# A Lightweight Data Sharing System Based on Bidirectional Transformations

Adrien Duchêne
University of Namur
Namur, Belgium
adrien.duchene@student.unamur.be

Hugues Marchal
University of Namur
Namur, Belgium
hugues.marchal@student.unamur.be

Zhenjiang Hu
National Institute of Informatics
Tokyo, Japan
hu@nii.ac.jp

Pierre-Yves Schobbens
University of Namur
Namur, Belgium
pierre-yves.schobbens@unamur.be

## ABSTRACT

Although the data sharing and synchronizing problems have been raised many years ago, they remain major issues in the database community. Still, some tools are provided to end-users in order to answer some of their needs. Yet, those platforms are most likely very complicated to handle notably because they ask the user to have very much knowledge, the user sometimes being the developer. Also, most of those systems do not really insure data consistency. Our approach based on bidirectional transformations (BXs) resolves collaboration between companies having their own data structure in an easier way, guaranteeing data consistency thanks to BXs. All this means that the user does not need to know databases structure other than his and the shared mappings, and will also never be asked to use pure code or database knowledge, limiting then the complexity. In addition to this, the system profits the bidirectional transformations properties to authorize or not editing the shared data. The bidirectional functions coded in BiGUL have indeed the power to grant or not any other user in the sharing group to edit the data. Moreover, the system is extensible in the way that the user can easily join a sharing group, after providing to the bidirectional functions a GLAV mapping table matching his local structure with the shared one.

## CCS CONCEPTS

• **Information systems → Data exchange**;

## KEYWORDS

Data sharing, bidirectional transformations, BiGUL, databases, lightweight selective data sharing system.

**ACM Reference Format:**
Adrien Duchêne, Hugues Marchal, Zhenjiang Hu, and Pierre-Yves Schobbens. 2018. A Lightweight Data Sharing System Based on Bidirectional Transformations. In *Proceedings of 2nd International Conference on the Art, Science, and Engineering of Programming (<Programming'18> Companion).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3191697.3191722


*<Programming'18> Companion, April 9–12, 2018, Nice, France*

ACM ISBN 978-1-4503-5513-1/18/04…$15.00
https://doi.org/10.1145/3191697.3191722

## 1 INTRODUCTION

Considering two companies close to each other due to common research field, it is possible that they may want to share some of their data. Yet, the two companies might be different in the way they operate, for storing data notably. Because of this, the structure of the database is unlikely to be the same, even though it may be related. Despite all those constraints, they still want to be able to share relevant and consistent data without having to know every technical aspect of the sharing process, including each other's structures.

This problem of sharing data between companies or even among them has been raised many years ago and has already been answered in some way such as the Hyperion Project [2], ORCHESTRA [5] and PeerCSDB [11], for example. Yet, most platforms are not very user-friendly, either by showing so much information to the user or by asking him to know how to code or how all the databases are structured. Also, most of those platforms cannot really guarantee the consistency of the data they share. Still, this last concern has already been taken into account, notably by the "Relational Lenses" [3] platform, using bidirectional properties. Nevertheless, this last approach remains quite difficult to handle, as mentioned before.

In this paper, we propose a user-friendly lightweight system based on bidirectional transformations [10] that can consistently manage the data sharing process. We consider the system lightweight thanks to the small amount of code needed to implement all the features and because it does not need all the "databases functions" used in the "Relational Lenses". Indeed, our system does neither need the "join" function nor the proofs of well-behaveness of BiGUL. The system provides an interface limiting the needed knowledge and enhancing its easiness. In this case, the user will neither be directly drown inside the complicated database nor be coding its way to share the data, as the roles of the user and the developer are completely distinct. In our system, once the data to share has been easily determined, the bidirectional functions take care of concretely sharing it in background. The following summarize the three distinguished features of our system.

- Our system being based on BXs and coded in BiGUL [6, 8, 9], a bidirectional language, its well-behaveness is achieved for free. Indeed, bidirectionality is notably used to enhance the

insurance of consistent propagation, strong integrity and update of the data thanks to its well-behaveness properties.
- Our system uses the authorization property provided by the bidirectional transformations themselves. With such property, the system is able to authorize or not the propagation of updates depending on the will of the user who originally possessed the data. The data is then respectively *writable* or *read-only*.
- Our system is extensible. Indeed, it has been conceived in a way that a new user can easily join the group sharing data. The newcomer just needs to provide the BiGUL functions a GLAV mapping table (*Global-Local-As-View*) [7] matching his local format to the shared one. When that is done, the system uses these BiGUL functions to retrieve and later update the shared content, if authorized.

This paper is organized as follows. Section 2 gives a technical and user perspective of the system, while Section 3 explains a concrete example of the usage of the system. Finally, conclusion and future work can be found in Section 4.

## 2 LIGHTWEIGHT SELECTIVE DATA SHARING SYSTEM

First, the system is described in the end-user point of view, showing what the user needs to do in order to share data. Second, a more technical perspective is given, describing in the large the background aspects of the system. Thus, the front-end of the blackbox system is described in the user perspective while the back-end is in the technical perspective. It is important to note that the system has been developed in Haskell and obviously BiGUL, using a MySQL server [1]. The implementation is available at [4].

### 2.1 User Perspective of the System

As previously mentioned, the system only asks the user essential information in an understandable way, without using specific code or notations, for example.

At the first use of the system, it is mandatory for the user to use the *"setup"* feature provided. He is only asked which database he wants to use for data sharing. After verifying that the mapping table exists in the database, the system displays that the setup is complete. Note that the system is configured using a setup file which is easily editable. It notably contains access to the database.

```
*MainU> setup
Which Database do you want to use ?
bigul
"Setup done"
*MainU> main
Which Database do you want to use ?
bigul
Push or pull data ?
push
Which Table do you want to use ? (':q' to quit)
["a_b","album","album_tracks_join","bformat","mapt","test","tracks"]
album
Select columns you want to synchronize
["Album","Quantity","Rating"]
Album Quantity
Selection column ?
["Album","Quantity"]
Album
Condition ? (For example : '==10') ('*' for all)
==Show
Writable (T/F) ?
T
Do you want to synchronize this data (Y/N) ?
TableSync "album" ["Album","Quantity"] [([SqlByteString "Show",SqlInt32 3],True)]
Y
```

**Figure 1: The user interface of the system to push data**

The user can now start the sharing process. The system then asks him which database he wants to use and whether he wants to push or pull the data on or from the network.

In the case of a push, the list of tables inside the database is shown so that the user can choose one of them. After, all the columns of the selected table are displayed, and again, the user needs to choose which ones he wants to share with others. Then, he needs to provide the column on which the selection condition is applied and the condition itself. Concerning the condition, the user only writes the comparison operator and value, as shown in Figure 1. Then, he is asked whether he wants the data to be editable or not. He can answer "T" (*True*) for the data to be *writable* and "F" (*False*) for *read-only*. Finally, the user is asked to confirm that the data to share is the one displayed on the screen. If he accepts, the system then automatically shares the data using bidirectional transformations and restarts the procedure to push data, displaying the list of tables. Note that the user is able to exit the system by typing ":q" whenever after the list of tables is displayed.

To receive shared data, the user needs to choose "pull" instead of "push". He is then requested to enter some keys provided by other users, representing the data on the network. After entering those keys, the system, based on bidirectional transformations takes care of the rest, even putting the data back in database, including updating existing data. Putting back in database and updating are yet conditioned by the existence of relations in the table mapping the shared format to the local one. If some mappings are not defined in the GLAV mapping table, depending on the configuration file, the user will either be asked to give a match, or the match will be automatically given by the system.

### 2.2 Technical Overview of the System

To better understand, this section is explained in a top-down approach, shown in Figure 2, pushing the data on the network. Therefore, first comes the extraction of data from the database. In order to do so, the user is asked to give all the input stated in the previous section.

Note that the system queries the database during the "input process" to display the needed information. Yet, the data itself is only retrieved when all the input has been entered. It is then given to the BiGUL function where the data is consistently transformed so that it contains the authorization boolean entered by the user. This BiGUL function also selects the columns and rows from the retrieved data, obviously following the user input. The selection of the columns is coded so that the view is set to contain only the selected columns of the source. Also note that as the function is bidirectional, an updated source can be put by adding the unshared columns to the source. The rows are selected by applying the condition given by the user to each row in the source. When it is satisfied, the row in question is directly set in the view. This function being bidirectional, to update the source from the view, the condition is set on the identifier of the row. This therefore allows to add new rows to the source.

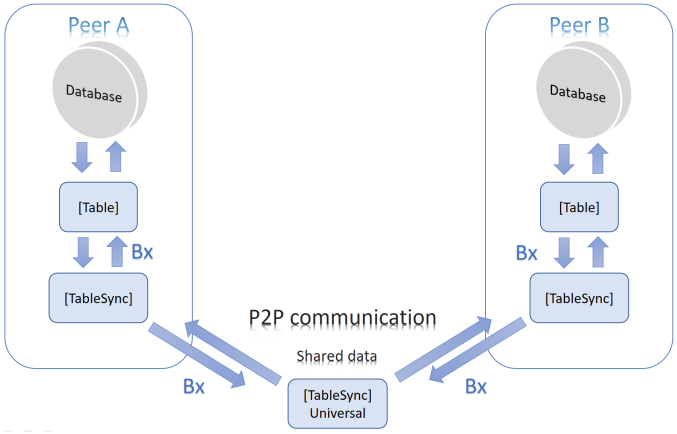


**Figure 2: Global schema of the system**

After the selection, the tables are matched to a *"universal format"* significant only to the group of people sharing this data. This provides a way for the user to only know what is his data structure and what is happening inside of it, without even knowing a single column of the others' database. By using this *"universal format"* via mapping tables and bidirectional transformations, the guarantee of consistency is still preserved and the system is greatly simplified for the user.

The BiGUL function caring for the conversion takes the mapping table as parameter and for every row in the source, consistently changes the name of the identifier to form the view. The reverse operation is clearly possible. It is also important to note that the names of the columns and table are translated too, using the same BiGUL function. As the authorization boolean has been included in the data, the BiGUL function is coded so that only the *writable* content satisfying the condition is pushed on the network. When the data is pulled form the network, as all the operations stated are executed in the reverse order, the boolean cannot be modified by the user, guaranteeing the respect of the authorizations.

When the content has been converted to the universal data, it can be sent on the peer-to-peer network, used mainly for the time-shared storage property. The data is then always available and retrievable by its representative keys. Those keys are useful for the network propagation protocol, Chord [12], because they improve the complexity of the peer-to-peer protocol thanks to consistent hashing.

Finally, it is important to remember that most of the system is coded in BiGUL, so functions are meant to do round-trip modifications. In other words, most functions, even though only one implementation exists, can be used for both sending and receiving data. In this case, the procedure is applied in the exact opposite order, from the network propagation to the insertion or update of the database, passing by the matching of *"universal format"* to the local one and the reconciliation of shared data to the tables, as Figure 2 suggests. A critical aspect of the system is its well-behaveness, thanks to the bidirectional transformations. Yet, the behavioral correctness of the functions still needs to be proven. Our system establishes this correctness through experiments instead of the Hoare logic [8] which takes significantly more effort.

## 3 CASE STUDY

This section presents a concrete use case of the system. Let us say, for example, that two airlines, namely A and B, recently merged because A company bought B company to have two different ranges of flights. The two companies continue to operate separately, but want to share data. Furthermore, a third company C is specialized in statistics, notably about flights delays depending on departure time. Lately, C has been very interested in the delays of flights leaving Tokyo and then asks A to share its information, namely the departure city and time, and the delay. Obviously, as B is a part of A, all data of B should be included, making B also a member of the sharing group.

Answering the request, A airline has to share its own data presented in Figure 3. The company A therefore selects, using the interface described before, the "ID", "Fly_Dep", "Dep" and "Delay" columns, standing respectively for the flight number, the time of departure, the departure city and the delay, where the departure city is Tokyo. Also, this company does not wish to allow others to edit its data, setting the privileges to *read-only*. Given this input, the system consistently produces, using bidirectional transformations, a view containing the selected data to be shared. As company A's configuration file states that when a value is missing from the mapping table, it should be given by the user, A has then to fill in missing mappings. When it is done, the system uses a BiGUL function again to translate the local data into the universal format agreed and understood by everyone. The company finally agrees to push the data on the network.

When A is done sharing content, B then uses the system to produce a BiGUL view containing the data selected from its database: the "Name", "Dep_H", "Dept" and "Del", standing respectively for the same as A, where the departure city is Tokyo. B also sets the data to be *read-only*. Then, the data needs to be translated using BXs and B's mapping table containing the universal format. At this moment, the BiGUL function responsible for the translation checks the edition authorizations on the data to be pushed. As the "Name" *JAP36* refers to the universal *J42* which is already present in the shared data on *read-only* privilege, the bidirectional function refuses to edit the row. Indeed, it is possible to see in Figure 3 that company B has a delay of *5* while the shared data of company A has *Null* for row *J42*. So, the *JAP36* local row referring to *J42* is not shared, but the others, namely *CF2* and *DS9*, are.

Finally, C can retrieve the shared data when B has updated the shared content. All the data is then pulled from the network. When the system verifies C's mapping table, it auto-generates the unmatched local values, as the configuration files states. The system is then able to use this source of received universal format to get the translation into the local format, using the exact same bidirectional function provided to others like A or B. The local format is then computed in the put direction, using the bidirectional selection function, to be merged to the unselected content which is updated it if necessary. All of the content is then put back into either a new SQL table, or an existing one, depending on the user's choice.

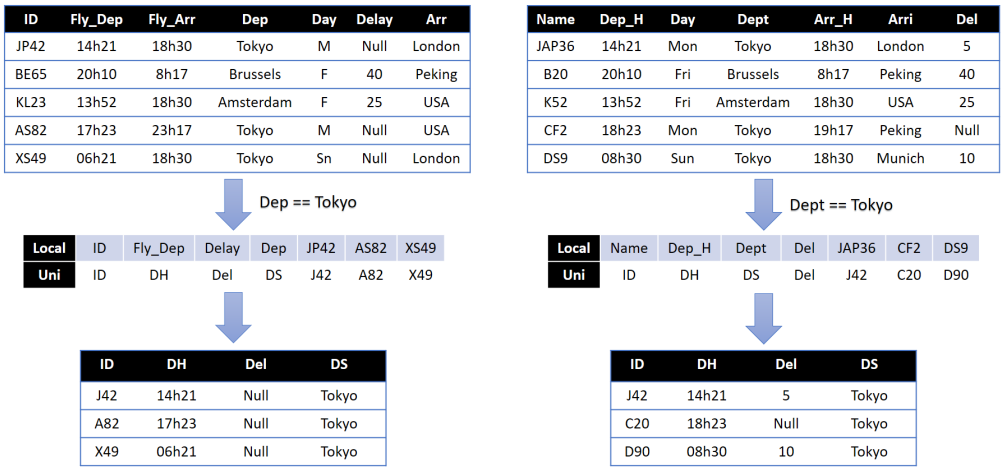


**Figure 3: The database of company A (left) and company B (right)**

## 4 CONCLUSION

This paper proposes a new user-friendly, lightweight and consistent system trying to solve the selective data sharing problem. Indeed, the system has been implemented in a way to be more easily handled thanks to the database knowledge limitation in both structure and code. This limitation is notably obtained by interacting with the user and the mapping tables provided to the bidirectional functions. Furthermore, the use of bidirectional transformations enhances the insurance of data consistency and integrity thanks to the proved well-behaveness of BiGUL. Last but not least, the user has the power to authorize or not others to edit its shared data, as BiGUL functions can easily deal with such properties while maintaining the well-behaveness of the system. It is then possible to conclude that the bidirectional transformations are here very useful and can be used properly and efficiently in the selective data sharing context.

The system could nevertheless be optimized notably by integrating the foreign keys management. The mapping table construction and the universal format could also be improved in order to be automatically generated in a meaningful way. Finally, the network part of the system could be strengthened in order to completely manage the keys representing the data, or could be replaced by a client-server architecture implemented following safety concerns to guarantee the availability of the data.

## ACKNOWLEDGMENTS

We would like to thank Hsiang-Shang "Josh" Ko and other members in the programming research laboratory in NII for helping us to understand bidirectional transformation and to use the BiGUL to develop our system. We would also like to thank the University of Namur for the internship opportunity they gave us. This work is partially supported by the Japan Society for the Promotion of Science (JSPS) Grant-in-Aid for Scientific Research (S) No. 17H06099, and by the Natural Science Foundation of China under the grants of No. 2015CB352200 and No. 61620106007.

## REFERENCES

[1] Oracle Corporation and/or its affiliates. 1995. MySql. https://dev.mysql.com/. (1995).
[2] Marcelo Arenas, Vasiliki Kantere, Anastasios Kementsietsidis, Iluju Kiringa, Renée J. Miller, and John Mylopoulos. 2003. *The Hyperion Project: From Data Integration to Data Coordination.* Dept. of Computer Science University of Toronto, School of Inf. Technology and Engineering University of Ottawa, Canada.
[3] Aaron Bohannon, Benjamin C. Pierce, and Jeffrey A. Vaughan. 2006. *Relational Lenses: A Language for Updatable Views.* University of Pennsylvania, Pennsylvania,United States.
[4] Adrien Duchene and Hugues Marchal. 2018. Lightweight Data Sharing System based on Bidirectional Transformations. https://github.com/AdrienDuchene/Bx_data_shared.git. (2018).
[5] J. Nathan Foster and Grigoris Karvournarakis. *Provenance and Data Synchronization.* University of Pennsylvania, United States.
[6] Zhenjiang Hu and Hsiang-Shang Ko. 2017. *Principles and Practice of Bidirectional Programming in BiGUL.* National Institute of Informatics, Japan.
[7] Yannis Katsis and Yannis Papakonstantinou. 2009. *View-based Data Integration.* Computer Science and Engineering UC San Diego, University of California-San Diego,La Jolla, CA,USA.
[8] Hsiang-Shang Ko and Zhenjiang Hu. 2018. *An Axiomatic Basis for Bidirectional Programming.* POPL 2018, Los Angeles, California, United States.
[9] Hsiang-Shang Ko, Tao Zan, and Zhenjiang Hu. 2016. *BiGUL: A Formally Verified Core Language for Putback-Based Bidirectional Programming.* PEPM 2016, St. Petersburg, Florida, United States.
[10] Czarnecki Krzysztof, Foster J. Nathan, Hu Zhenjiang, Lämmel Ralf, Schürr Andy, and Terwilliger James F. 2009. *Bidirectional Transformations: A Cross-Discipline Perspective.* Springer, Berlin, Heidelberg.
[11] Xinpeng Shen and Zhanhuai Li. 2008. *Implementing Database Management System in P2P Networks.* School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an, 710072, China.
[12] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. 2001. *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications.* MIT Laboratory for Computer Science, United States.