A MODEL OF A SYSTEM FOR CENTRALIZED SYNCHRONIZATION OF INDEPENDENT APPLICATIONS[1]

Nikola Valchanov*, Todorka Terzieva & Anton Iliev*****
Bulgaria, 4003 Plovdiv, 236, Bulgaria blvd., University of Plovdiv,
Faculty of Mathematics and Informatics
*, *** & Bulgaria, 1113 Sofia, Acad. Georgi Bonchev Str., bl. 8, Bulgarian Academy of Sciences, Institute of Mathematics and Informatics
*nvalchanov@gmail.com, **dora@uni-plovdiv.bg, ***aii@uni-plovdiv.bg

ABSTRACT

One of the most pressing problems in information systems is distributed data storage. The work of territorially separated branches, different delivery services, support departments, etc. is impossible without individual data bases, whose information constantly changes. This causes the need of data synchronization between the different departments of the organization. For the development of modern corporate applications, we need a solution that comprises the servers, workstations and the notebook - collaborators in a single information environment. In addition, every participant must receive specific information and exchange specific information with the central server in the company's office. It is of great importance that the system is reliable and needs minimal administrator intervention.

There are two basic approaches:

The first one is to use Web based solutions. In this way, all users will actually work directly with the centralized data base. In that case, merging the information becomes unnecessary. A substantial plus is the extremely facilitated

[1] This paper is partially supported by projects ISM-4 and MU-03-09 of the Department for Scientific Research, Plovdiv University "Paisiy Hilendarski".

support of the software. It’s deployed only once on the hosting server (of course, when the system load is high, the number of servers can be easily increased). Another substantial asset is that the Web technology does not depend on the hardware platform and works on every computer, never mind their operating system or computational power. Amongst the drawbacks of the Web based applications is the fact that all remote objects must have fast and constant connectivity with the central office. Unfortunately, the connectivity cannot be 100% guaranteed. It is location and provider dependant – there are places that have no internet connectivity, which will be a problem for the notebook associates, on the other hand no internet provider can guarantee that his service will not stop due to technical failure. These problems could block the work of the branch, notebook associate or the entire company which is unacceptable for most of the businesses.

The second possible solution is synchronization between the different data bases. In this way, every object would work independently from the others. At a given moment, they would synchronize their data during short communication sessions. During those sessions, the objects would exchange information about the changes that have taken place in their data bases. In this way, the different objects would no longer be dependent on the connectivity with the central server. Most commonly, the changes between two different sessions are exchanged between the data bases by exporting data from the source and later importing them into the target. The process is usually done by an operator, which on one hand makes it human dependant, and on the other poses a threat of error, for example “forgetting” to export or import a particular document.

The subject of this paper is the data synchronization between independent applications.

The data synchronization is a process during which the data for an object (product) or a person (business partner) are made consistent between computer systems and applications in an automated manner.

The problem with data synchronization has been solved differently during the different periods of computer systems evolution. Some applications depend on the replication tools provided by the data base servers. The data replication is a way of asynchronous exchange of information between SQL servers. It allows different departments of a company that are connected through Internet or Ethernet to work with big amounts of data exchanging only the last changes in their data sources. The central server processes and sends the data to the subordinate servers. In that way, each of them has actual data. The synchronization is done automatically, which eliminates the probability of human error. All data is up to date and, on top of that, the data is stored at several places, which eliminates the need for backup or archives.

Another approach is the implementation of custom libraries for internal synchronization. This method is applied in the cases where the system is data source independent or works with data sources that do not provide replication tools. These solutions are extremely effective and reliable, as their implementation is tightly connected with the business processes of the specific system. They can be optimized for the structure of the data that they work with. In this way, the system can gain performance. Another asset of the method is the better control over the system. Applications that work with replication mechanisms implemented in the SQL servers have no knowledge of the internal processes that are executed during data synchronization. In this case, the replication framework is a black box for the system and the developers – it is used as is and cannot be examined or changed for optimization purposes. Development of a custom library for internal synchronization, on the other hand, allows direct control over the process and all problems related to it.

Unfortunately, these two methods are not always applicable. Let us take as an example a case where the goal is to synchronize the data of independent applications. The term independent application would mean an information system with a different structure of the data they store, using different data sources, and developed on different platforms. In this scenario, the systems are given as black boxes and the only opportunity for automated access to the information that they work with is through their instruments for importing and exporting data.

Based on these instruments, we can build a common model of a system for centralized synchronization that keeps the consistency of the data that the applications work with.

INVESTIGATION OF THE PROBLEM

In computer informatics, we often speak of cross-platform applications, code flexibility and interoperability.

In the business logic architecture of big systems, this is achieved with design patterns. Let us take the adapter pattern [1], for instance. Its goal is to transform the interface of a given class into another interface that is expected by the client library. In this way the adapter pattern allows classes to work together even in the cases where it is impossible due to incompatibility of their interfaces.

When the goal is building a centralized business system that will have client applications, working on different platforms – Web services [2] come to the rescue. The fact that they work with a unified text format (XML [3]) allows their clients to be completely independent of the technology used for the Web service development. XML documents are nothing more than simple text files structured in a specific format. They are not dependent on the operation system, the development platform or other factors.

Let us now review the following case scenario: let us have two or more independent applications developed on different technologies that need to work in a synchronized manner. Let the only common thing between them be the ability to export and import data in a format that is specific for each one of them (functionality widely supported by the contemporary information systems). In order to be completely different, the data base structure should vary between the different applications.

One possible solution is the development of a new product that provides the needed combined functionality. This would include the development, integration of the new product and migration of the old data. This approach seems the most clean and painless way of solving the problem. The new system could be designed based on the experience of the organization and could completely model the robust business processes in it. On the other hand, the assets of the old products could be kept and their drawbacks – taken in mind. As a final product, we will obtain a system corresponding completely to the specific needs of the organization. The problem here is that this solution usually involves time and resources for the development and the data migration.

Another possibility is merging the separated systems into one or their appropriate expansion so that they could interact. This approach is good but not always applicable. Merging systems is practically impossible when they're based on different technologies. System expansion also depends on external factors. A company could stop the support of an obsolete product. Having these drawbacks in mind, it is only natural to look for a solution that is not bound to any of the systems in particular. It has to be general and should not depend on their specifics (technology, built-in synchronization mechanisms etc.).

It is not hard to see the applicability of the adapter concept here. It can be extended as a mediating application between the synchronized systems. In order to be independent of the technologies used to create each one of the applications, we can use a unified format of communication between them.

MODEL BASICS

Let us take for granted that all systems support data export and import in a format that is specific for each one of them. In this way, we could right away do a manual synchronization of two systems by exporting the common data and for each system converting the obtained information to a readable format and importing it.

It is obvious that this process could be automated. The synchronization system could keep track of the registered applications and the correlation of the records from the different data sources. Also, it should contain a full description of the operations supported by the import and export tools for each of the registered applications. A description of one way synchronization processes could be also kept by the system – export operation of the source application and import operation of the target application. These synchronization operations should contain information about the transformation that has to be applied to the data before the import operation so that they'll be readable by the target import tool.

This way the synchronization system does not keep implementation specific information about the applications it works with. It only knows the format used to communicate with every one of them and it is completely sufficient for conducting the synchronization process.

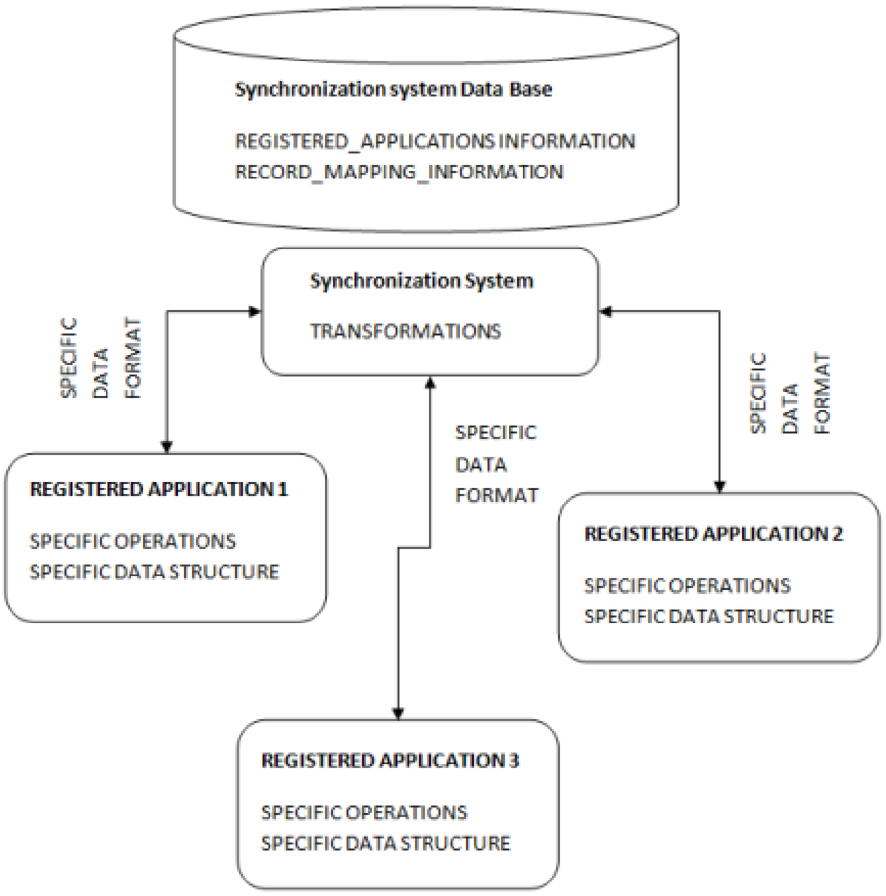

Fig. 1 Common schema of the synchronization mechanism

A MORE TECHNICAL LOOK

Technically, the data that the model would work with can be separated in three basic nomenclatures: registered applications, operations and transformations.

Corresponding to each nomenclature should be a table in the system's data base:

- **Registered Applications** – every application is described by a name, local path to its working directory and executable name that provides access to the import/export tool of the system.
- **Operations** – every operation is strictly associated with a given application. It is described with an operation name, operation type (import/export), validation schema that would validate the data before executing the import operation to avoid invalid format exceptions and an operation script that would be a parameterized pattern with instrument settings for the given operation.

- **Transformations** – every transformation contains an ordered couple of export operation and import operation that belong to different applications. The transformation contains a transformation script that converts the data obtained by the export operation to a valid format readable by the target application.

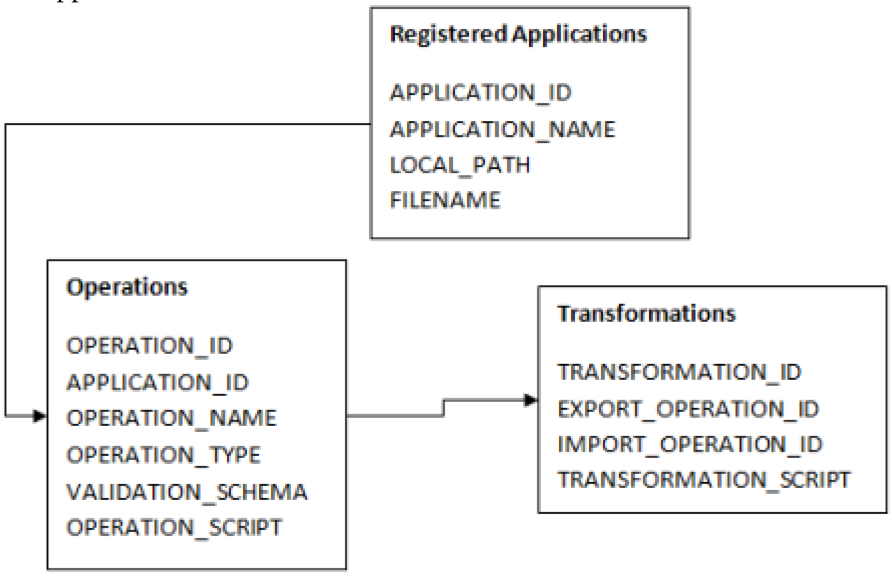

Fig. 2 Structure of the data about the registered applications, operations and transformations

The inner structure of the model could be organized in collection classes that implement the communication with the data source and the data persistence. Every one of the system's entities is described in its own class.

While the **RegisteredApp** class does not have a specific functionality, the classes **AppOperation**, **DataTransformation**, **XSLTransformation** and **CSVTransformation** implement processes execution and operation management.

The method **AppOperation**.**Execute**() executes the given operation. If it is an export operation, it validates it and returns a stream with the result from it. In the other case, it validates it and imports the received data.

The **DataTransformation** class contains the data transformation logic. It is used to transform the data obtained by one operation to a format that validates against the validation schema of the corresponding input operation. This class is designed by the strategy pattern, completely extracting the transformation logic in a class member that implements the **ITransformation** interface. In this way, the system is not bound to a specific format. The exported data could be XML, CSV etc. and adding a different format would only mean developing another class that implements the **ITransformation** interface. Another asset of this approach is that the system is not bond to a specific transformation mechanism (XSLT, custom transformation mechanism etc.). The choice of the transformation algorithm could be done dynamically from a configuration file or can be stored as additional information for the given transformation or export operation. The **SynchFacade** class is used to separate the inner work of the synchronization mechanism from the other part of the system. It allows the client code to execute synchronization without having specific knowledge of the process or the internal library classes.

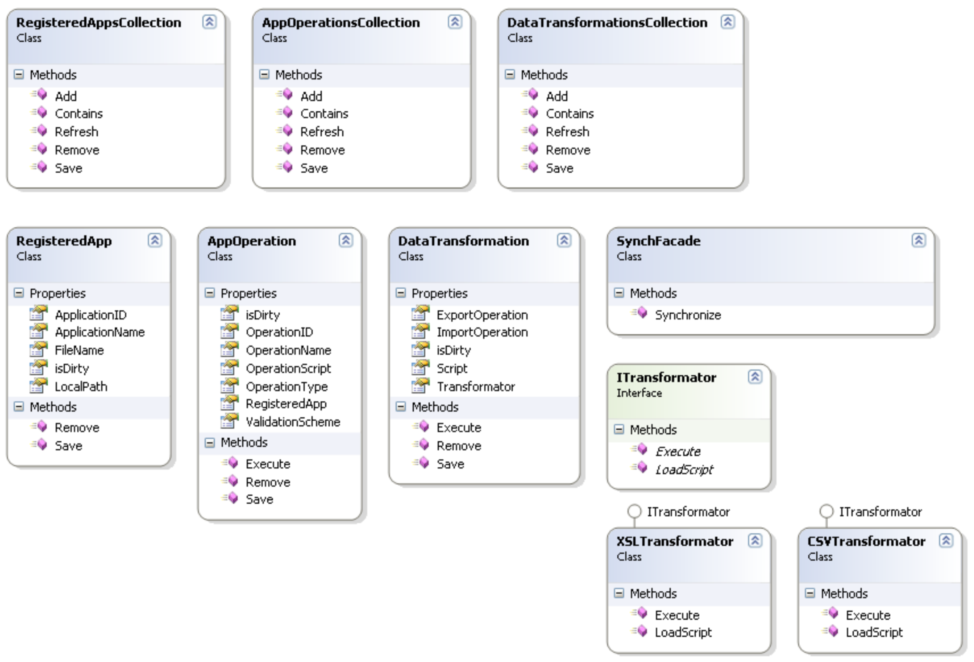

Fig. 3 Sample class diagram of the model

CONCLUSION

The market offers a wide range of implementations of synchronization frameworks. They are often bound to a specific technology, which narrows the scope of their application. Despite the relatively simple structure of the model described in this paper, its way of operation allows it to be applied for solving problems that would hamper other synchronization systems. The examples presented in this paper describe some of these scenarios in details. The model is technology independent and can be applied for synchronization of any systems. Of course, the idea is developed at a very basic conceptual level. Serious problems that an eventual implementation would face could be primary key mapping for the synchronized records, optimizations of the data base targeting duplicate records elimination, development of specific algorithms for searching and identifying of duplicate records in two record sets and last but not least development of algorithms for identification of deleted and changed records. Specific implementation of this model would have a wide scope of application. It would provide a way for direct access to the synchronization processes, for setting up scheduled process execution and data migration from one system to another.

REFERENCES

[1] Gamma, E., R. Helm, R. Johnson, J. Vlissides, Design Patterns, 1995.
[2] MSDN, http://msdn.microsoft.com/en-us/library/ms188266.aspx
[3] Wikipedia, http://en.wikipedia.org/wiki/XML