Synchronized Data Management and Its Integration into a Graphical User Interface for Archaeological Related Disciplines

Daniel Kaltenthaler[1], Johannes-Y. Lohrer[1](✉), Peer Kröger[1], Christiaan H. van der Meijden[2], and Henriette Obermaier[3]

[1] Institute for Informatics, Ludwig-Maximilians-Universität, Munich, Germany
{kaltenthaler,lohrer,kroeger}@dbs.ifi.lmu.de
[2] IT Group, Vetenarian Faculty, Ludwig-Maximilians-Universität, Munich, Germany
v.d.meijden@it.vetmed.uni-muenchen.de
[3] Bavarian State Collection for Anthropology and Palaeoanatomy, Munich, Germany
henriette.obermaier@palaeo.vetmed.uni-muenchen.de

Abstract. In this paper, we describe xBook, a generic, open-source e-Science infrastructure for distributed, relational data management that is particularly designed for the needs of archaeological related disciplines. The key feature of xBook is that it can be used as an offline resource at remote sites during excavations and can be synchronized with a central server at any time. While some scientists can record data in xBook in the field where no internet connection is available, colleagues can already work with and analyse the previously synchronized data via the central server at any location in the world. Incarnations of the xBook framework are used in archaeology, and archaeobiology (anthropology and archaeozoology). We will highlight one of them, OssoBook, an e-Science service that implements a data model for animal remains from archaeological sites (mainly bones) and has emerged as one of the European standards for archaeozoology.

1 Introduction

As in many other applications in archaeology a main part of the work comprises in collecting, sharing and analysing data. Often many researchers from different institutions and even varying countries are involved in excavation projects. Therefore entering data directly into databases is required to easily access data from different places and work simultaneously on recording as well as analysing the data. Archaeological data is often gathered in field work, i.e., at remote sites that do not offer a convenient environment for IT services, it is typically not possible to enter the data into databases that must be accessed via an internet connection. As a consequence, IT services are hardly used in these projects. Rather, data is typically recorded on paper and is (if at all) later processed electronically using proprietary and/or file-based data management tools like Excel, etc. for

© Springer International Publishing Switzerland 2015
A. Marcus (Ed.): DUXU 2015, Part II, LNCS 9187, pp. 317–329, 2015.
DOI: 10.1007/978-3-319-20898-5_31

doing simple descriptive statistics. This is significantly inconsistent with the need to sustainably store data on the cultural heritage claimed by the UNESCO[1].

Obviously, researchers from these archaeological domains would significantly benefit from a profound e-Science infrastructure that supports digital recording, implements sustainable data management and storage as well as offering powerful analysis tools. The key limitation of such an IT service is the problem of multiple users that need access to data recording and data analysis even if a permanent internet connection cannot be established. A synchronization process is required, implementing a client server architecture as visualized in Fig. 1, to ensure working offline at remote places, but also storing data globally, where it can be shared with other users is the solution.

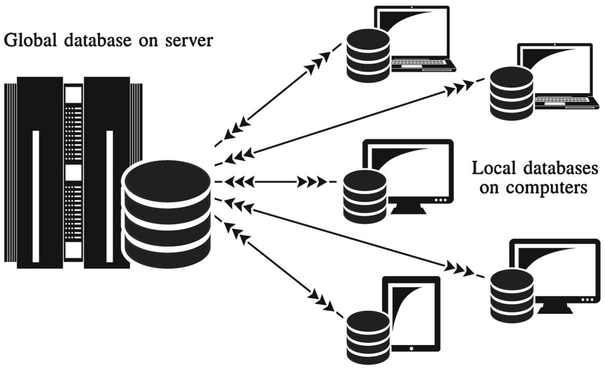


Fig. 1. The local clients are connected to the global server. The synchronization allows data exchange, so data can be recorded on the local machines, but can be backupped and shared via the server.

Existing commercial solutions for this problem are typically integrated in a dedicated database management system and/or cloud service. For licence or financial reasons as well as due to privacy concerns however, not all institutions can or want to resort to these systems. Budgets for archaeological excavation projects are typically optimized in terms of logistics and man-power. Reserving a considerable part for IT infrastructure is completely unrealistic. In addition, as long as the data is not yet analysed and the results are not yet published, the participating researchers are very wary about giving their data into the hands of commercial cloud services.

In this paper, we propose the framework xBook, a solution for the sketched problem that follows the architecture depicted in Fig. 1 is directly included into the application and, thus, can be used independently of the underlying database software. In particular, it can also be used with non-commercial and open source database management systems; in fact, xBook uses a MySQL database.

[1] http://www.unesco.org.

In addition, xBook implements a sophisticated privacy management. The solution can both be run as a server and can be installed on remote clients (e.g. laptop computers). Thus, each institution or consortium running an excavation can implement their own e-Science service without the need to give the data to a third, potentially not trusted party. Finally, the xBook framework is independent with respect to the data model. Thus, each research institution or consortium is free to implement its specialized data model reflecting different working paradigms, different ways of recording data, etc.

2 Problem Formulation

In this chapter, we discuss the requirements of an e-Science infrastructure for the archaeological sciences in more detail. A synchronized distributed e-Science infrastructure for data management and data analysis in the archaeological sciences should address the following issues:

- **Distinctability of Entries:** Two different entries must be distinct from each other, no matter on which local database they were created.
- **Conflict Handling:** Different users are able to work on the same data simultaneously on different local databases. The synchronization must recognize that a conflict occurred and provide options to solve it.
- **Time-Delayed Execution:** It cannot be guaranteed that a user of the database can always execute the synchronization process. This could be technical reasons like temporary internet disconnects, but also for logistic reasons that there is no internet connection available. It must be possible to continue working offline and synchronize the data later as soon as the computer is reconnected to an internet connection again.
- **Interruptible:** After a loss of connection or if the user interrupts the exchange of data, the synchronization process must be able to continue and no data may be lost or corrupted.
- **Modularity:** To avoid transfer of unnecessary data a selection of data is required. The user should be able to select parts of data he wants to synchronize to save time and to reduce data overhead, so data should be separated in modules ("projects"). Furthermore, the local database of the user should not save any data sets, for which he has no reading rights.
- **Currentness of Look-up Table Data:** The look-up table data, that is saved on the database of the global server, needs to be updated many times. This concerns regular as well as dynamic updates of this data. Administrators, with the permission to edit, must be able to insert new entries and edit or delete existing ones. These changes must be passed as soon as possible to the users, to ensure they can work with current data.
- **Rights Management:** Not every user needs to see all the data. The synchronization must check if the user, who has requested any data, has appropriate rights before sending or committing this data.

– **Easy To Use:** The synchronization should be an automatic process that can be run with few mouse clicks. The user is most likely not a skilled computer user, but must be able to use the synchronization. Therefore, the synchronization process should be carried out without any external data devices.

As discussed above, in addition to these specific requirements, an e-Science infrastructure for the archaeological sciences should also be cost effective and ensure the privacy of unpublished data.

In general, the basic idea of synchronization is not a new one. Both scientific and industrial databases often require functions to share data in any way, especially, if several users should be able to work on the data simultaneously. Synchronization techniques exist in the most commonly used database systems worldwide, including Microsoft SQL Server, Oracle and MySQL: Snapshot replication, Transaction replication, Replication with streams, Merge replication, Master-slave replication system.

While all of these methods are sufficient for many areas of application, to the best of our knowledge none of them cover all of the requirements stated above. Only few of the available solutions provide the ability to work offline and none of them allow synchronization of single datasets, which are two of the most important requirements for archaeologists. Therefore we created a synchronization that fulfils all previously stated requirements, is independent of the database management system and hence can be used for any database system.

3 The xBook Framework

With the aim of solving the same problems in several, different databases the idea of the xBook framework has emerged. The core function of xBook is the synchronization, but it also provides structures for the user management, right management, the graphical user interface and the data handling as indicated in Fig. 2. The databases are based on the same basic structure, but each of them can be individualised. This allows e.g. own input fields for entry mask or individual extensions. The function in detail:

User Rights Management: For archaeological data, it is not only important to allow parallel data entry of several persons, it is also necessary to be able to share data for complex analysis. Because not every user wants to make his data accessible to the public (especially if it is still unpublished), a flexible solution for the user rights management is required.

The framework provides default permissions (read, write and project management), each database can be extended with individual rights. As often is the case, to work together with the same group of people (e.g. within an institute) xBook does not only support the assignment of rights to single users, but also to user groups. These groups can be created by each user.

Graphical User Interface: xBook provides a graphical user interface because all databases that are based on the framework use the same functions and even the workflow is similar. This interface reuses common elements, but it also gives support to alter existing functions and to insert new, individual ones.

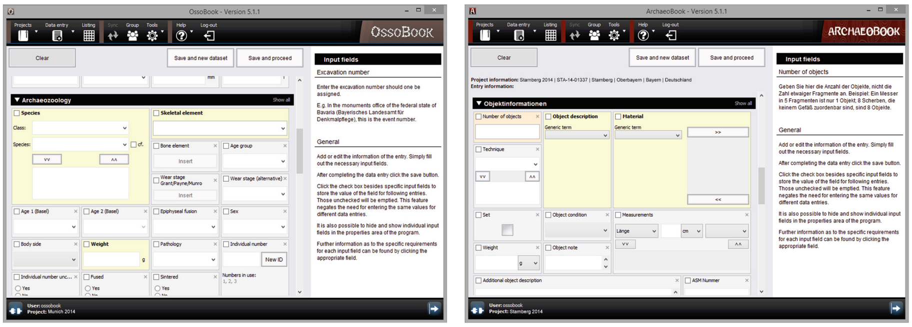
Especially the entry masks benefit from the framework a lot, because input fields, that were already implemented for one database, can be reused in other databases as well. Currently, xBook provides a wide variety of different input fields, e.g. text and number fields, dynamic comboboxes, selection fields, etc. Its advantage is that all databases profit by updates of a single input field. Customisations or individual, new fields can be integrated at any time. The similarity and the common elements in the user interfaces are illustrated in the screen shots depicted in Fig. 2.

Data Handling: The different databases very often use the same algorithms in the program execution. Especially for the huge amount of input fields, but also for single features, the underlying logic is repeating. To avoid multiple implementations in the code, xBook uses a combination of the single data managers (cf. Sect. 4) and the controller.

The controller connects the graphical user interface with the model and is responsible for the database-specific queries. It offers basic structures that are necessary for each database, but also allows customization if necessary.

Plug-in Interface (for data analysis): For the information processing of archaeological data the possibility of analyses is required. Different scientific research questions need different analyses, that cannot be predefined in xBook. So xBook contains a plug-in interface that allows the integration of analyses that can be added dynamically to the application. In general everyone who is familiar with coding in Java and working with SQL databases can create plug-ins for xBook. An API for the plug-in interface is in preparation. As default, a generic QBE plug-in for retrieval is implemented.

Synchronization: One of the core functions of xBook is the synchronization that is introduced in detail in Sect. 4.



Input fields

Excavation number

Enter the excavation number should one be assigned.

E.g. In the monuments office of the federal state of Bavaria (Bayerisches Landesamt für Denkmalpflege), this is the event number.

General

Add or edit the information of the entry. Simply fill out the necessary input fields.

After completing the data entry click the save button.

Click the check box besides specific input fields to store the value of the field for following entries. Those unchecked will be emptied. This feature negates the need for entering the same values for different data entries.

It is also possible to hide and show individual input fields in the properties area of the program.

Further information as to the specific requirements for each input field can be found by clicking the appropriate field.

Project information: Starnberg 2014 | STA-14-01337 | Starnberg | Oberbayern | Bayern | Deutschland
Entry information:

Input fields

Number of objects

Geben Sie hier die Anzahl der Objekte, nicht die Zahl etwaiger Fragmente an. Beispiel: Ein Messer in 5 Fragmenten ist nur 1 Objekt; 8 Scherben, die keinem Gefäß zuordenbar sind, sind 8 Objekte.

General

Add or edit the information of the entry. Simply fill out the necessary input fields.

After completing the data entry click the save button.

Click the check box besides specific input fields to store the value of the field for following entries. Those unchecked will be emptied. This feature negates the need for entering the same values for different data entries.

It is also possible to hide and show individual input fields in the properties area of the program.

Further information as to the specific requirements for each input field can be found by clicking the appropriate field.

Fig. 2. The input mask of OssoBook (left) and ArchaeoBook (right). Both of the applications are based on the xBook framework, that provides a basic graphical user interface and functions, but allow customisation like e.g. individual input fields.

4 Synchronization

4.1 Realization in the Database

To achieve the synchronization we are aiming for, some necessary additions in the database had to be made.

Database ID: One of the most important concepts is the database ID. A column for this ID is added in each table a user can enter data in. In addition this column is also marked as a primary key, to allow several distinct entries with the same ID from various databases. The value of this number for this database itself is stored both in a separate table and also as a property in the configuration settings of the operation system. The database ID is generated when the user connects to the server the first time, or if the value of the ID in the database is different to the value in the properties. This is to prevent errors, when a user copies his database to a different computer.

When the user enters a new entry in the database, the database ID is automatically filled in with the above defined value. If the entry is edited the database ID is not touched. Because we want to enable a modular approach with projects, all tables also must add the ID and the database ID of the project, otherwise the entry cannot be assigned to a project.

Status: To achieve both conflict management and identification which entries have to be updated locally the column "Status" is added to each table. It stores the time the entry was modified on the server. This value is modified only on the server via a trigger:

```
SET NEW.Status = NOW() + 0;
```

If the entry on the client has a lower (older) status it needs to be updated.

If the entry on the server has a higher (newer) status when committing data to the server, a conflict occurred, because someone else modified the entry.

Message ID: The next important addition is the column "message id". It stores the current status of the entry (locally). The different options are:

- **Synchronized (0):** Indicates that the entry is synchronized and no uncommitted changes were made. This does not mean that the entry is up-to-date, it just means the entry has no local changes and can be synchronized.
- **Changed (1):** Indicates that the entry was changed locally. This prevents the entry being updated with data from the database, to prevent overriding changes. In most cases this would also mean that a conflict occurs, but conflict checking is already done when the entry is committed, so no need to take action here. After the entry is successfully committed to the server, the status is set to "synchronized" again.

- **Conflicted (-1):** Indicates that this entry is conflicted and therefore can not be committed or updated. A conflict occurs when an entry that is changed locally has an older (lower) status than the entry on the server. Then the changes on the local entry were made on an older version of the entry, and if the entry would be committed, there would be possible loss of data.

4.2 Realization in the Application

Some of the basic requirements are already fulfilled in the database, but as the data has also to be sent and some features are still missing, several concepts had to be developed in the application as well.

Database Update: To allow constant updates to the application, including changes in the database scheme, we added a check for the version of the database. This number is stored in the "version" table and is compared to the built-in number in the program. If the numbers don't match, the server is queried for an update. This check is done after the connection to the database is established, but before the user can begin working with the data. If the user has no internet connection the update can of course not be made, but since he could also not upgrade the program without an internet connection, this doesn't seem like a big issue.

Code Tables: The look-up tables, which contain mappings for values in different languages e.g. the name of the animals, that are displayed in the graphical user interface, are named "code tables" in our synchronization. To be able to change or add values to the code tables without having to distribute a new program version, all values are stored in the database. To receive the newest version of the data only the database has to be updated. This can therefore be done during the usage of the application. To find out which values are changed, all tables have the column "Status". Just like the entry the value of the status is the timestamp of the last global change and is updated with triggers on insert and update:

```
SET NEW.Status = NOW() + 0;
```

The last update is set locally, after all changes have been (successfully) made, and is retrieved from the server before the update progress is started. Then only a check has to be made, if the local value is lower than the global value, and only if so, the code tables are out of date and therefore have to be updated.

Manager: To control the communication with the database from the client, we created manager classes, that have all the knowledge about the columns for the table they are "responsible" for. The structure of the manager is as follows:

- **Table Manager:** The base class for all managers. It holds the connection object, contains the basic methods like insert and update and sends SQL queries to the database.

- **Abstract Synchronization Manager:** This manager holds all the important information for the synchronization. It handles the insert and update of entries from the server, retrieves uncommitted entries and sets data to synchronized or conflicted.
  All managers that need to be synchronized must extend this.
- **Base Entry Manager:** The base entry manager is responsible for getting the main entry from the input unit (the main table for entries). It also calls the underlying *Extended Entry Managers* to retrieve the data for the complete entry. This class also manages the saving, loading and updating data for itself and forwards the call to the underlying methods.
- **Extended Entry Manager:** A manager for entries that extend a base entry, that is needed for example if an entry can have more than one value of a specific type. So the list of values would be stored in a different table.
- **Base Project Manager:** The base project manager is responsible for getting the main entry for entries that are valid for the whole project (e.g. project information itself). It also calls the underlying *Extended Project Managers* to retrieve the data for the complete entry. This class also manages the saving, loading and updating data for itself and forwards the call to the underlying methods.
- **Extended Project Manager:** This manager is for entries that extend a base project entry. This manager is needed for example if an entry can have more than one value of a specific type. So the list of values would be stored in a different table.

Data Structure: To store the data and retrieve a complete entry we created some classes to easily load, save and update the data.

- **DataColumn:** The most basic data type is the DataColumn. In it only one value is stored together with its column name.
- **DataRow:** Represents one row in the database. It is an ArrayList of *DataColumn* containing all the data for this row.
- **DataTable:** Contains all *DataRow*s for the current entry in the specific table. In addition to the *DataRow* it only knows to which table it belongs.
- **DataSet:** Represents one entry. Therefore it has all *DataTable*s that define the entry, and additionally hold the key of the entry and the key of the project the entry belongs to.

Synchronization Process: The actual progress of the synchronization consists of three different steps:

(1) Check if the user has the required rights to access the project and therefore is allowed to synchronize it.
(2) All uncommitted, not conflicted entries in project and entry tables are retrieved, one by one, from the database and send to the server, this is done by iterating over all *Base project* and *Base Entry Manager*s. These

call their belonging sub managers, load all data belonging to the current entry and send their data to the server. If the entry already exists in the global database, then the server checks the timestamp of the entry that was sent with the one of the entry that is already in the database. If the global timestamp is newer than the local one there is a conflict and the client is notified of it. (cf. Sect. 4.2)

(3) The project and entry data is transmitted from the server to the client. For identifying which entry has to be transmitted, the timestamp of the newest entry of the current table, therefore the last entry that was synchronized, is transmitted. To prevent loss of data after an incomplete synchronization, the first query requests entries that have exactly the same timestamp as the highest timestamp locally. For all later queries always the next data with a higher timestamp is retrieved. The entry is then only updated if the corresponding value in the local database either doesn't already exist or is not conflicted or changed.

Deletion: Due to the fact that the Synchronization can only identify changes with the check of the "Status" column, it is not easily possible to delete entries. Still, there needs to be the option to delete an entry. To solve this problem a column "Deleted" was added. It is an enumeration that has only two options: "Y" and "N" - with "N" as default value. Instead of deleting an entry the value of the "Deleted" column is set to "Y". Then this change can be synchronized to the global server. From there it can also be synchronized to other clients. When the client gets the information that an entry is deleted it can safely delete the entry locally. On the server however an entry is never deleted, because the information about the change must always remain available for the clients. The same logic is applied to code table entries, with the exception that entry tables are only synced to the clients.

Conflict Management: If an entry was marked as conflicted during the synchronization process, the conflict has to be solved before it can be merged with the entry in the global database. After the conflict has been merged e.g. by providing both the global and local entry, and allowing the user to select the diverse values, the merged entry is saved to the database with the timestamp of the global entry. This ensures, that if the entry was updated between the solving of the entry and committing the entry to the global database, this change will not be overridden, but a new conflict is generated.

5 Synchronization in the Graphical User Interface

As presented in Sect. 4 the synchronization consists of a powerful, but complex architecture. However the realization in the application must consider that most of the archaeologists are not used to work almost exclusively with a computer. That is the reason why it is absolutely necessary to hide the complexity of the synchronization behind an intuitive input mask that is easy to use and allows its usage even if the user is not technically versatile. Here we describe how the synchronization is integrated into the graphical user interface of xBook:

5.1 Manual Data Synchronization

To exchange project data there are three basic procedures that must be possible in the synchronization panel.

Global projects for which a user has read and/or write permission must be downloadable from the server. Therefore the corresponding projects can be selected in the right project selection in the synchronization panel.

Local projects that have not been synchronized with the server before must be able to be uploaded to the server. These projects can be selected in the left project selection.

Existing projects (as well local and global ones) must be updateable. For this purpose the corresponding projects must be selected, like explained above. However the application recognize if there was selected a project on the server project list that is also available on the local project list, and vice versa.

By pressing the "Synchronize" button the procedures are executed. Depended on the internet speed and the number of projects and datasets (e.g. in OssoBook exist projects with an six-digit number amount of entries) the synchronization may take several hours. Thus user feedback is displayed in message boxes and progress bars (each one for general, project and dataset layer)

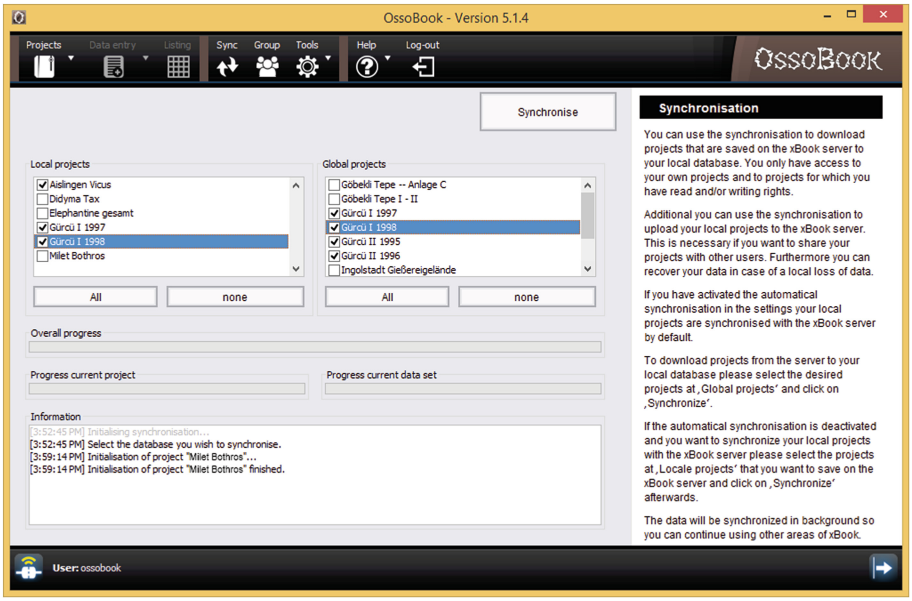
When the procedures are running the user can continue working with the application, the synchronization is running in background. It can also be interrupted by closing the application and continued to a later time.

5.2 Automatic Data Synchronization

The automatic synchronization can be activated in the application settings. Thereby the project information and data sets are synchronized with the server automatic in background. However it is necessary to manually define once which projects shall be downloaded from the server. This is important to avoid that all projects are downloaded even if the user does not want to save them on the local database (Fig. 3).

6 Case Study OssoBook - an Archaeozoological Database

OssoBook [8] was original released with dBASE as technical basis in 1990. Since then the application was continuously updated and extended [4], and different tools for data analysis were implemented and integrated to OssoBook as plugins [1,5–7]. In 2011 the database as well as the logic and the appearance of the application was restructured again to modernise the ageing program structure



Local projects:
- [✓] Aislingen Vicus
- [ ] Didyma Tax
- [ ] Elephantine gesamt
- [✓] Gürcü I 1997
- [✓] Gürcü I 1998
- [ ] Milet Bothros

Global projects:
- [ ] Göbekli Tepe -- Anlage C
- [ ] Göbekli Tepe I - II
- [✓] Gürcü I 1997
- [✓] Gürcü I 1998
- [✓] Gürcü II 1995
- [✓] Gürcü II 1996
- [ ] Ingolstadt Gießereigelände

```
[3:52:45 PM] Initialising synchronisation...
[3:52:45 PM] Select the database you wish to synchronise.
[3:59:14 PM] Initialisation of project "Milet Bothros"...
[3:59:14 PM] Initialisation of project "Milet Bothros" finished.
```

Synchronisation

You can use the synchronisation to download projects that are saved on the xBook server to your local database. You only have access to your own projects and to projects for which you have read and/or writing rights.

Additional you can use the synchronisation to upload your local projects to the xBook server. This is necessary if you want to share your projects with other users. Furthermore you can recover your data in case of a local loss of data.

If you have activated the automatical synchronisation in the settings your local projects are synchronised with the xBook server by default.

To download projects from the server to your local database please select the desired projects at ‚Global projects' and click on ‚Synchronize'.

If the automatical synchronisation is deactivated and you want to synchronize your local projects with the xBook server please select the projects at ‚Locale projects' that you want to save on the xBook server and click on ‚Synchronize' afterwards.

The data will be synchronized in background so you can continue using other areas of xBook.

Fig. 3. The synchronization panel in OssoBook.

and to prepare the software for future work [2,3]. Since then the application, especially the synchronization, is developing strongly.

OssoBook is currently used by approximately 200 users including scientists, Ph.D. students and students in institutes of universities, museums and scientific collections with archaeozoology as field of work, as well as freelance archaeozoologists. The eScience service is available in German, English, French and Spanish. In the context of the IANUS[2] project of the German Archaeological Institute, Berlin, Germany, OssoBook will serve as a standard for the archaeozoology domain. As introduction of the usage of the software there are annual workshops in several European countries.

OssoBook offers all the advantages of a database system. Furthermore, first analysis tools, that can be integrated into OssoBook as plug-ins are also available, including a module for the analysis of age distribution [1], a module for the cluster analysis of measurements [1], a plug-in for similarity search on multi-instance objects [5], a plug-in for the execution of sample data mining methods [6], and a module offering some analysis methods for archaeozoological data [7]. Subject-specific, the application should primarily allow the collection of the minimum standards, the input to these fields is mandatory. In addition there are numerousness of further possibilities to enter data. Many of them have user-friendly features (e.g. a few hundred pictures of all measurements for different animal classes), that will be extended step by step to fulfil the needs of the users. A text in the sidebar always explains the visible feature and includes details to the current selected input field.

2 http://www.ianus-fdz.de/.

7 Discussion

In this paper we described a list of requirements that should be fulfilled by e-Science infrastructures for a synchronized distributed data management and data analysis. We discussed existing solutions for synchronized distributed data management, in relation to the previously listed requirements. We then describe the xBook framework that meets the identified requirements. Then we took an in depth look at the synchronization process of xBook, which is independent of the data model and could be used potentially in any application domain if needed. Afterwards, we presented an existing incarnation of the xBook framework, called OssoBook, which has emerged as a standard for a growing European community and is already used in many archaeozoological projects.

While the synchronization has many benefits for the user and also someone that wants to add tables to the synchronization, there are also some limitations that still need to be addressed in the future, such as reducing the overhead for initialization, compression, and improved conflict handling.

8 Availability

The xBook framework on the web: http://xbook.vetmed.uni-muenchen.de/

References

1. Kaltenthaler, D., Lohrer, J.: Visual and density based cluster analysis of the archaeological database OssoBook in consideration of aspects of data integrity. Consistency and Quality, Diplomarbeit, Ludwig-Maximilians-Universität, Database Systems Group, Munich (2012)
2. Kaltenthaler, D.: Design and implementation of a graphical user interface for the archaeozoological database OssoBook. Projektarbeit, Ludwig-Maximilians-Universität, Database Systems Group, Munich (2011)
3. Lohrer, J.: Design and implementation of a dynamic database for archaeozoological tasks. Projektarbeit, Ludwig-Maximilians-Universität, Database Systems Group, Munich (2011)
4. Schiebler, J.: OSSOBOOK, a database system for archaeozoology. In: Anreiter, P., Bartosiewicz, L., Jerem, E., Meid, W. (eds.) Man an the Animal World: Studies in Archaeozoology, Archaeology, Antropology and Palaeolinguistics in Memoriam Sndor Bökönyi. Budapest, Archaeolongua (1998)
5. Danti, S.: Cluster analysis of features of animal bones and similarity search on multi instance objects of the archaeozoological data pool. Diplomarbeit, Ludwig-Maximilians-Universität, Database Systems Group, Munich (2010)
6. Tsukanava, Y.: Development and appliance of data mining methods on the palaeoanatomic data collection. Diplomarbeit, Ludwig-Maximilians-Universität, Database Systems Group, Munich (2010)
7. Neumayer, T.: Design and implementation of analysis methods for archaeozoological data. Bachelorarbeit, Ludwig-Maximilians-Universität, Database Systems Group, Munich (2012)

8. Kaltenthaler, D., Lohrer, J., Kröger, P., van der Meijden, C., Granado, E., Lamprecht, J., Nücke, F., Obermaier, H., Stopp, B., Baly, I., Callou, C., Gourichon, L., Pöllath, N., Peters, J., Schibler, J.: OssoBook v5.1.1. Munich, Basel (2014). (http://xbook.vetmed.uni-muenchen.de/)