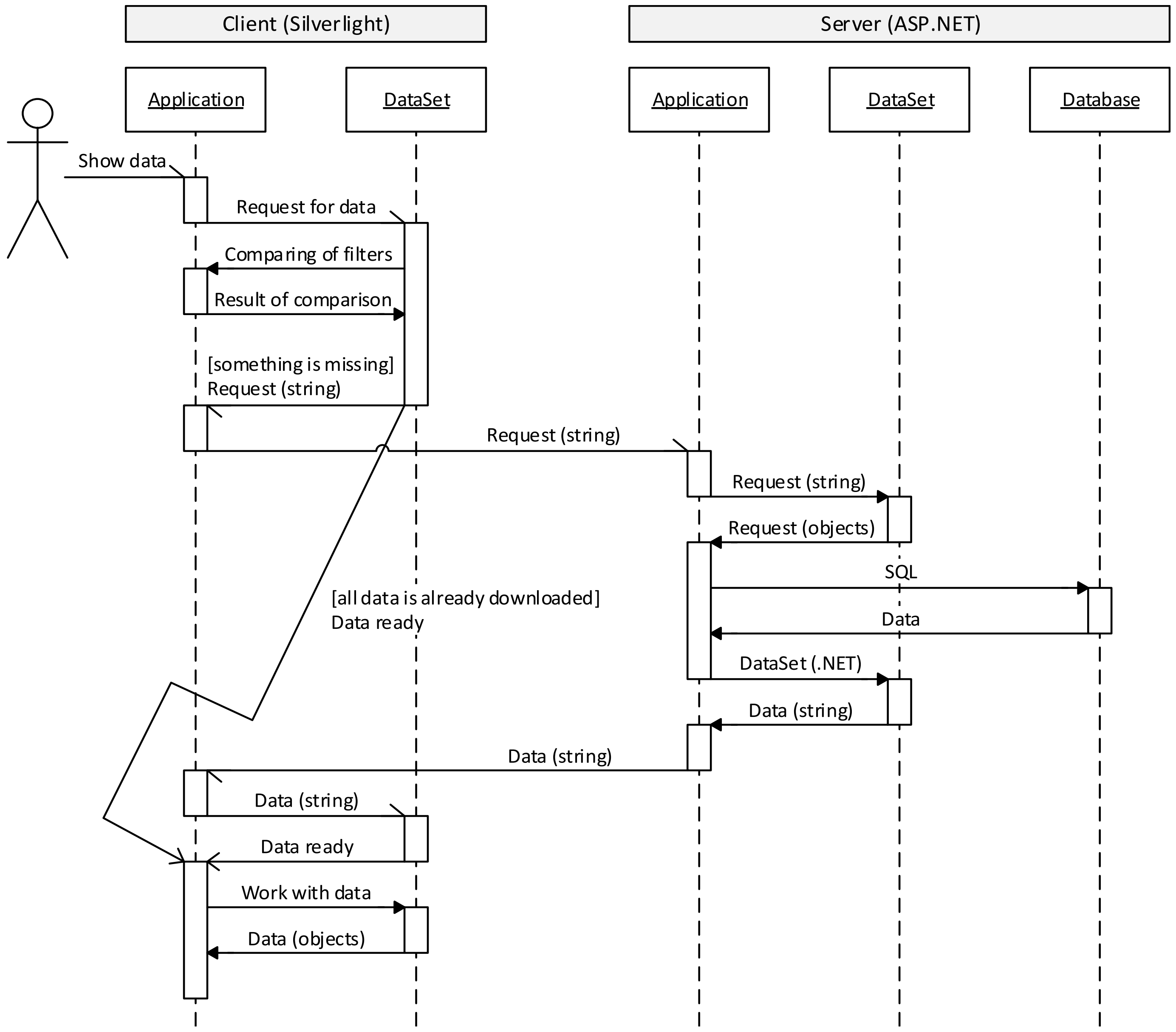

Fig. 3 – Sequence diagram of the process to read data using the DataSet

The DataSet exposes a delegate `CompareFilters`, which (if set) is executed whenever the two filters be compared and the DataSet is unable to evaluate them itself. The application then returns to the DataSet the result of this comparison as an enumeration `FiltersCompareResult`. Possible results are shown in Fig. 2.

In the first case it is necessary to download everything. In the second case, it is needed to download only a part of the data, so the DataSet to the filter adds a compressed list of IDs of already downloaded records and these records will not be downloaded again. In the third case, when there is no need to download, the request for data is directly evaluated as finished without the need to contact the server. The last fourth case is treated the same as the second case, but for next time is noted that all the data of this table are completely downloaded, which is advantageous for example for codebooks.

Filters are therefore in all cases (except the third) in its unchanged version sent to the server, where they are after deserialization of requirement forwarded to this part of the application for retrieve of the data delimited by these filters. In the second and fourth case a list of ID of records is also attached (as `IEnumerable<int>`), which is no longer needed to download.

B. Delayed data

Fields of certain tables may contain more extensive data with variable length. For example, the Firebird[19] database supports BLOB[20] data type for storing binary data or very long texts, which cannot be easily stored in columns of any other standard data type. [13, pp. 181-190]

Such data is not appropriate for bulk transfer of multiple records at one time. The reason is bulkiness of these data and

[19] Firebird – open-source relational database system, www.firebirdsql.org

[20] BLOB – Binary Large OBject

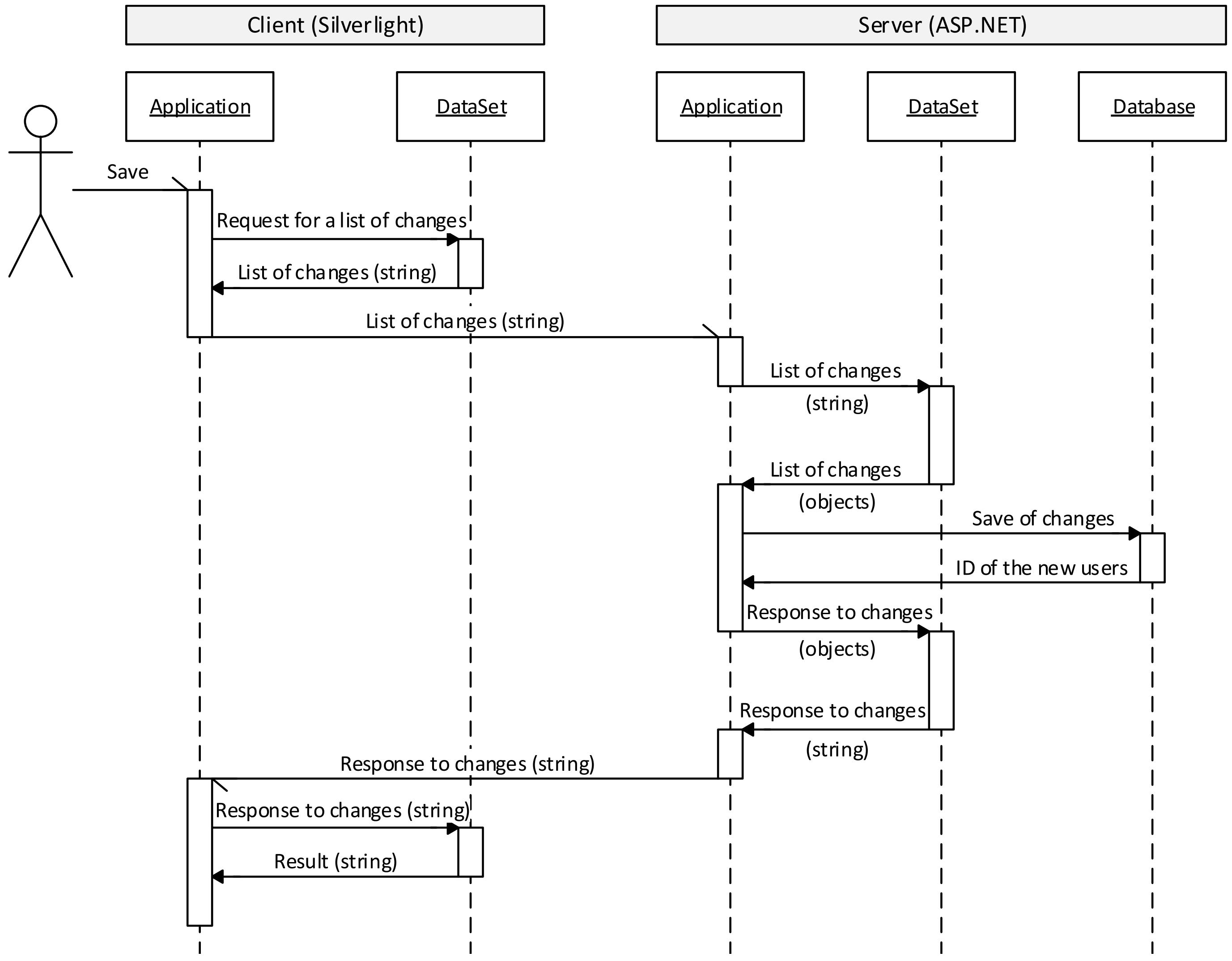

Fig. 4 – Sequence diagram of the process of storing of data changes using the DataSet

also the fact that the data do not appear directly in a table overviews (e.g. QML data of a test question). These data are usually displayed when the detailed listing of the one individual record is requested, so these data only for one particular record is needed to download until at this moment.

These types of values can be distinguished in the attribute of the property in the class definition (see Code 1) as another optional parameter. The DataSet then does not download them when a bulk data is requested, but only on special request. This requirement must relate to one object specified by its ID and it must include a list of items of this kind of values (table columns) to be downloaded (see [9]). Simultaneously, for each object is separately registered, which items of delayed data have been downloaded for distinguished them from empty fields (null) and don't download them unnecessarily again.

IV. Saving of Changes

Any changes of the data are collected and recorded in a special list. Repeated changes of the same data are recognized and only the last change is registered with regard to its type. There are most commonly three types of changes (sorted according to the priority):

- **Delete** – If the data record is deleted, the previous changes are not necessarily further registered, only the ID of this record is needed. If the record was previously added and it is not yet saved, then it is removed from the list changes completely.
- **Insert** – The newly inserted record always sends all values to the server, even if it is later changed (before submitting).
- **Update** – Any changes of values in existing records are recorded and only their new values are sent to the server.

Fig. 4 shows the progress of the process of saving changes to the server. At the moment a user requests saving changes to the server, the application delegates the DataSet to assembly a serialized list of changes. This list is sent to the server, where it is passed to the web part of DataSet for deserialization. The returned object contains "understandable" list of changes which the application stores in the database. This operation should be performed in a transaction [14], because in case of a failure, the incomplete data changes will be avoided. The application, of course, can support this procedure. Possible competitive changes made by another user can also be checked at this stage, e.g. by the means of a time stamp.

A newly allocated ID is added to the list of changes for newly inserted objects by the database during data storage or after confirmation of the transaction (depending on the system of connection to the database). Furthermore, it is also necessary to confirm that every change was stored successfully, or specify the error message that arose when trying to save.

The object of list of changes supplemented by these reactions to changes is passed back to the DataSet for serialization and the output string is sent back to the client application. The client application passes the reaction string to its version of the DataSet, which returns either an empty string as the proof of everything being OK, or it returns an uncluttered list of error messages that arose during the process.

V. Conclusion

The DataSet for Silverlight is a useful tool for data transfer between the Silverlight application and the server. It tries to minimize the connection to the server and thereby reduce the burden, but also accelerate work with the client application, thereby the user comfort is improved while working with it. The DataSet receives data in standard relational form on the server side, and on the client side, it translates and manages data as objects of classes defined by the author of application. Definitions of relationships of these objects with database elements is extremely simple, non-redundant and clear.

The DataSet, in combination with the authentication protocol (see [15] and [16]) and encryption (see [17] and [18]), is a safe and easy-to-use tool for the Silverlight that enables the creators of cloud applications to concentrate more on the development of application logic without the need of solving the problems associated with the transmission and security of data via the public Internet.

Basic classes `SilverlightDataSet` and `WebDataSet` can easily be integrated into the project and set. These classes can assist in the development of such applications that work with the centrally stored data. Procedures that were used to implement these classes can be used in other environments and languages easily. It can be also an inspiration for further development in the area of data transfer in the cloud computing and RIA applications (e.g. [19]). For example, the client part of the DataSet could be adjusted for use in modern UWP[21] applications which can be also used for controlling of IoT with *Windows 10 IoT* operating system.

Acknowledgment

This research work has been supported by specific research project of University of Hradec Kralove, Faculty of Education and Faculty of Science in 2015.

References

[1] D. Esposito, *Building Web Solutions with ASP.Net and ADO.NET*. Redmond: Microsoft Press, 2002, ISBN 978-0735615786.

[2] L. Richardson and S. Ruby, *RESTful Web Services*. Sebastopol: O'Reilly Media, 2007, ISBN 978-0-596-52926-0.

[3] P. Voborník, "Effective object-relational mapping data transfer in the cloud computing," in *Information Technology for Practice 2011*, Ostrava: VŠB-TUO, 2011, pp. 189–197, ISBN 978-80-248-2487-1.

[4] P. Voborník, "Tool and mechanisms for efficient transfer of data in cloud client-server applications," in *Recent Advances on Systems, Signals, Control, Communications and Computers*, Budapest, Hungary, WSEAS Press, 2015, pp. 166–171, ISBN 978-1-61804-355-9, ISSN 1790-5117.

[5] J. Liberty and D. Xie, *Programming C# 3.0*. 5th ed., Sebastopol: O'Reilly Media, 2008, ISBN 978-0-596-52743-3.

[6] J. Papa, *Data-Driven Services with Silverlight 2*. Sebastopol: O'Reilly Media, 2009, ISBN 978-0596523091.

[7] J. Hilyard and S. Teilhet, *C# 3.0 Cookbook*. 3rd ed., Sebastopol: O'Reilly Media, 2007, ISBN 978-0-596-51610-9.

[8] P. Pialorsi and M. Russo, *Programming Microsoft LINQ*. Redmond: Microsoft Press, 2008, ISBN 978-0735624009.

[9] P. Voborník, "Concept for development of large-scale applications through configuration frameworks", in *Recent Advances on Systems, Signals, Control, Communications and Computers*, Budapest, Hungary, WSEAS Press, 2015, pp. 83–90, ISBN 978-1-61804-355-9.

[10] P. Voborník, "Configuration Frameworks," *International Journal of Mathematics and Computers in Simulation*, vol. 10, 2016, pp. 180–191, ISSN 1998-0159.

[11] M. MacDonald and M. Szpuszta, *Pro ASP.NET 3.5 in C# 2008: Includes Silverlight 2*. 3rd ed., New York: Apress, 2008, ISBN 978-1430215677.

[12] V. Strnadová, *Interpersonal communication*. Hradec Králové: Gaudeamus, 2011, 543 p., ISBN 978-80-7435-157-0.

[13] H. Borrie, *The Firebird Book: A Reference for Database Developers*. New York: Apress, 2004, ISBN 978-1-59059-279-3.

[14] J. Lowy, *Introducing System.Transactions*. Microsoft, 2005. Available: http://msdn.microsoft.com/en-us/library/ms973865.aspx

[15] P. Voborník, "Secure authentication in client-server applications on the Internet through a mandatory unique salts," in *Internet, Competitiveness and the Organisational Security 2011*, Zlín: Tomas Bata University, 2011, pp. 347–354, ISBN 978-80-7454-012-7.

[16] P. Voborník, "Fast multiplatform authentication on the Internet," in *IP Networking 1 – Theory and practice (proceedings of scientific works)*, Žilina, University of Žilina, EDIS, 2011, pp. 41–45, ISBN 978-80-554-0494-3.

[17] P. Voborník, "Modification of the perfect cipher for practical use," in *Advances in Mathematical Models and Production Systems in Engineering, Proceedings of the 7th International Conference on Manufacturing Engineering, Quality and Production Systems (MEQAPS '14)*, Brasov, WSEAS Press, 2014, pp. 64–68, ISSN 2227-4588, ISBN 978-960-474-387-2.

[18] P. Voborník, "Migration of the Perfect Cipher to the Current Computing Environment," *WSEAS Transactions on Information Science and Applications*, vol. 11, art. #21, pp. 196–203, 2014, ISSN 1790-0832.

[19] Š. Hubálovský, "Remote desktop access us a method of learning of programming in distance study," in *14th International Conference on Interactive Collaborative Learning (ICL2011) – 11th International Conference Virtual University (VU'11)*, Bratislava, Slovak University of Technology in Bratislava, 2011, pp. 450–455, ISBN 978-1-4577-1746-8.

[20] P. Voborník, *Universal Testing Environment*. Ph.D. thesis, Hradec Králové: University of Hradec Králové, 2012. Available: http://download.petrvobornik.cz/docs/disertace.pdf

[21] T. Lammarsch, W. Aigner, A. Bertone, S. Miksch, T. Turic and J. Gärtner, "A Comparison of Programming Platforms for Interactive Visualization in Web Browser Based Applications," in *International Conference Information Visualisation*, Washington DC, USA: IEEE Computer Society, 2008, ISBN 978-0-7695-3268-4.

[22] T. Kanik, "Architecture development of web based application for a construction cost prediction," in *Proceedings of the International Conference OSSConf 2010*, Žilina: Society for Open Information Technologies, 2010, ISBN 978-80-970457-0-8.

[23] S. Robinson, C. Nagel, J. Glynn, M. Skinner, K. Watson and B. Evjen, *Professional C#*. 3rd ed., Indianapolis: Wiley Publishing, 2004.

[24] S. Manoharan, "Application state, serialization and versioning," in *Proceedings of IADIS International Conference Applied Computing 2004*, Lisabon: IADIS, 2004, pp. 57–60, ISBN 972-98947-3-6.

[25] J. Richter, *CLR via C#*. 3rd ed., Redmond: Microsoft Press, 2010.

[21] UWP – Universal Windows Platform, https://msdn.microsoft.com/en-us/library/dn894631.aspx