An e-Science Infrastructure for Collecting, Sharing, Retrieving, and Analyzing Heterogeneous Scientific Data

Dissertation zur Erlangung des Doktorgrades
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

vorgelegt von

Johannes-Y. Lohrer

München, den 19.02.2018

Tag der Einreichung: 19.02.2018

Erstgutachter: Prof. Dr. Peer Kröger
Ludwig-Maximilians-Universität München
Institut für Informatik
Lehrstuhl für Datenbanksysteme und Data Mining

Zweitgutachter: Prof. Dr. Stefan Conrad
Heinrich Heine Universität Düsseldorf
Institut für Informatik
Datenbanken und Informationssysteme

Drittgutachter: Prof. Dr. Reinhard Förtsch
Universität zu Köln
Philosophische Fakultät
Archäologisches Institut

Vorsitz: Prof. Dr. Heinrich Hußmann
Ludwig-Maximilians-Universität München
Institut für Informatik
Arbeitsgruppen Medieninformatik und Mensch-Maschine-Interaktion

Tag der Disputation: 02.05.2018

Eidesstattliche Versicherung

(siehe Promotionsordnung vom 12.07.2011, § 8, Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

München, den 19.02.2018

Johannes-Y. Lohrer

Contents

Abstract ix

Zusammenfassung xi

1 Introduction 1
- 1.1 Overview . . . 3
- 1.2 Attribution . . . 4

2 Data Collection: Development of the xBook Framework 7
- 2.1 Introduction . . . 8
- 2.2 Origin of OssoBook . . . 9
  - 2.2.1 First OssoBook Version in dBASE . . . 9
  - 2.2.2 Conversion to Java . . . 10
  - 2.2.3 First Implementation of a Synchronization . . . 12
  - 2.2.4 Redevelopment of the Application . . . 12
- 2.3 From the Application OssoBook to the xBook Framework . . . 13
  - 2.3.1 Input Fields and Input Mask . . . 15
  - 2.3.2 Update Procedure . . . 15
  - 2.3.3 Plug-in Interface . . . 16
  - 2.3.4 Database Identification . . . 17
  - 2.3.5 Registration . . . 18
  - 2.3.6 Server-Client Architecture . . . 18
  - 2.3.7 Launcher . . . 22
- 2.4 Features of the xBook Framework . . . 27
  - 2.4.1 Synchronization . . . 27
  - 2.4.2 Graphical User Interface . . . 29
  - 2.4.3 Multiple and Crossed-Linked Input Masks . . . 30
  - 2.4.4 Listing and Export . . . 32
- 2.5 Applications Using the xBook Framework . . . 32

3 Sharing Data: A Timestamp-Based Synchronization Process 35
3.1 Introduction . . . 36
3.2 Problem Formulation . . . 39
3.3 Evaluation of Existing Synchronization Methods . . . 40
3.4 Synchronization . . . 43
3.4.1 Realization in the Database . . . 44
3.4.2 Realization in the Application . . . 46
3.5 Synchronization in the Graphical User Interface . . . 52
3.5.1 Manual Data Synchronization . . . 53
3.5.2 Automatic Data Synchronization . . . 54
3.5.3 Code Table Update . . . 54
3.5.4 Conflict Management . . . 54
3.6 Discussion . . . 56

4 Retrieving Data: Introducing the Reverse-Mediated Information System 59
4.1 Introduction . . . 60
4.2 Problem Formulation . . . 64
4.3 Requirements . . . 66
4.4 Reverse-Mediated Information System . . . 67
4.4.1 Initialization . . . 68
4.4.2 Registration . . . 69
4.4.3 Search . . . 72
4.4.4 Rights . . . 73
4.4.5 Configuration of the Connector Application . . . 75
4.5 Related Work and Comparison . . . 76
4.6 Case Study: Retrieving Archaeo-Related Information . . . 81

4.7 ReMIS Cloud . . . 83
4.7.1 Structure . . . 84
4.7.2 Data Retrieval . . . 84
4.7.3 Use Case: Scientific Example (Archaeology) . . . 86
4.7.4 Use Case: eLearning Example . . . 87
4.8 Conclusion and Discussion . . . 90

5 Analyzing Data: Tools for the Scientific Field of Application 93
5.1 An Embeddable Analysis Tool . . . 94
5.1.1 Requirements . . . 95
5.1.2 Existing methods . . . 97
5.1.3 Realization . . . 98
5.1.4 Integration . . . 102
5.1.5 Definition of Custom Workers . . . 104
5.1.6 Example of Provided Basic Workers . . . 106
5.1.7 Workflow . . . 109
5.1.8 Discussion . . . 113
5.2 A Visual Analytics Application for Temporal and Spatial Data . . . 115
5.2.1 Terminology . . . 116
5.2.2 Background and Related Work . . . 117
5.2.3 Illustration of the Spatial and Temporal Distribution of Findings . . . 119
5.2.4 Case Study: Distribution of Faunal Remains . . . 127
5.2.5 Conclusion and Discussion . . . 130

6 Conclusion and Discussion 133

References 137

Own Publications 147

List of Figures 151

List of Tables 155

List of Algorithms 157

Abstract

The process of collecting, sharing, retrieving, and analyzing data is common in many areas of scientific work. While each field has its own workflows and best practices, the general process can be aided by an e-Science infrastructure. The contribution of this thesis is to support the workflow of the scientists which can be split in four parts:

In the first part, we introduce xBook, a framework which aids the creation of database application to collect, back-up, and share data.

In the second part, we describe the synchronization which is a vital part of the xBook framework that, with the use of timestamps, allows data to be entered offline. The data then can be shared with coworkers for analyses or further processing. It also can be used as a backup system to avoid data loss.

Third, we present an architecture allowing data from distributed data sources to be retrieved without a central managing instance. This is achieved with the use of minimal search parameters which are guaranteed to exist in all connected data sources. This architecture is based on the concept of mediators, but gives data owners full control over their data sources as opposed to the traditional mediator where the connected data sources are managed by a central administrator.

Fourth we describe an embeddable analysis tool which can be integrated into a base application where the data is gathered. With the aid of simple modules, called “Workers”, this tool empowers domain experts to easily create analyses particularly designed for their area of work in a familiar working environment. Additionally, we present another tool which allows the graphical display of temporal and spatial information of archaeological excavations. This tool uses an interactive Harris Matrix to order findings temporally and allows the comparison with their spatial location.

Zusammenfassung

Viele wissenschaftliche Bereiche haben gemeinsame Vorgänge, wie die Erfassung, das Teilen, das Abrufen und das Analysieren von Daten. Jeder Bereich hat zwar seine eigenen Vorgänge und bewährte Verfahren, jedoch kann der allgemeine Prozess durch eine e-Science Infrastruktur unterstützt werden. Der Beitrag dieser Dissertation ist die Unterstützung des typischen wissenschaftlichen Arbeitsablaufes, der in vier Teile unterteilt werden kann:

Im ersten Teil stellen wir xBook vor, ein Framework zur Erstellung von Datenbankanwendungen, das Wissenschaftler dabei unterstützt, Daten zu erfassen, zu sichern und zu teilen.

Im zweiten Teil beschreiben wir die Synchronisation, die ein wichtiger Teil des xBook Frameworks ist. Diese erlaubt, dass Daten offline bearbeitet werden können, indem Änderungen über Zeitstempel protokolliert werden. Diese Daten können dann mit Kollegen für Analysen oder weitere Eingaben geteilt werden. Die Synchronisation kann zusätzlich als Sicherungssystem verwendet werden, um Datenverlust zu verhindern.

Im dritten Teil präsentieren wir eine Architektur, um Daten aus verteilten Datenquellen ohne ein zentrales Verwaltungssystem abrufen zu können. Dies wird mit Hilfe eines minimalen Suchparameters, der in allen angeschlossenen Datenquellen existieren muss, ermöglicht. Diese Architektur basiert auf dem Konzept des Mediators, benötigt aber im Gegensatz zum traditionellen Mediator keinen zentralen Administrator zur Verwaltung der Datenquellen und gibt deren Besitzern volle Kontrolle über ihre Daten.

Abschließend, im vierten Teil, beschreiben wir ein einbettbares Analyse Tool, das in eine Hauptanwendung integriert werden kann, in der Daten erfasst werden. Dieses Tool ermöglicht Fachexperten auf einfache Weise, mit Hilfe von speziellen Modulen, Analysen in einer vertrauten Arbeitsumgebung zu erstellen, die genau für ihr Fachgebiet benötigt werden. Zusätzlich stellen wir ein weiteres Tool vor, das die zeitlichen und räumlichen Informationen archäologischer Ausgrabungen visualisiert. Dieses Tool verwendet eine interaktive Harris Matrix, um Funde zeitlich zu ordnen und erlaubt den Vergleich ihrer räumlichen Position.

Chapter 1

Introduction

Attribution

This chapter does not use any material from previous publications.

In many scientific disciplines the gathering and recording of data of different material and immaterial objects is an integral part of the daily work. Examples range from archaeology and bioarchaeology, where human or animal remains are determined and the results saved, to biology, where for example data of birds is collected, to medicine, where information of patients are collected. All these disciplines rely on data they gather to make analyses which improve the understanding of the past, presence, and the future and help to understand the nature of things.

Each discipline uses the collected information to answer different scientific questions. However, they all have in common that the basic workflow is similar in the different domains and sub-domains. At the very first, the data is always collected and stored preferably in a relational database, e.g. in field work, in a lab, or in a clinic. For this the data is collected as exact and detailed as possible and usually standardized. Then, this data serves as the basis for later scientific analyses. Often a collaboration is useful and necessary to split workload or to separate different subfields during the data collection. Then, the data can be shared with coworkers to allow different scientists to carry out different analyses or use a broader basis of information for their analyses. This requires to consider privacy information, so that not everybody is allowed to see all data and also not everybody who has access to the data can edit it. Additionally, it might be interesting for scientists to take into account data which has a specific attribute in common. Therefore, they search for data from other available data sources which they can consider in their analyses. Each of these work steps contributes to the data analyses which usually are the purpose of scientific work. A sketch of the workflow of scientific work can be seen in Figure 1.1.

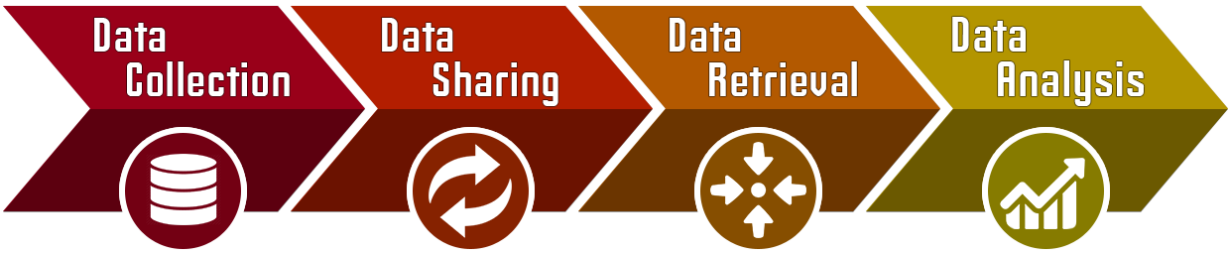


Figure 1.1: *A visualization of the workflow of scientific work.*

Many different domains could benefit from a digital infrastructure. Especially seasoned scientists still use analog methods or basic tools like spreadsheet applications to store their information. But, not only habits are an obstacle for digital applications. Often the lack of funds or technical know-how also hinder the development of tools which support the scientists in their work. This makes sharing, retrieving, and analyzing data difficult and time-consuming.

However, the digitization is becoming more and more important. Asked about the requirements of a database for archaeological and bioarchaeological excavations, the participants of the workshop "Digitale Grabungsdokumentation – objektiv und nachhaltig"[1] of the *Verband der Landesarchäologen*[2] stated that data security, working in parallel, and available plausibility and error checks are the most important topics [Gö18]. While data security is a big challenge for a digital infrastructure, plausibility and error checks are not possible if working analogously.

In biology, the demand to compare a larger amount of biological specimens was identified. The digitization "boost[s] the impact of collections to research and society through improved access" [BC12].

In other areas the digitization is also becoming a priority. For the "Masterplan Bayern Digital" [Sta14], the *Bayerische Staatsregierung*[3] plans to invest additional three billion Euros for the digitization until the end of 2022. While in this case, the government support is aimed for security, traffic, environmental protection, and education, it still underscores the importance of the topic and that it will play an even more important role in future.

Additionally, the digital collection of data also provides new research possibilities, e.g. the nature conservation research benefits from the collection of tracking data digitally Cuckoos are equipped with satellite transmitters which track the position information of the birds during their whole annual cycle. This method is applied for a few years to

[1] `http://www.landesarchaeologen.de/verband/kommissionen/archaeologie-und-informationssysteme/projektearbeitsgruppen/workshop-digitale-grabungsdokumentation`

[2] `http://www.landesarchaeologen.de`

[3] `http://www.bayern.de`

research the reasons of the population decline of cuckoos in Bavaria and Belarus and to develop conservation measures after having obtained the knowledge about the migration threats. [Her14] Therefore, areas in which the digitization is not common yet should feel encouraged to start using methods to digitally collect data, to be able to increase the research possibilities.

An e-Science infrastructure which is adapted to the needs of the specific field could greatly improve the willingness to start working with digital methods. It also increases the possibility that new and improved analyses are carried out which are faster and more meaningful. Additionally, the infrastructure can help to reduce entering incorrect data during the collection. A general framework would provide an option to allow the creation of the required tools with minimal financial expenditure. We hope to contribute to support different scientific disciplines with the methods provided in this thesis.

1.1 Overview

In this thesis proposes a scientific infrastructure which aids scientist in different scientific domains in collecting, sharing, retrieving, and analyzing their data. The remainder of this thesis is structured as follows:

In Chapter 2, we give a short overview of the historic development of XBOOK, a framework which enables the creation of databases for the gathering and sharing of scientific data sets. XBOOK offers a variety of input elements that can easily be arranged in order to provide a graphical user interface that the user can use to enter data in a user-friendly way. Possible manifestations of an entry can be entered with predefined values, reducing risks of incorrect data input. XBOOK also offers a rights management which allows users to exactly specify to which users specific data sets shall be made accessible.

Then, in Chapter 3, we explain the synchronization, a vital part of the XBOOK framework in more detail. The synchronization allows the sharing of data sets with other scientists as well as the possibility to back-up the data. Using timestamps the synchronization keeps track on changes done to the data sets while also providing a way to detect conflicts that can occur if several scientists change the same data set at the same time.

Third in Chapter 4, we introduce a new architecture called Reverse-Mediated Information System (REMIS) which allows the retrieval of distributed data sets from anonymous data sources. This architecture is built upon the idea of the Mediator concept. A mapping is created for each data source to allow the retrieval of data for predefined search parameters. But instead of using one central managing instance, this architecture hands the control of the data sources to the data owners, allowing them to freely connect and disconnect from the network and to choose the data sets they are willing to share.

Then, in Chapter 5, we describe an ANALYSIS TOOL, that can be embedded into a

base application. The tool aims to be used inside a familiar working environment of the scientists to encourage them not only to use their default analyses, but also to create new analyses which might lead to new knowledge. The tool allows the creation of complex analyses with the use of simple independent modules which can result in a variety of visual representations displaying the results. The ANALYSIS TOOL also allows the creation and integration of new modules which might be necessary for field specific analyses that cannot be carried out with the modules available by default. We also present TARDIS, an individual analysis which is aimed for archaeological excavations to graphically visualize both the spatial and the temporal history of different layers of an excavation considering the findings information within them. TARDIS uses a Harris Matrix to display the temporal order of different layers and also allows to filter for different attributes of findings inside these layers.

Finally, in Chapter 6, we discuss the current state of the infrastructure and how the infrastructure can benefit from future research.

1.2 Attribution

This chapter gives an overview of the previously published papers that are included in this thesis (in order of appearance in this work) and clarifies the contributions of the author of this thesis.

Chapter 2 is based on the publication *The Historic Development of the Zooarchaeological Database OssoBook and the xBook framework* [KL18] by Daniel Kaltenthaler and Johannes-Y. Lohrer, which describes the development of the database OSSOBOOK and the framework XBOOK. The concept, implementation, and design of the XBOOK framework was solely realized by Johannes-Y. Lohrer and Daniel Kaltenthaler, including the technical and collaborative infrastructure, the software development, and the draft and implementation of the included features. The mentioned applications OSSOBOOK [KLK+18a] (since the new development from version 4.0), EXCABOOK [KLK+18e], ANTHRODEPOT [KLK+18b], and PALAEODEPOT [KLK+18f] are being developed exclusively by Johannes-Y. Lohrer and Daniel Kaltenthaler. ARCHAEOBOOK [KLK+18d] was originally developed by Johannes-Y. Lohrer and Daniel Kaltenthaler, and later Ciarán Harrington supported the implementation. ANTHROBOOK [KLK+18g] was developed by Tatiana Sizova and Anja Mösch on the basis of the XBOOK framework until 2016. Then Johannes-Y. Lohrer and Daniel Kaltenthaler took over the development in 2017. INBOOK [KLK+18c] is being developed by Ciarán Harrington. The database applications consider the valuable domain expertise of primarily Henriette Obermaier (OSSOBOOK and PALAEODEPOT), Christiaan van der Meijden (OSSOBOOK), Sonja Marzinzik, Erich Claßen, and Heiner Schwarzberg (each ARCHAEOBOOK), Michaela Harbeck and Andrea Grigat (each ANTHRODEPOT

and ANTHROBOOK), Anita Toncala (ANTHROBOOK), and Silke Jantos, Agnes Rahm, Ina Sassen, Tilman Wanke, and Roland Wanninger (each EXCABOOK).

Chapter 3 is based on the publication *A Generic Framework for Synchronized Distributed Data Management in Archaeological Related Disciplines* [LKK+14] by Johannes-Y. Lohrer, Daniel Kaltenthaler, Peer Kröger, Christiaan van der Meijden, and Henriette Obermaier, which describes a timestamp-based synchronization method to enable sharing of data and collaboration with colleagues. The synchronization logic was primarily drafted and implemented by Johannes-Y. Lohrer in cooperation with Daniel Kaltenthaler. A follow-up publication of the same title [LKK+16b] by the same authors additionally explains the conflict management and the integration into the graphical user interface in more detail, which was primarily drafted, designed, and implemented by Daniel Kaltenthaler in cooperation with Johannes-Y. Lohrer. The work was developed under supervision of Peer Kröger and Christiaan van der Meijden. Henriette Obermaier contributed domain-specific input for the two mentioned papers.

Chapter 4 is based on the publications *Reverse-Mediated Information System: Web-based Retrieval of Distributed, Anonymous Information* [LKK+17] by Johannes-Y. Lohrer, Daniel Kaltenthaler, Peer Kröger, and Christiaan van der Meijden and *Retrieval of Heterogeneous Data from Dynamic and Anonymous Sources* [LKR+18] by Johannes-Y. Lohrer, Daniel Kaltenthaler, Florian Richter, Tatiana Sizova, Peer Kröger, and Christiaan van der Meijden, which describe an architecture to retrieve data from heterogeneous, distributed, and anonymous data sources. The architecture was primarily drafted by Johannes-Y. Lohrer and implemented by Johannes-Y. Lohrer, Daniel Kaltenthaler, and Tatiana Sizova, under supervision of Peer Kröger. Florian Richter attributed valuable input about the usage of the system in other domains. Christiaan van der Meijden contributed with valuable discussions. Henriette Obermaier contributed domain-specific input.

Chapter 4.7 is based on the publications *A Distributed Information Management System for Interdisciplinary Knowledge Linkage* [KLRK17] and the extended publication *Interdisciplinary Knowledge Cohesion through Distributed Information Management Systems* [KLRK18] by Daniel Kaltenthaler, Johannes-Y. Lohrer, Florian Richter, and Peer Kröger, which describe an information system that enables the data retrieval from interdisciplinary domains. The architecture was primarily drafted by Daniel Kaltenthaler in cooperation with Johannes-Y. Lohrer under supervision of Peer Kröger. The implementation was realized by Johannes-Y. Lohrer and Daniel Kaltenthaler. Florian Richter contributed the use case for the eLearning context and the discussion about ethics.

The Chapter 5.1 is based on the publication *Leveraging Data Analysis for Domain Experts: An Embeddable Framework for Basic Data Science Tasks* [LKK16a] by Johannes-Y. Lohrer, Daniel Kaltenthaler, and Peer Kröger, which describes an ANALYSIS TOOL that can be embedded into other applications to enable data analyses without the need of programming skills. The described logic of the tool was primarily drafted and implemented

by Johannes-Y. Lohrer under supervision of Peer Kröger. Daniel Kaltenthaler attributed valuable input towards the tool.

Chapter 5.2 is based on the publication *TaRDIS, a Visual Analytics System for Spatial and Temporal Data in Archaeo-related Disciplines* [KLP+17] by Daniel Kaltenthaler, Johannes-Y. Lohrer, Ptolemaios Paxinos, Daniel Hämmerle, Henriette Obermaier, and Peer Kröger, which introduces a visual analysis application to put spatial information in relation with temporal data from archaeological sites. The concept of the tool was drafted by Johannes-Y. Lohrer and Daniel Kaltenthaler under supervision of Peer Kröger. The implementation of the presented prototype was developed by Johannes-Y. Lohrer, Daniel Kaltenthaler, and Daniel Hämmerle. Henriette Obermaier contributed domain-specific input. The executed analysis was interpreted by the domain scientist Ptolemaios Paxinos.

Chapter 2

Data Collection: Development of the xBook Framework

Attribution

This chapter uses material from the following publication:

- Daniel Kaltenthaler and Johannes-Y. Lohrer. The Historic Development of the Zooarchaeological Database OssoBook and the xBook Framework for Scientific Databases. *ArXiv e-prints: 1801.08052*, January 2018 [KL18]

See Chapter 1.2 for a detailed overview of incorporated publications.

In many scientific domains, collecting of data is the very first step (c.f. Figure 2.1). It builds the basis for future analyses and is especially important for long-term archiving. Especially a detailed data collection is important for sciences that deal with material things which are collected and stored in e.g. collections or museums. Most often, the origin situation of the place where the object was found is destructed after the collection and cannot be restored which makes a digital backup important. In other cases, it is in the nature of things that the initial situation cannot be restored. For example, biological observations of animal locations, medical information about patients, chemical samples in laboratories, etc. All this data require to be accessible so that it can be used in analyses in the future.

For all these cases, usually the gathering method and data that scientists gather is similar in their field. Therefore, a standardized program to store the data could be beneficial. In this chapter, we introduce XBOOK, a framework aimed to allow the creation of database applications to support scientists to collect data, and additionally provide the option to share the data.

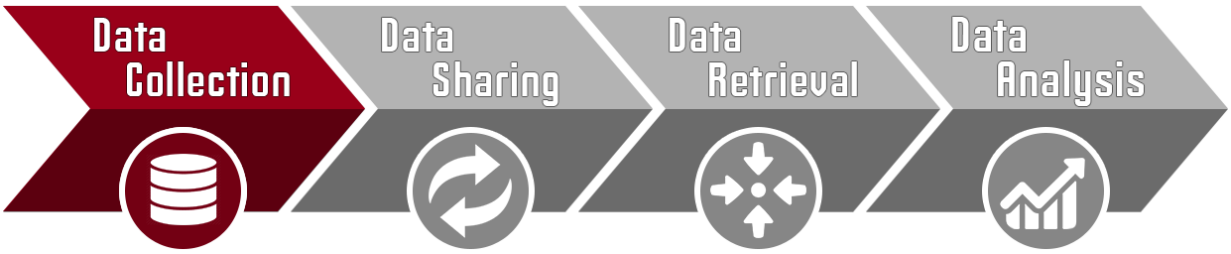


Figure 2.1: *Data collection is usually the first step in any scientific workflow.*

We first give an overview of the historic evolution from the zooarchaeological database OssoBook [KLK+18a] to the xBook framework and then introduce the different features of xBook.

2.1 Introduction

In general, software and its possibilities are developing to an ever more advanced level. The implementations are changing over time and new technologies need to be considered and integrated. Different ideas and concepts of developers, and different expectations of customers have to be taken into account when developing the application. Even though these approaches may differ from expected realizations, especially in the range of data gathering, the requirements are quite similar in several scientific areas.

It is often the case that different disciplines develop their own software solutions to gather and manage own data specifically for their own need. However, this causes the decisive disadvantage that new features which can be applied to several disciplines have to be implemented multiple times for each of the individual solutions. This is not only a huge temporal, but also a financial expenditure.

This situation became especially apparent during the development of OssoBook, a database for zooarchaeological findings. Other disciplines in the archaeological context also gather data with similar methods. Of course, these differ content-related, but the requirements on the basic features strongly overlap with the functionalities of OssoBook. It is not limited to the archaeological context but other scientific disciplines could also benefit from similar solutions.

In other archaeo-related disciplines (like archaeology, anthropology, archaeobotany, etc.) there is also a necessity of gathering data which consists of similar workflows like in the zooarchaeological field. In consequence, the idea of the xBook framework developed out of this context. In the xBook framework features are developed centrally and are provided to all applications that are instances of the framework. New features do not have to be implemented individually, however, custom extensions are still possible.

In case of errors and bugs, it is not necessary to fix them in each application, they can be fixed centrally. Thereby, it causes the creation of similar structures in the gathering and analysis of data. As of now, the xBOOK framework enabled the development and usage of a number of archaeo-related applications, like OSSOBOOK, ARCHAEOBOOK [KLK+18d], ANTHROBOOK [KLK+18g], EXCABOOK [KLK+18e], PALAEODEPOT [KLK+18f], ANTHRODEPOT [KLK+18b], and INBOOK [KLK+18c] (cf. Chapter 2.5). However, the xBOOK framework is not limited to the requirements of the archaeo-related context. It can be applied to many other scientific applications as well.

In this chapter we describe the development process of the OSSOBOOK application and the xBOOK framework in the recent decades. We first describe the origins of OSSOBOOK in Chapter 2.2 and explain the further development of the application and the extraction of the xBOOK framework in Chapter 2.3. Finally we show the basic, most important features of xBOOK in Chapter 2.4.

2.2 Origin of OssoBook

Below, we describe the original development of the OSSOBOOK application since 1990.

2.2.1 First OssoBook Version in dBASE

The first version of OSSOBOOK was originally released in 1990 by Jörg Schibler and Dieter Kubli of the University of Basel, Switzerland. The technical basis was dBASE[4], a file based database application for computer systems running the operating system DOS. The exclusively in German language published OSSOBOOK database enabled the recording of zooarchaeological data. Five input fields for archaeological information, eleven input fields for zooarchaeological data, and for each skeleton element eight further input fields for the recording of measurements were provided (cf. Figure 2.2).

Besides the data input, the early version of the application already provided the possibility to implement simple data analyses. Three analyses were available for skeleton element representations: One for the age of long bones (cf. Figure 2.3), one set of analyses for bone parts, and one last analysis for measurements of bones. The results of these analyses could be saved to extern files which could be viewed and edited with spreadsheet like Microsoft Excel or Open Office Calc (today: LibreOffice Calc).

Besides the database itself, this version also provided an editor called OSSOINST which allowed defining custom numerical codes for different data inputs, e.g. the mapping of a species ID to a species name. This offered the advantage that each database

[4] dBASE's underlying file format, the .dbf file, is widely used in applications needing a simple format to store structured data.

Figure 2.2: *The input mask in* OSSOBOOK *1.0. [Loh12, Kal12]*

could be used with individual data, and each user could easily extend the list of available species.

The archaeological data and the corresponding mappings were saved as ASCII files on the local hard disc drive of a computer, or on floppy disks. This made the sharing of data complicated and difficult, and therefore was rarely practiced. Nevertheless, OSSOBOOK already allowed the translation of specific texts of input values to other languages by using OSSOINST. First partial translations (especially to English and French) were already possible and done. [Sch98]

2.2.2 Conversion to Java

At the latest with the release of Microsoft Windows XP in 2001, the operation system DOS fades into the background. The technique of the dBASE database became increasingly obsolete. Even though the application is still running on modern operation systems (e.g. in the command-line interface on Windows computers, or the system console in other operation systems), the usability and optical presentation of OSSOBOOK was no longer up-to-date. The disadvantage that data could only be saved on the local computers, also contributes that a new, enhanced version of OSSOBOOK should be developed.

The new version of OSSOBOOK was initialized by Christiaan H. van der Meijden of the *Veterinary Faculty*[5], together with the *Institut für Informatik*, *Lehrstuhl für Datenbanksys-*

[5] `http://www.vetmed.uni-muenchen.de`

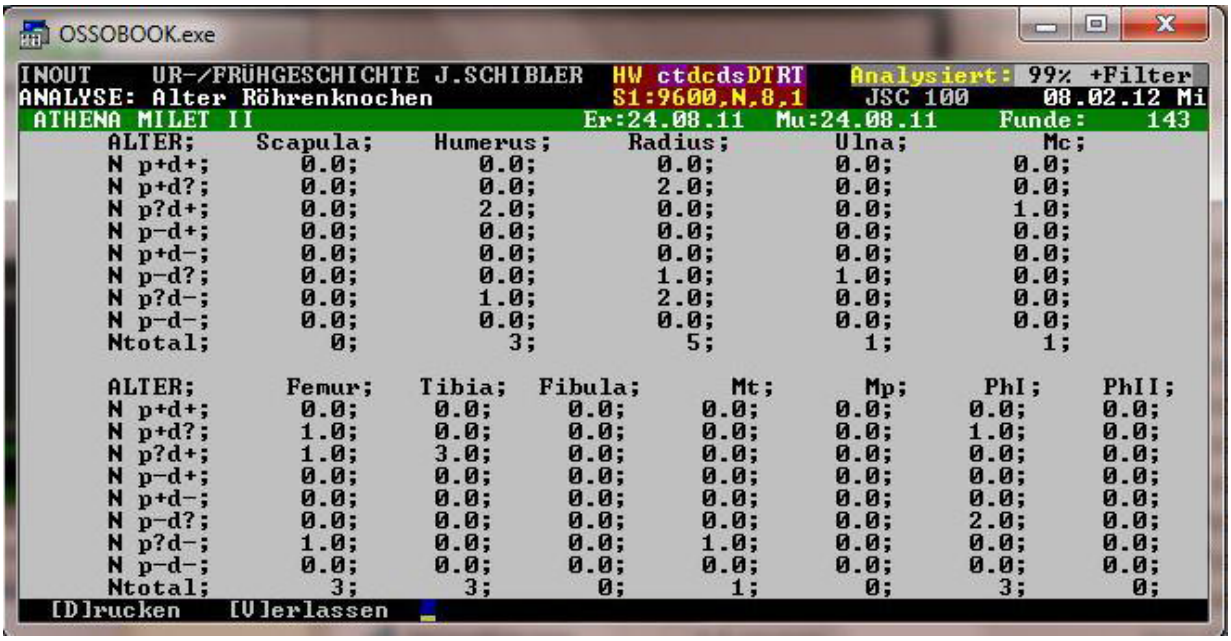


Figure 2.3: *The analysis of the age of long bones in* OssoBook *1.0. [Loh12, Kal12]*

teme and Data Mining[6] at the Ludwig-Maximilians-Universität München, Germany. The application was converted to the object-oriented and platform-independent programming language Java. On the servers of the university, there was installed a single, global MySQL database which should be used by the employees at the *Institute of Palaeoanatomy, Domestication Research and History of Veterinary Medicine*[7]. They used the client to connect to the server and directly work with the data of OssoBook on the global database. [Lam08]

In addition, the mapping of IDs to values for specific fields was adopted, but the functionality to add or change them manually was removed. The users worked with standardized, predefined values for the necessary input fields. This should improve the comparability of the entries that were saved in the database. Today, these mappings are called “Code Tables” in the application.

In combination with the port of the application to Java, OssoBook got a graphical user interface for the first time. As shown in Figure 2.4, the input fields were arranged in four sections and offered first input assistances, e.g. by using selection boxes for predefined values. Statistical information about the data sets and simple analyses were displayed in several tabs, which also provided more space for further input possibilities.

Version 3.4 was the first Java version of OssoBook and was released in 2007.

[6] http://www.dbs.ifi.lmu.de
[7] http://www.palaeo.vetmed.uni-muenchen.de

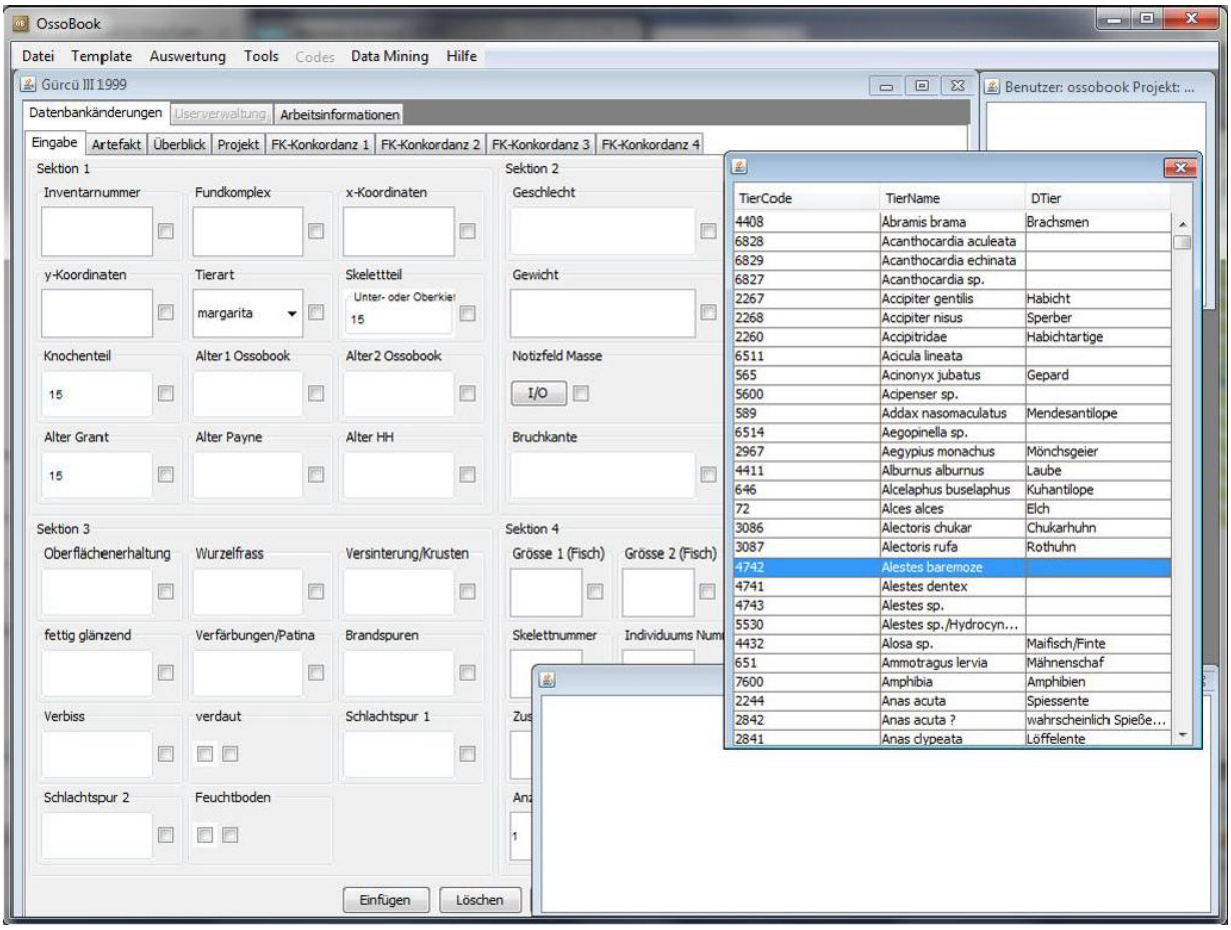


Figure 2.4: *The input mask in* OssoBook *3.4. [Loh12, Kal12]*

2.2.3 First Implementation of a Synchronization

Until this date, the users of OssoBook could only connect and work directly on the global database on the servers. Originally this was only possible with connections within the network of the university, later a tunnel enabled the connection from other places.

In 2008, the first implementation of a synchronization in OssoBook allowed the entry of data in a local database of the clients. The users could enter their data remotely without any connection to the network and later synchronize it to the global database.

The implementation and the full development of the synchronization is described in detail in Chapter 2.4.1.

2.2.4 Redevelopment of the Application

At this time, we were tasked with the further development of OssoBook and became the new development team for the upcoming years. However, this Java version of OssoBook

caused big problems for the development and usage. One could see that the application was only a port and did not use the potential of the object-orientated programming language Java. Most of the input fields were not very intuitive because of working with numerical codes in general, which meaning had to be looked-up in external spreadsheet or PDF files. In addition, the application included a lot of errors and bugs which were not displayed in the graphical user interface of the application. In some cases these problems made the data input impossible and let the application crash.

From the point of view of the development team it was nearly impossible to implement the requests of the zooarchaeologists and to add new features. The first tries to integrate new elements into the application made already clear that it is more reasonable to re-implement the application from scratch instead of trying to continue developing for the current version. The main problem of a further development of the current version was the very static program code elements that did not allow adding new input elements to the input mask. This static design also made an object-oriented design using inheritance impossible. Also the programming code was only sparsely commented and Javadoc comments were missing for the most parts. In addition, the names of the methods were not intuitive, so new developers would need a long familiarization to be able to develop the application.

It was decided that the database scheme of OssoBook and the OssoBook client should be newly developed considering the Model–View–Controller architecture [Loh11, Kal11]. We put a lot of emphasis for future features being able to be fast and dynamically integrated to the application to enable a simple and resource-efficient further development. To avoid data loss and to be able to continue using the data of the previous OssoBook version, we wrote a script that converts the old database scheme into the new one.

In autumn 2011, version 4.1 of OssoBook was released that included the same features than the previous version of the database application at first, but was more flexible in the usage for the users and developers. A screenshot of this version can be seen in Figure 2.5.

2.3 From the Application OssoBook to the xBook Framework

From this point of time, OssoBook was ready for scholarly usage. At the same time, it was assured that further development and future adjustments to the database are possible.

Since the gathering of data is an essential task of the work in archaeo-related disciplines, other disciplines got also interested in the architecture of OssoBook. While the

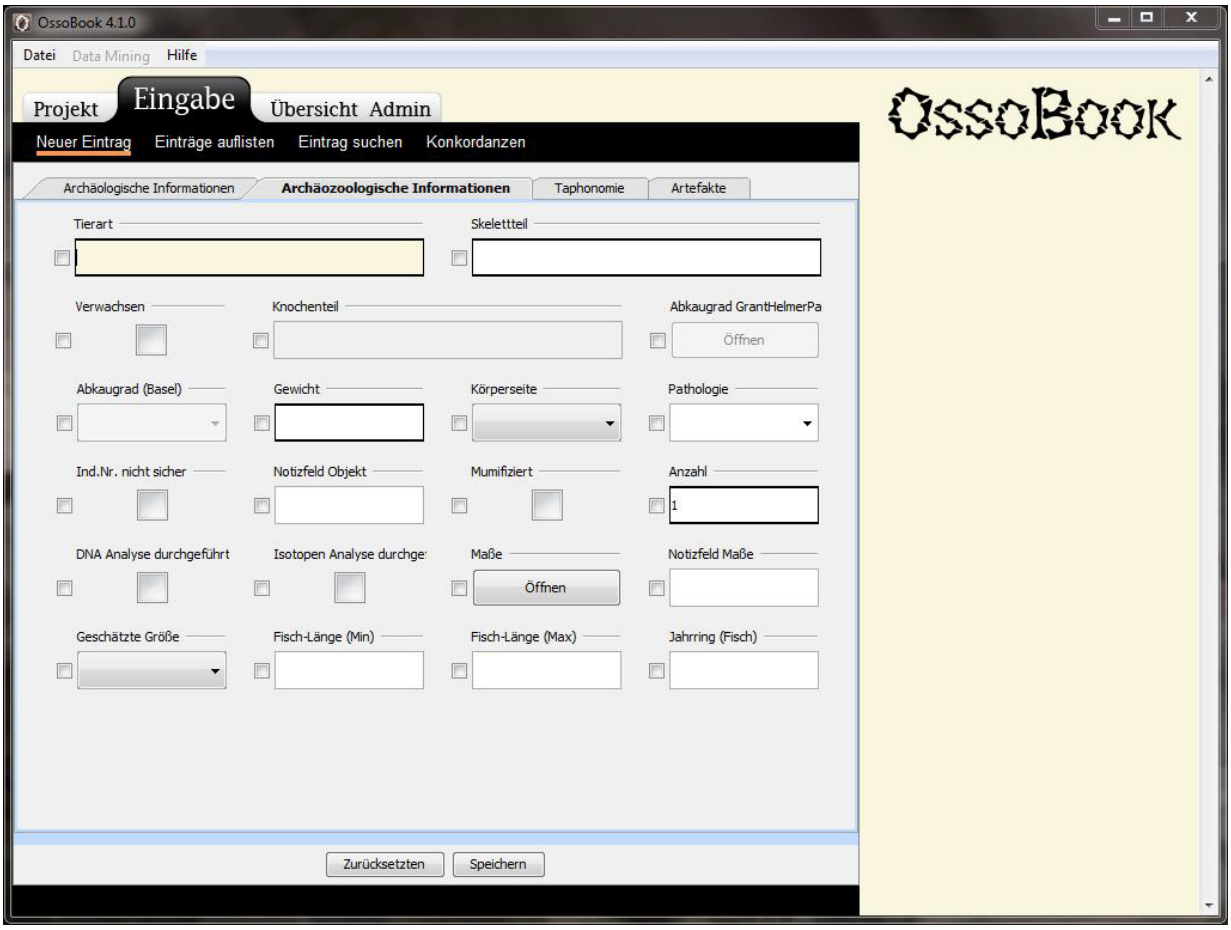


Figure 2.5: *The input mask in* OssoBook *4.1. [Loh12, Kal12]*

workflow process is similar in all disciplines (archaeology, anthropology, zooarchaeology, palaeobotany, palaeontology, etc.), the collected data is different in each special field. An individual database solution based on the OssoBook architecture would greatly support the scientists in their work.

So we set the challenge to provide a generic solution for supporting the scientists in all disciplines, this means that OssoBook has to be as customizable as possible to allow all required information about the specific data to be gathered. Therefore, we used the basic architecture and features of OssoBook and extracted xBook, a generic framework including the common and basic features for a database for archaeo-related disciplines.

Below, we describe the most important functionalities that are features of the xBook framework.

2.3.1 Input Fields and Input Mask

The input fields were strongly enhanced and extended. Previously, there had been only four basic types of fields available: Text, numeric values, check boxes, and Code Tables. Several new types of fields were integrated which can be reused for new input fields, e.g. combo boxes for values and IDs, multi selection data, buttons to open panels for more complex data inputs, date and time choosers, etc. Furthermore, several individual input elements were added that have specifically been implemented for single data elements of OSSOBOOK. Especially the input fields for species, skeleton elements, measurements, wear stages, bone elements, and the gene bank number benefited from the individual input possibilities.

Also the visual presentation of the input mask was updated, as shown in Figure 2.6. Besides the arrangement of single elements inside an input element, they were wrapped with a visible box that was able to be colorized dependent on different states. A mandatory field that has to be filled before saving the entry is highlighted with a yellow background color. If an input is not valid in a field, this is indicated with a red background color. Further enhancements in the graphical user interface were also added, e.g. a box for temporarily displaying text like warnings and errors as a feedback for the user.

2.3.2 Update Procedure

To enable a dynamic development of the application and the version-independent use of the synchronization, it was necessary to keep the database and the program version up-to-date. Only if the local and global database match the same database scheme, the synchronization is able to exchange data correctly. So an update procedure was integrated that consisted of three steps:

When the user logs in, the program version of the local client is checked if it is up-to-date. If not, the OSSOBOOK UPDATER, a small helper application, was automatically started which let the user update the program files by executing the update process.

Then the update process updated the database scheme and the data itself to the current version, if necessary. In the program code it was defined which database version is required for the current version. If the current, local database version did not match with the database version in the program code, the necessary SQL queries are loaded from the server and executed. This was done recursively until the database versions matched.

The Code Tables were updated. To guarantee a consistent data structure, it was also important that the mapping of the values to the corresponding IDs is the same on each client and on the server. So the synchronization was extended with a method that

Figure 2.6: *The input mask in* OSSOBOOK *4.1.14. [Loh12, Kal12]*

updated the Code Tables in the local clients.

2.3.3 Plug-in Interface

At that time, OSSOBOOK provided an interface for plug-ins which was used for the integration of different sets of analyses that were developed by students of the Ludwig-Maximilians-Universität München:

- a module for the analysis of age distribution [Kal12]
- a module for the cluster analysis of measurements [Loh12]
- a plug-in for similarity search on multi-instance objects [Dan10]
- a plug-in for the execution of sample data mining methods [Tsu10], and

- a module offering some analysis methods for zooarchaeological data [Neu12].

These plug-ins were able to be run directly in the application and used with the data from the database. Each plug-in could be imported to the application by simple copying the corresponding jar-file into the plug-in folder of OSSOBOOK.

2.3.4 Database Identification

For the synchronization and the possibility to work offline, it was essential to differentiate data sets that were entered on different computers. The problem is that one single ID for the data sets is not sufficient because the same ID could be assigned on different computers several times which would cause errors in the synchronization process. This was solved by the addition of a new column called Database ID for each entry.

Storing and handling of the Database ID required several iterations to prohibit errors and problems with different aspects of user interaction:

The first iteration considered a file called OBINIT which was a short SQL script that was generated during the registration process and sent to the user. The file included the username and password and additionally assigned a unique Database ID to the local database. The main issue of this solution was that the OBINIT file was necessary to initialize the database, but if the users installed the application on two different computers and used the same OBINIT file, the identical Database ID was used for both computers. Furthermore, the users had to save the email and the OBINIT file because it was not possible to recover the data once the information is lost. This solution also bound the password to the OBINIT file which was also a reason for why the password was not editable.

The second iteration approached these main issues. The assignment of the Database ID was realized by the server while initializing the database. So every time a new database was installed and initialized on a different computer, the server was queried for a new Database ID which was used for the local database. In theory, this approach solved the problems with different computers, however some users created local backups of the application folder which also includes the local database. So it occurred again that the combination of identical entry IDs and database IDs were used when the users restored the application from the backup.

The final iteration closes this gap by defining that the application could not be installed on any folder anymore and additionally saving the database ID in the registry. Now the XBOOK LAUNCHER (cf. Chapter 2.3.7) installs the application data and the database in a folder which grants the logged-in user reading/writing access, e.g. in Windows we use the AppData folder. When the application is started, the Database ID inside the database is checked against the ID saved in the registry. If they do not match, or is not yet available, a new Database ID is issued, which is then saved again in both, the database and the registry.

2.3.5 Registration

Originally, the users could register for an OSSOBOOK account at a password protected homepage only. The users had to enter an email address and got an email including the username, password, and the necessary OBINIT file (cf. Chapter 2.3.4). This information was sufficient to work with the application, however, common mechanics like editing the user name or email address, or change/recover the password were not supported.

Later we changed this system to a more modern approach. The registration was moved directly into the application. At the login screen we added a button to register, where the users can enter some basic information: User name, first name, last name, email address, and a password. Now, there is no user restriction anymore, everyone can register and use the application. Once registered, the users can login to the application without the need of a OBINIT file – due to the reasons described in Chapter 2.3.4. Furthermore, OSSOBOOK was extended with profile settings where the users can manage their provided data. Especially the application was extended with a feature allowing the users to change and to recover their passwords.

2.3.6 Server-Client Architecture

Having a reliable Server–Client infrastructure is an important requirement to be able to synchronize and back-up data. This also helps to ensure no unauthorized changes are done, e.g. by a hacked client. In our case the Server–Client architecture has to handle different scenarios. The first one is the registration and login process. For this it is necessary to connect with the server from anywhere. After the user logged in, the server has to check if the client is up-to-date or first has to be updated. For this the database scheme has to be sent to the client along with values of the code tables. After the version check is completed, the main task of the server is to handle the synchronization requests from the client. These use cases require the communication to handle a variety of dynamic and versatile data. Additionally – since client and server are implemented in different programming languages – built-in serialization tools like the Java Serialization [GHK+15] cannot be used. In an environment where multiple users can create, edit, and share their data, it is important to have a managed architecture that can be accessed and used from everywhere without any restrictions.

2.3.6.1 Challenges

A Server–Client architecture faces many challenges. Many of these are common in every Server–Client application, but some are very specific to the needs of the XBOOK framework.

- **Security:**
  Prevent unauthorized access. Users have only to be able to access the data they have the rights to. Unauthorized access has to be prevented.

- **Availability:**
  The server has to be available from everywhere. Using a Socket-based architecture [Lib] generally requires the usage of ports which have to be manually opened by an administrator in a firewall-protected secure environment. However, the opening of a port is not possible in every working environment because of strict regulations which forbids users to communicate with servers on other ports than 80 (HTTP) or 443 (HTTPS), for example in offices of state authorities or some institutes. Therefore, we had to find a solution how to make a connection from the client to the server possible in spite of restrictive firewall policies and how to use the available ports for the xBook server to accept requests.

- **Scalability:**
  The server has to work for single and also multiple users at the same time. This is true for most Server–Client architectures, still multiple users working on the same server has to be able to work simultaneously and not having to wait for one request to be completed until the next one is carried out.

- **Flexibility:**
  The server should run independent of the Book (an application that inherits from the xBook framework) without any knowledge of data scheme inside the server. Therefore, the specifics of each individual Book should not be hard coded inside the server application, but dynamically loaded from a configuration file or the database.

2.3.6.2 Evolution

The first synchronization of the data in OssoBook was handled by directly connecting to the database on the server from the client application. This connection required a manually entered passphrase which was given out with the registration, but was identical for all users. Additionally – apart from the client itself – no further checks for authorization were made. To address these issues, a C++ server application was created with the development of the xBook framework which now was the communication partner of the client application. The server is connected to the database and analyzes incoming requests if the user has the authorization. If this is the case, the server carries out the command and sends a confirmation back to the client together with data which was retrieved by this request. Both, a Thread Pool [GS02, Goe] and a Connection Pool [Gol14]

were used to allow multiple users to work simultaneously. The communication between server and client was handled via sockets with a custom serialization of all objects that were transmitted. While this architecture provided a fast, secure, scalable, and flexible way to communicate with the server, it became clear that – due to the nature of sockets – it could not be guaranteed that the connection can be established behind proxy servers that only allow certain ports. Therefore, the communication had to be moved to a different type of protocol.

To solve the problem with proxies restricting certain ports, the communication had to be done over ports which are not restricted by most proxies. These are usually port 80 (HTTP) and port 433 (HTTPS). Of course, the possibility remains that certain IP addresses are blocked. However, it would really get into hacking to get around this. We did not explore this possibility further. The server which is running the server application is also running an Apache server. This is used to distribute the xBOOK LAUNCHER and to download the files required to start the individual BOOKs. Besides the server hosts the xBOOK Wiki[8], a MediaWiki that provides helpful information. So it was not possible to change the port to 80 or 433 the old C++ application was listening on.

A new server application was required that does not conflict with the Apache server, but can run alongside it. Many different web applications would allow this, but PHP was chosen as a scripting language, since it does not require additional server configuration.

2.3.6.3 Communication

In a traditional web service, the users would enter a URL in their browsers. The web server would then analyze the request and return a website with the requested information. Since in our case the client has to communicate directly with the server, the response has not to be human readable, but interpretable by the client. Because the client was already able to communicate with the old C++ server, the serialization on the client side was already available and working. Still, it had to be modified that it would be able to communicate with a PHP server. However, PHP is not designed to work with serialized objects, but to load a script with some parameters, and then return and display results.

There are several possibilities to realize a cross-platform serialization. One is JSON [JSOa, JSOb], a lightweight, text-based, language-independent data interchange format. Since JSON uses a human readable format, it has the disadvantage of data overhead [McA]. There are ways to optimize the transmission size, e.g. XFJSON [XFJ] transforms the JSON format into a binary-hex form that is additionally encoded and decoded. Considering that there is no necessity to read the transmitted data and that we expect a huge amount of data sets for single projects, we focus on bandwidth-efficient solutions. So we need an alternative that is not human readable. However, a solution like FlatBuffers

[8] `http://xbook.vetmed.uni-muenchen.de/wiki/`

[Gooa] was not published at the time of the implementation of our serialization method. Protocol Buffers [Goob] do not support PHP at all. BSON [BSO] in direct comparison with Protocol Buffers can give an advantage in flexibility, but also a slight disadvantage in space efficiency due to an overhead for field names within the serialized data. However, BSON is mainly used as a data storage and network transfer format in MongoDB[9] databases. Therefore, we would have had to distribute all libraries for MongoDB which seemed unreasonable, since there is no stand-alone implementation. Because no good alternative for our serialization was available at the time of implementation, we had to implement an own solution.

Since the communication with a PHP server is asymmetrical, requests and results do not communicate the same way. Therefore, it was necessary to split the serialization in two parts:

The first part consists of the data transmitted to the server. For this, the request is serialized to a string which is then appended to the requested URL with the HTTP POST method. This allows the PHP server to read the data and deserialize the string back to the request which is then carried out.

After the server completed the request, the result is serialized again and the resulting string is displayed as the content. This is read by the Java client and is returned deserialized.

The communication is done with a message object. The message object holds the type of the request, e.g. synchronization, login, register, and additionally a list of further data. All classes that can be added as data are instance of the interface `Serializable` which has methods to serialize and deserialize itself. For each request, the type and amount of parameters of the data that is sent is predetermined. Of course, the data itself is not known beforehand.

The serialization requires special classes that implement the `Serializable` interface even for basic data types like `String` or `Integer`. Currently 16 different classes are used. These are mostly required for the synchronization and the initialization of the database scheme.

To secure the communication HTTPS is used. The server should hold no information about the specific BOOK apart from necessary information of the corresponding database so that the independence of the database can be ensured. Information about the tables that have to be included in the synchronization, are saved in tables inside the database. This allows those tables to be dynamically adjusted without the need to update the server. If a request is carried out, those tables are checked whether and which columns can be accessed.

[9] `http://www.mongodb.com`

2.3.7 Launcher

A very important part of the xBook databases is the xBook Launcher. Over the years, the application took on an increasingly important role. It developed from a simple updater application that was executed to check if a new OssoBook update was available to the central place where all Books can be installed, updated, and started independently.

2.3.7.1 OssoBook Updater

The first idea of the OssoBook Updater was born through the necessity to allow the user to update the program. This was required because the latest version of OssoBook was ensured to be able to work in online mode and to communicate to the server, because the local database scheme has to be identical with the global scheme on the server when synchronizing data. So the update process is a frequent process that has to be executed with each minor update of the application. A screenshot of the OssoBook Updater can be viewed in Figure 2.7.

We wanted to avoid that the users had to run and install the update process manually. They should not have to go to the website, download the latest version, and install the files in the appropriate directory. The OssoBook Updater was automatically called when it was detected that the local program version was out-of-date when the user tried to connect to the global database. The OssoBook Updater had a list of files that had to be checked for updates and compared them to the files available at a specific website. Since OssoBook could be installed in an arbitrary directory, the OssoBook Updater had to use this directory to update the files. For this the updater was also in this directory and was updated by OssoBook before the updater was executed.

To prevent different instances being run on one single computer, we had to specify the directory in which OssoBook was located. This guarantees – together with the "Database ID" (see Chapter 2.3.4) – that the instance on a single computer was both, unique and could be identified uniquely. To avoid permission problems, a directory had to be used where writing permissions are guaranteed for the users. For Windows environments we chose the "AppData" directory. This allowed users to easily install OssoBook on one computer, synchronize their data, and continue to work on a different computer.

Instead of updating the files in the directory of the updater, the updater became a independent file that from now on was called xBook Launcher, since it also served as the entry point for the application. So instead of directly starting OssoBook, now the users had to run the xBook Launcher which then checked if all files were up-to-date and then allowed the execution of OssoBook.

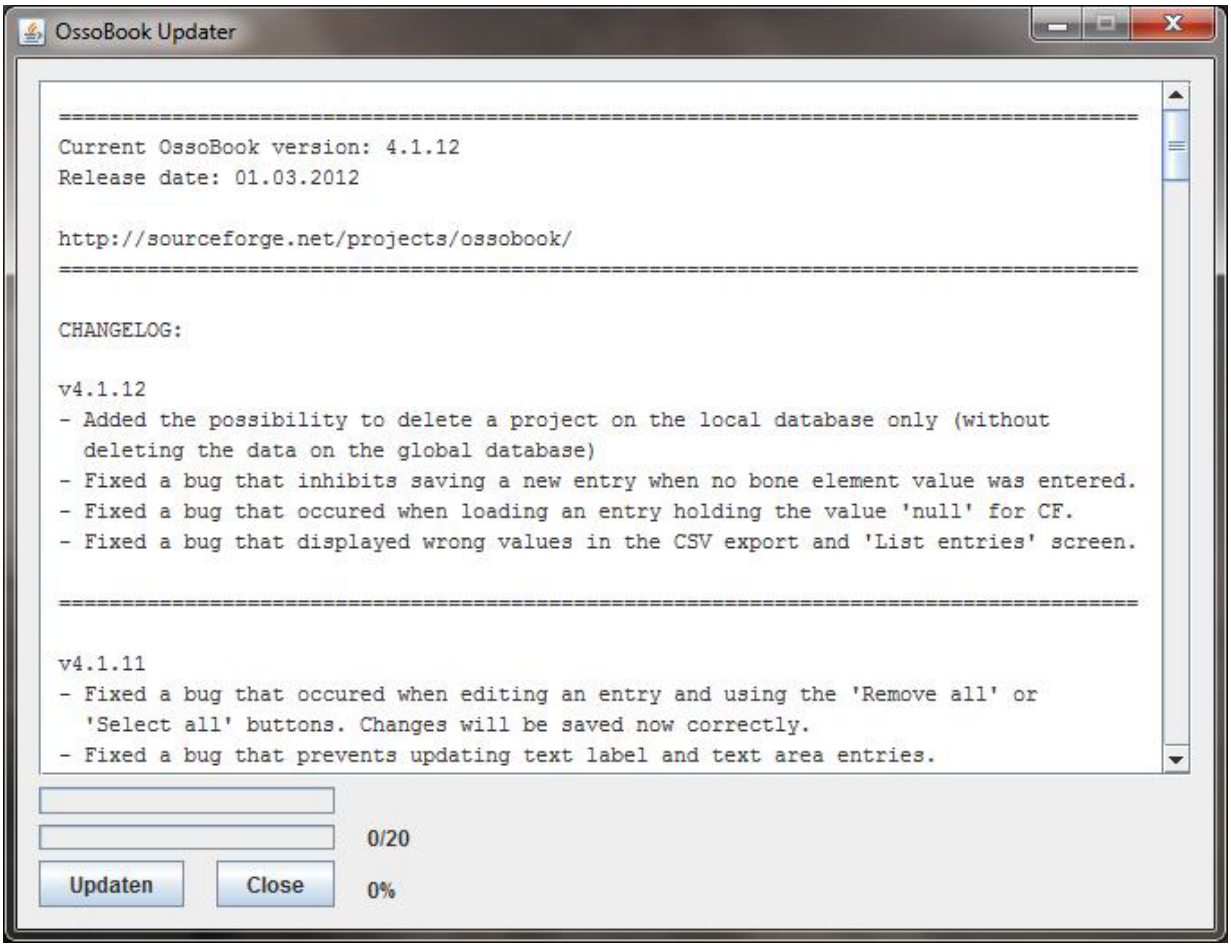


Figure 2.7: *The* OSSOBOOK UPDATER *allowed the user to update the* OSSOBOOK *application. [Loh12, Kal12]*

2.3.7.2 Development of the XBOOK LAUNCHER

With the development of more and more BOOKs for different areas of work, the requirements for the XBOOK LAUNCHER changed and – to avoid the need of several individual launcher applications for several databases – had to be adjusted to support more than one database. Therefore, the XBOOK LAUNCHER was extended with a BOOK selection.

Each supported BOOK was represented as an own row in the selection, displayed by an application icon, the application name, a short description of the database, and the supported languages. The user could select the desired database and execute it. Furthermore, the XBOOK LAUNCHER was extended with general settings that affects all BOOKs (like the language selection and the selection of the automatic or manual synchronization) and a frame to output the messages of the development console. The update functionality was still available which updates all BOOKs at the same time. A screenshot of the version 1.0 of the XBOOK LAUNCHER can be viewed in Figure 2.8

Figure 2.8: *The first* XBOOK LAUNCHER *1.0 with the selection of four different* BOOKs *based on the* XBOOK *framework.*

However, it became difficult to use one single launcher for all available BOOKs. Some of the BOOKs are scientific databases which are publicly available, others are local databases for the inventory of findings in museums or state collections. So not all BOOKs should be accessible in public. Furthermore, also the users do not need to have listed all existing BOOKs in their launcher. The previous additionally required an own instance of the XBOOK LAUNCHER for almost every BOOK, a roundabout way for the increasing number of BOOKs. Furthermore, it became necessary that the different databases were not hosted on the same server any longer. The individual BOOKs should be supported to save their data on their own servers. So the structure of the XBOOK LAUNCHER was renewed again.

Therefore, the XBOOK LAUNCHER was extended to enable adding single BOOKs dynamically to the list of BOOKs. The users can enter a valid URL where the configuration file of the corresponding BOOK is saved – for example *http://xbook.vetmed.uni-muenchen.de/books/ossobook* in the case of OSSOBOOK. The configuration file is defined to be named "book.xml" which holds all necessary information for the corresponding BOOK. This includes especially information that is used in the launcher to display the

information about the BOOK (like application name, application ID, multi-language descriptions, etc.). It also defines the file that should be executed when running the BOOK and the location of the data that is required when installing or updating the application. The structure of the configuration file is illustrated in Algorithm 1.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<book>
  <displayName>OssoBook</displayName>
  <id>ossobook</id>
  <fileName>OssoBook.jar</fileName>
  <author></author>
  <bookColor>#624D47</bookColor>
  <sidebarBackgroundColor>#D5CECA</sidebarBackgroundColor>
  <defaultLanguage>en</defaultLanguage>
  <downloadLocation>
    http://xbook.vetmed.uni-muenchen.de/books/ossobook
  </downloadLocation>
  <languageFiles>
    <entry>
      <key>de</key>
      <value>
        <entry key="description" value="[...]" />
        <entry key="descriptionShort"
                value="[...]" />
        <entry key="author" value="[...] "/>
      </value>
    </entry>
    <entry>
      [...]
    </entry>
  </languageFiles>
  <languages>en</languages>
  <languages>de</languages>
  <languages>fr</languages>
  <languages>es</languages>
  <languages>de_ch</languages>
</book>
```

Algorithm 1: *The (shortened) example structure of the* book.xml *file.*

This way, the XBOOK LAUNCHER can be used as a central platform to manage all available BOOKs, independent on their location. Especially the users benefit from this architecture. They can configure their launcher as they wish and do not have to install or integrate BOOKs they do not work with at all. Furthermore, the single BOOK applications can now be developed and hosted completely separately. In the past, all BOOKs had to use the same program version because they all depended on the same update files. Now, every BOOK can be developed independently from other existing BOOKs and do

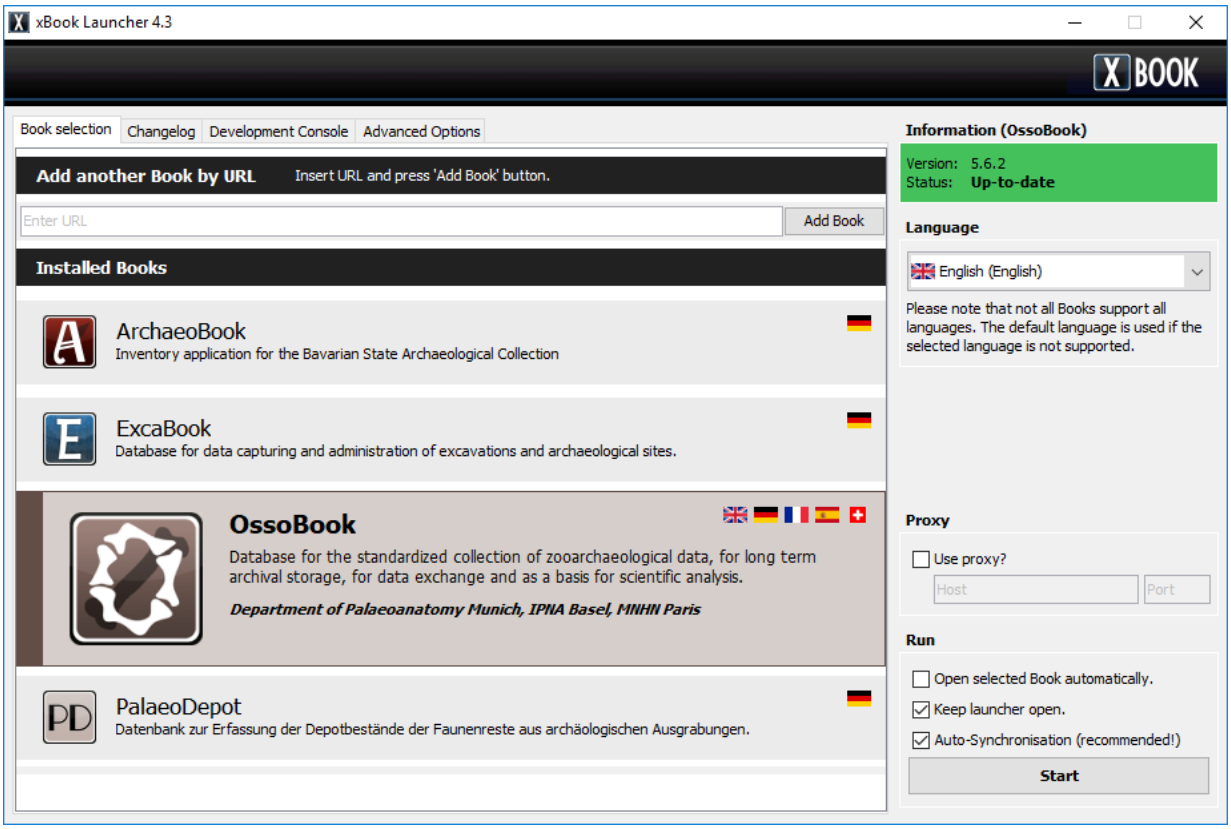


Figure 2.9: *The further developed version* XBOOK LAUNCHER *4.3.*

not necessarily have to be updated to the latest version of XBOOK. This became more important with more and more different BOOKs being published by different institutions and developers.

In addition, the XBOOK LAUNCHER was extended with two technical features. With the increasing amount of users, more users required the usage of a proxy server in order to be able to connect to the server. For this, the XBOOK LAUNCHER was extended with a possibility to enter proxy information. Furthermore, it becomes apparent that a lot of users work on computers with a limited amount of computer memory which was not sufficient for using some of the features of XBOOK, like exporting the partial huge amounts of data. To solve this problem, the XBOOK LAUNCHER was extended with advanced options where the users can allocate more RAM to Java applications, and therefore for the BOOK. A screenshot of the current XBOOK LAUNCHER version 4.3 can be seen in Figure 2.9.

2.4 Features of the xBook Framework

There are many different areas in the archaeological field of science which struggles with the same problems concerning the collection of data. OSSOBOOK is a database for zooarchaeological findings, but similar databases were also demanded for other areas, like anthropology and archaeology. Instead of individually implementing a completely new database for each area, we decided to extract the features from OSSOBOOK into a new framework called XBOOK.

This framework should provide all basic functionalities of the application and provide them to all instances which are called "BOOKs".

2.4.1 Synchronization

Because of the conversion of OSSOBOOK to Java, the collaboration with colleagues within the *Institute of Palaeoanatomy, Domestication Research and History of Veterinary Medicine*[10] and other institutions in gathering and maintaining zooarchaeological data is an important part of the concept. Initially, the data of the application was saved in one global database on the server of the university (cf. Chapter 2.2.2). The employees of the collaborating institutes could connect and work on this database. They could enter, view, and edit the zooarchaeological data directly on the server. This made it possible to work together on one project, enabled the exchange of data with other zooarchaeologists, and fulfills the need to sustainably store data on the cultural heritage as claimed by the UNESCO[11]. Later, a tunnel also enabled connections from other places as well.

But zooarchaeological data is often gathered in field work, i.e. at remote sites that do not offer a convenient environment for IT services. Therefore, it is typically not possible to enter the data into databases that have to be accessed via an Internet connection. A synchronization process was required. A Server–Client architecture was implemented to ensure working offline at remote places, but also storing data globally, where it can be shared with other users. [LKK+14]

The first concept for the synchronization was drafted in 2008. "A client-server architecture ensures that each client [...] manages its own local database that is schema-equivalent to the central database at the server. This way, each client can make its updates locally, independently, and – most important – offline. At a given time, e.g. when a network connection to the server is established, the client and the server synchronize, i.e. the updates of the client are inserted into the central database at the server and the update of the server" [KKO+09]. However, the direct connection to the database without any synchronization would still be possible within the institutes. This architecture

[10] http://palaeo.vetmed.uni-muenchen.de
[11] http://www.unesco.org

is drafted in Figure 2.10. The concept of the synchronization was initially implemented by Jana Lamprecht in 2008 [Lam08].

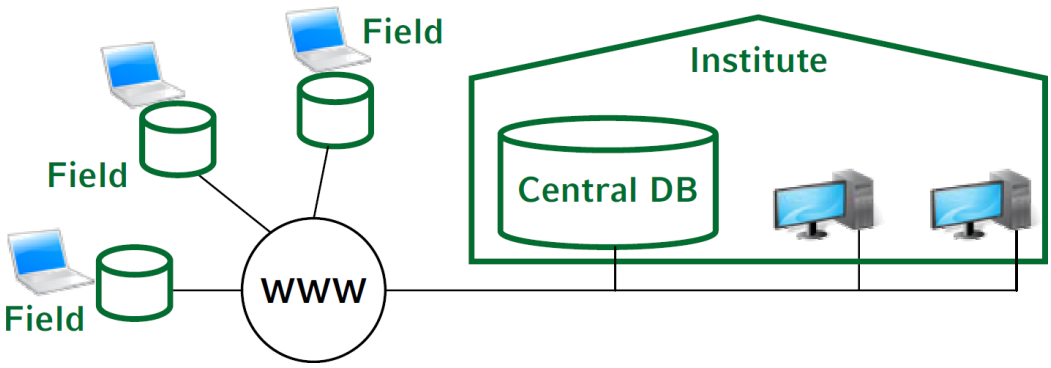


Figure 2.10: *The origin draft for the architecture for the* OSSOBOOK *synchronization. [KKO+09]*

Since this implementation, the concept of the synchronization was updated in many development cycles. The synchronization process itself was enhanced and was extended with new features. First of all, the clients do not directly connect to the database anymore, but to a server application that handles the requests and data manipulations. The server application also enables the feature to let the users manage their basic profile information, especially to change and recover their password if needed – a feature that was lacking before.

Additionally, the possibility to work on different computers with one account was added for a single user. In the origin implementation, this was theoretically also possible, but the technical implementation could cause the loss of data sets during the synchronization process.

The server application was originally written in C/C++, later it was replaced by a PHP program to avoid problems with restrictive firewall settings that made a communication impossible through other ports than port 80 (HTTP) or 443 (HTTPS). Due to security reasons, all the communication has to go through the server application. Therefore, a direct connection to the central database is not supported anymore, not even inside the institutes.

At the same time the logic of the synchronization was also improved and simplified. Also the synchronization panel was extended. Now, it provides additional information about each project, like the number of entries, the project owner, and how many entries are not synchronized yet.

The detailed implementation of the synchronization is described in Chapter 3. A screenshot of the latest panel of the synchronization in OSSOBOOK can be viewed in Figure 2.11.

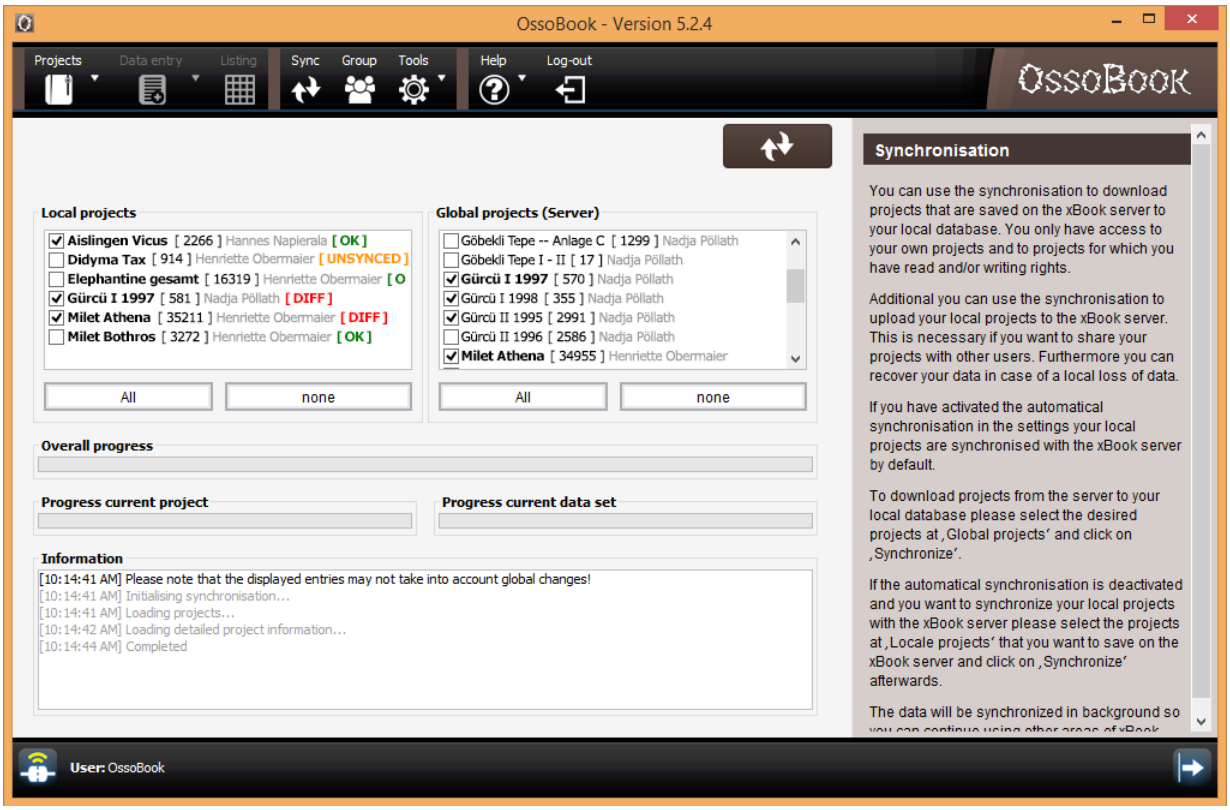


Figure 2.11: *The synchronization panel in* OssoBook *5.2.4. [LKK+16b]*

2.4.2 Graphical User Interface

To enable a flexible graphical user interface for the framework and its applications, it was necessary to modify the existing graphical user interface. All static elements had to be replaced with dynamic elements that can be individually adjusted for each Book. However, elements that are required for each Book have to be integrated to the graphical user interface by default.

In the first step, we replaced the navigation. The old navigation bar was not designed for elements being dynamically added or updated. It was composed of different images for the normal, hovered, and selected states. The displayed texts on these elements were part of the images which made translations difficult. For the xBook framework, the navigation elements are now arranged side by side, are clarified with an individual icon and a short text label. The elements which were displayed in the sub navigation bar before, can now be accessed by opening a pop-up menu. All required navigation elements which are necessary to be accessed in every Book (like 'Projects', 'Data Entry', 'Listing', 'Tools', 'Help', and 'Log-out') are displayed by default. Individual main navigation elements can individually be added by each Book. Adding new elements to the pop-up menu is supported.

The functionality of the elements of the graphical user interface is fully implemented, but the abstract classes provides methods that have to be overridden and implemented by each BOOK. This way the elements can be customized by defining their contents and configurations. As an example, the input fields – either the predefined or custom, new ones – can be defined with their settings and information for the database individually for each BOOK. The overridden classes define the individual input fields. The logic for handling and displaying the data is defined in the implementation of the XBOOK framework. So each BOOK can have different input fields in the input mask, like indicated in Figure 2.12.

The individualization of other elements of the XBOOK framework is similar. The overridden, abstract methods are used to define the information displayed in the project overview screen. The location where to save settings, individual output for the export, the general style of the BOOK (like the logo, the name of the BOOK, used colors), and also the definition of the elements are displayed in the main and sub navigation.

2.4.3 Multiple and Crossed-Linked Input Masks

Before, each BOOK only had one single input mask with input fields where the users could enter data. During the development of the XBOOK framework and its BOOKs it became clear that different, separated input masks are required, especially to avoid redundancy and inconsistency.

So we added the possibility that each BOOK may have an arbitrary number of input masks with individual input fields. These input masks may be independent from each other, but may also be cross-linked among each other.

As an example, many archaeological findings are often grouped in boxes or bags, but the information about these boxes and bags should not be entered for each finding again. An own input mask for the boxes or bags would be an advantage to avoid extra effort when entering data. Also it is possible that the content of a box or bag changes, so it is easier to select another box or bag instead of re-entering the data for each concerned finding.

The realization of these cross-linked input masks does not need a complicated architecture of the tables in the database. But it requires some more complex SQL queries and adjustments in the graphical user interface, especially for the display and selection in the input mask, the representation in the listing and the export, and adjustments in the synchronization process.

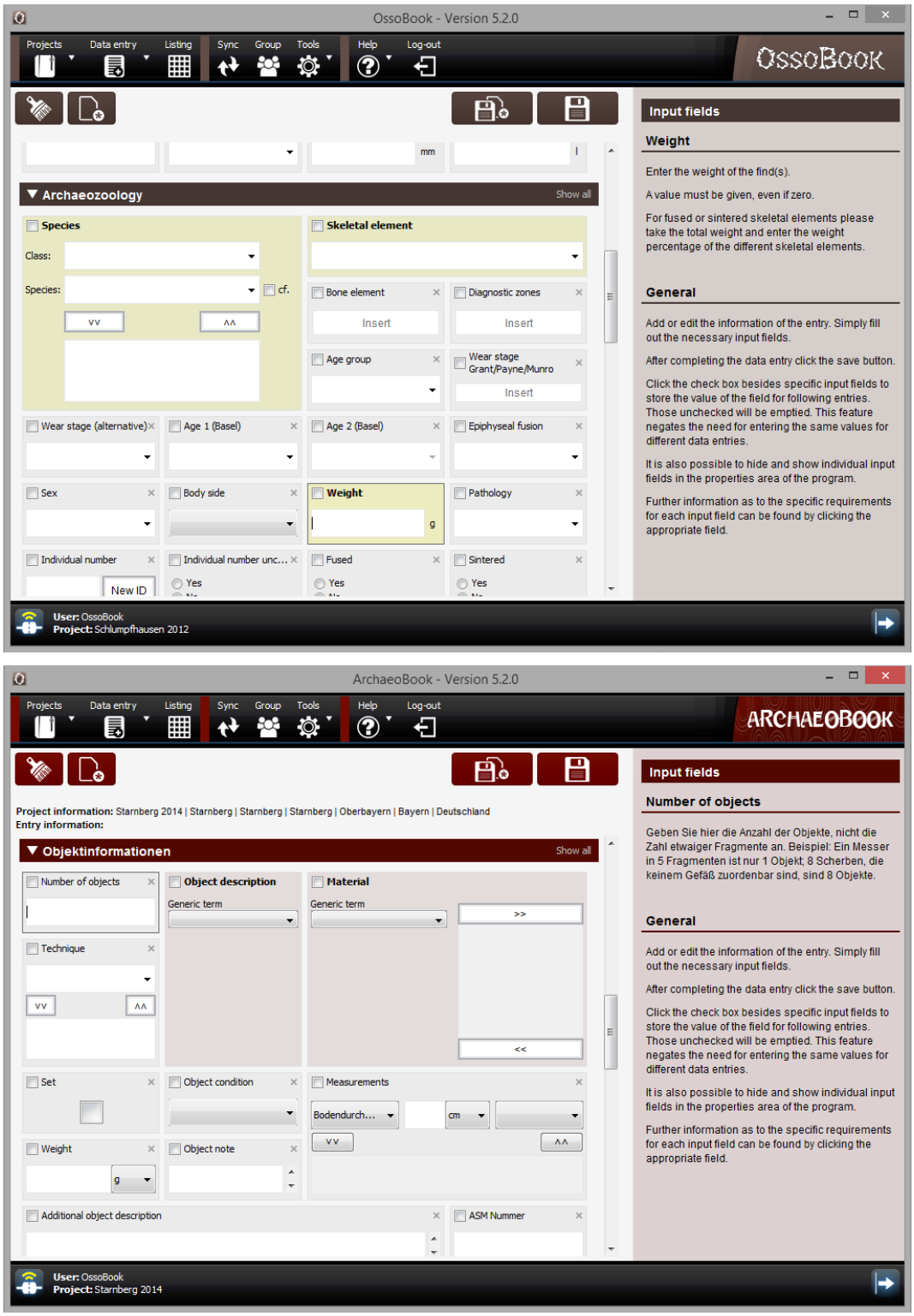


Figure 2.12: *The input mask of* OssoBook *(top) and* ArchaeoBook *(bottom). Both applications are based on the* xBook *framework that provides a basic graphical user interface and functions, but allows customization like e.g. individual input fields. [LKK+16b]*

2.4.4 Listing and Export

Besides the data recording, the listing of entered data is also important. The entered data of all input fields has to be meaningfully displayed in the data listing, i.e. the values should be human readable.

For most of the input fields the value can be directly displayed like simple text or numeric values. Other more complex input fields had to be adjusted for the readability, for example by displaying the corresponding text value instead of the corresponding ID, or by displaying a readable date or time format instead of a timestamp. Input fields with multi-inputs should also display all entered values, not only the IDs. For cross-linked values of other input masks representative text for the cross-linked entry was necessary as well.

The export of data is also an essential feature. Most of the archaeologists and bio-archaeologists still do their analyses in extern tools or applications like Excel, or want to print them for non-digital archiving. Therefore, XBOOK provides an export method that saves the data in a file on the local system. There are two supported files systems to export the data: Comma-separated values (CSV) and spreadsheet (XLS/XLSX) files.

Both, the listing and the export have full support for multiple input masks. In the listing, a selection of the input mask is available where the user can select which data has to be displayed. For the export, it is possible so select single input masks. If a BOOK with multiple input masks is exported, the data of each mask will be saved in an own CSV file, or in an own sheet of the XLS/XLSX file.

2.5 Applications Using the xBook Framework

In archaeo-related disciplines, there are many areas with the same or similar workflow. XBOOK offers a framework in which all databases benefit from the provided features. Currently, there are seven different instances of the XBOOK:

- **OSSOBOOK**, a database for zooarchaeological findings, used by *Bavarian State Collection for Anthropology and Palaeoanatomy Munich*, section Palaeoanatomy[12], *ArchaeoBioCenter*[13], *Institute of Palaeoanatomy, Domestication Research and History of Veterinary Medicine*[14], *IPNA Basel*[15], and members of *BioArch*[16] and *Deutscher ArchaeoZoologenVerband*[17]:

[12] http://www.sapm.mwn.de
[13] http://www.archaeobiocenter.uni-muenchen.de
[14] http://www.palaeo.vetmed.uni-muenchen.de
[15] http://ipna.unibas.ch
[16] http://www.archeorient.mom.fr/recherche-et-activites/participation-a-des-reseaux/GDRE-BIOARCH
[17] http://www.archaeozoologenverband.de

- **ARCHAEOBOOK**, a database for archaeological findings, used by *Bavarian State Archaeological Collection Munich*[18].
- **ANTHROBOOK**, a database for anthropological findings, used by *Bavarian State Collection for Anthropology and Palaeoanatomy*, section Anthropology[19].
- **EXCABOOK**, a database for archaeological findings, used by *Bavarian State Department of Monuments and Sites*[20].
- **PALAEODEPOT**, a database for zooarchaeological findings, used by *Bavarian State Collection for Anthropology and Palaeoanatomy Munich*, section Palaeoanatomy[21].
- **ANTHRODEPOT**, a database for anthropological findings, used by *Bavarian State Collection for Anthropology and Palaeoanatomy Munich*, section Anthropology[22].
- **INBOOK**, a database for archaeological findings, used by *Bavarian State Archaeological Collection Munich*[23].

Availability

The XBOOK framework is available from the URL:
`http://xbook.vetmed.uni-muenchen.de/download/`

[18] `http://www.archaeologie-bayern.de`
[19] `http://www.sapm.mwn.de`
[20] `http://www.blfd.bayern.de`
[21] `http://www.sapm.mwn.de`
[22] `http://www.sapm.mwn.de`
[23] `http://www.archaeologie-bayern.de`

Chapter 3

Sharing Data: A Timestamp-Based Synchronization Process

Attribution

This chapter uses material from the following publications:

- Johannes-Y. Lohrer, Daniel Kaltenthaler, Peer Kröger, Christiaan van der Meijden, and Henriette Obermaier. A Generic Framework for Synchronized Distributed Data Management in Archaeological Related Disciplines. In *10th IEEE International Conference on e-Science, eScience 2014, São Paulo, Brazil, October 20-24, 2014*, pages 5–12, 2014 [LKK+14]
- Johannes-Y. Lohrer, Daniel Kaltenthaler, Peer Kröger, Christiaan van der Meijden, and Henriette Obermaier. A generic framework for synchronized distributed data management in archaeological related disciplines. *Future Generation Computer Systems*, 56:558–570, 2016 [LKK+16b]

See Chapter 1.2 for a detailed overview of incorporated publications.

In Chapter 2, we built the basis to enable the collection of data for different scientific areas. The next step on a scientific workflow is usually to share and back-up data (c.f. Figure 3.1) Having a broad range of data can be vital for a successful and accepted analysis. For this, it might be necessary to also rely on data that other scientists have collected. Therefore, possibilities to exchange data with colleagues are required and necessary. Also archiving the collected data and therefore backing up the data in case of a system crash is necessary.

Figure 3.1: *Sharing is usually the second step in any scientific workflow.*

Different methods for sharing the data are possible, depending on the medium the data is collected with. For documents written on paper, the data can be shared by scanning the page and then mailing the image. Also the page can be copied and then the copies can be sent per postal mail. This can be very time-consuming – of course, this is way faster for digital data. If the data is entered into spreadsheet files, these can be sent directly per email. If the data is stored in a database, the data can be exported in to a spreadsheet file and then be shared. Still, there might be a high effort required to separate the data if only parts of the data is supposed to be shared. The easiest solution would be a synchronization process that is built into the application where data is collected. Such a process could improve the speed and quality of data since all data is based on the same data scheme.

The synchronization was already shortly mentioned in Chapter 2.4.1. In this chapter, we introduce the synchronization process of the XBOOK framework and describe the functionalities in more detail.

3.1 Introduction

Today database systems are a central component of software, websites, and data preservation. The large amount of data could barely be administered without them. They offer an efficient, consistent, and persistent way to save data flexibly in one central place, offer mechanics for data security and control, and allow the simultaneous access of several users to the data with transaction management.

Database systems have very far evolved to function flawlessly if the database is stored in a central place, for example on a server. The users can connect via network or Internet. This becomes more complicated on distributed architectures if data is entered in different databases and tables, and this data has to be exchanged among them. In this case synchronization is necessary that regards important features of the database management systems like consistency, durability, and integrity.

Still, different use cases require different types of synchronization. For a company

that has several servers with a database containing all necessary information, this database has to be automatically and quickly synchronized between all servers. In the modern world, where the users want to start writing e-mails at home on their tablet during breakfast, send the mail on their smartphones while commuting, and then receive the reply at work on their laptops or personal computers, it is necessary to synchronize the contacts, mails, and calendar between the different devices.

Not only the daily life, but also scientific data can benefit from a synchronization, by providing the possibility to collect the data without an Internet connection. Archaeology and bioarchaeology are an example where this is necessary. We will use this domain as an example throughout this chapter, but the problems and requirements are also valid for many other scientific areas.

In the archaeo-related disciplines, as in many other areas, a main part of the work comprises in collecting, sharing, and analyzing 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 analyzing 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 have to 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 doing simple descriptive statistics. This is significantly inconsistent with the need to sustainably store data on the cultural heritage claimed by the UNESCO[24].

Obviously, researchers, like from 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 application. The key limitation of such an IT service is the problem of multiple users that need access to data recording and data analyses even if a permanent Internet connection cannot be established. A synchronization process is required, implementing a client-server architecture as visualized in Figure 3.2, to ensure working offline at remote places, but also storing data globally, where it can be shared with other users is the solution.

Existing commercial solutions for this problem are typically integrated in a dedicated database management system and/or cloud service. For license 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

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

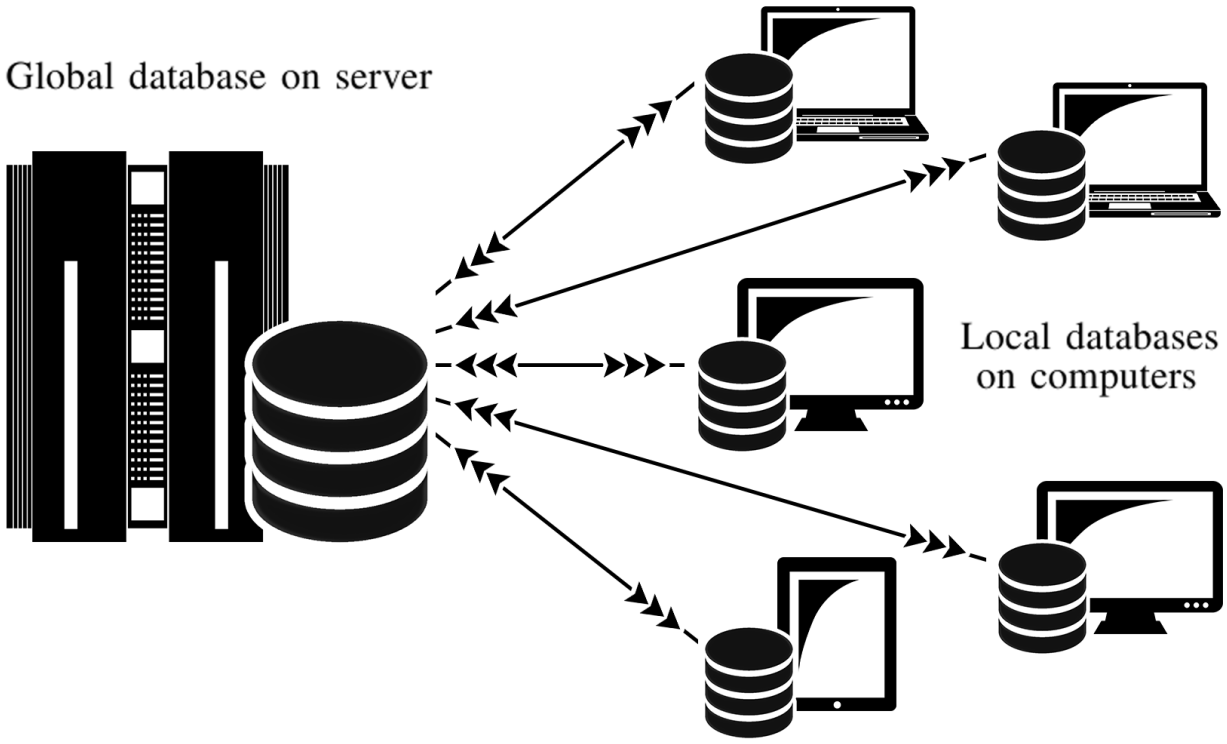


Figure 3.2: *The local clients are connected to the global server. The synchronization allows data exchange, so data can be recorded on the local machines, but can be backed-up and shared via the server.*

completely unrealistic. In addition, as long as the data is not yet analyzed 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 chapter, we describe the synchronization, a vital component of the framework XBOOK, a solution for the sketched problem that follows the architecture depicted in Figure 3.2 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 by default. Since the synchronization uses only a JDBC connection, and limited changes to the databases, which can be done in each relational database, it is possible to change the underlying database management system.

In summary, the main contributions are as follows: We list a set of requirements that should be addressed by e-Science infrastructures for a synchronized distributed data management and data analyses in the archaeological sciences that have been extracted from comprehensive discussions with domain experts, reflecting their typical working procedures (cf. Chapter 3.2). We discuss existing solutions for synchronized distributed data management and data analyses in Chapter 3.3. We describe the synchronization

process of XBOOK in more detail (cf. Chapter 3.4) that is independent of the data model and could potentially be used in any application domain if needed. Finally we describe the realization of the synchronization within the application (cf. Chapter 3.5)

3.2 Problem Formulation

In this chapter, we discuss the requirements of an e-Science infrastructure for (exemplarily) the archaeo-related sciences in more detail. By developing a database for zooarchaeologists, we have identified the following requirements of a synchronization that have to be fulfilled so that working with the data is possible.

Let us note that depending on the use case of the application some requirements listed below are obligatory and additional requirements might exist. However, the following list has been extracted after extensive communication with domain scientists analyzing their particular working procedures.

A synchronized distributed e-Science infrastructure for data management and data analyses in the archaeo-related sciences should address the following issues (the order of appearance is arbitrary and does not reflect priorities):

- **Distinctability of entries:** Two different entries have to 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. If they change the same entry before synchronizing a conflict can occur. The synchronization needs to 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 logistic reasons e.g. there is no Internet connection available. It has to be possible to continue working offline and synchronize the data at a later time when 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 has to be able to continue later and no data may be lost or corrupted.
- **Modularity:** To avoid the transfer of unnecessary data, a selection of data is required. In general, a particular users do not need or even are not allowed to synchronize all entries that are saved in the global database with their local device

but only parts of it. The users should be able to select parts of data they want to synchronize in order to save time and to reduce data overhead, so data should be separated in modules ("projects"). Furthermore, the local database of the users should not save any data sets, for which they have 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, have to be able to insert new entries and edit or delete existing ones. These changes have to be passed to the users as soon as possible, to ensure that they can work with current data.

- **Rights management:** Not every user needs to see all the data. The synchronization has to check if the user, who has requested any data, has appropriate rights before sending or committing specific 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 hast to 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.

3.3 Evaluation of Existing Synchronization Methods

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 most commonly used database systems worldwide. In the following, we discuss the synchronization techniques of such systems, including Microsoft SQL Server, Oracle and MySQL in the context of the requirements derived in Chapter 3.2.

- **Snapshot Replication:**
  Snapshot Replication is an easy way to share data that is used in Microsoft SQL Server and Oracle. It "distributes data exactly as it appears at a specific moment in time and does not monitor for updates to the data" [Mic08, Types of Replication]. The entire snapshot is generated and sent to the receiver, a selection of data is not possible. But especially if there is a lot of data saved in the database, it is necessary

to transmit all of this data - so the merge function allows several users to work on different sets of data, but does not reduce the amount of data that has to be sent. At least Oracle supports partial snapshots [Ora13a, Understanding Replication], but synchronizing single data sets is not supported. Furthermore, Snapshot Replication is laid out if data changes are infrequent. This is an unrealistic scenario in our application.

- **Transaction Replication:**
  Microsoft SQL Server supports a transaction replication that is not based on data sets, but on transactions. "[D]ata changes and schema modifications made at the Publisher are usually delivered to the Subscriber as they occur (in near real time). The data changes are applied to the Subscriber in the same order and within the same transaction boundaries as they occurred at the Publisher" [Mic08, Types of Replication]. Therefore, a transactional consistency is guaranteed even on high volume of insert, update and delete activities. But the transaction replication is developed for server environments. The clients have to have a permanent connection with the global server. Working offline is not supported.

- **Replication with Streams:**
  Using Oracle Streams it is possible to enable replication as well. "The stream can propagate information [...] from one database to another. [...] you control what information is put into a stream, how the stream flows or is routed from database to database, what happens to messages in the stream as they flow into each database, and how the stream terminates" [Ora13b, Understanding Streams Replication]. So Oracle Streams are powerful features that also support replications with the help of three background processes: The capture-process to collect information, the propagate-process to get the data out of the stream and the apply-process to handle the local changes (for more details, see [Ora13c]).

  However, a synchronization with the requirements of Chapter 3.2 is not possible by using Oracle Streams. The communication cannot be managed by the global server because all processes have to be defined by the local databases. To ensure that no reading operations are executed on the global database, it is necessary that the users are not administrators of their own local databases. The other way round it means that the global server is responsible for the local databases as well - but this is not possible in practice. [Lam08, p. 58]

- **Merge Replication:**
  Another type of replication in Microsoft SQL Server is merge replication. "[D]ata changes and schema modifications made at the Publisher and Subscribers are tracked with triggers. The Subscriber synchronizes with the Publisher when connected

to the network and exchanges all rows that have changed between the Publisher and Subscriber since the last time synchronization occurred" [Mic08, Types of Replication]. The merge replication matches with the requirements pretty well, but the complete data sets have to be in the database of the subscribers. Therefore, it is not possible only to store or synchronize specific projects. This makes the management of permissions for the synchronization also impossible.

- **Master-Slave Replication System:**
  MySQL supports replication by using a master-slave system. It "enables data from one MySQL database server (the master) to be copied to one or more MySQL database servers (the slaves). Replication is asynchronous by default; slaves do not need to be connected permanently to receive updates from the master" [MyS13]. Unfortunately, it is only possible to replicate all databases, selected databases or selected tables within a database - the replication of single data sets is not supported. Furthermore, MySQL does not include any conflict management.

- **Third Party Synchronization Software:**
  Products like SQL Anywhere [Nov] and PervaSync [Per] allow the synchronization of data from a central database to many connected clients. They also support with restrictions entering and updating data while working offline. Also subsets of data can be synchronize and therefore a right management can be implemented. Still this subset has to be defined for each client, so the user can not synchronize specific projects. Also they require licenses per user, so that in an open environment where the program is distributed to arbitrary clients this is not applicable.

- **Direct Working on the Global Database:**
  Even if it is not really a replication system, it is still a possibility that the users directly connect to the server and work on the global database. It would be an easy solution that fulfills almost all of the requirements of Chapter 3.2 - but of course a permanent Internet connection is required.

- **Clocks:**
  The Lamport Clock (also Lamport timestamp) [Lam78] keeps track of the order in which events on distributed systems occurred. A logical clock ordering algorithm allows to create a partial ordering of events and provides methods to determine facts about the order of the events. This is achieved by Lamport introducing a software mechanism that increments a counter maintained in each process. At the same time it does not allow to retrieve the actual time of the changes. Therefore, it is also not possible to only synchronize parts of the data. All entries have to be synchronized in the order of their last time of the change on the distributed systems.

Similar to the Lamport Clock, Vector Clocks [Fid88] [Mat89] keep track of the order of the events. But additionally, a vector of counters for the individual processes is sent together with each data set, so each process has its own clock. This allows to meet the problem of the Lamport Clock to enable the synchronization of only a subset of entries, but it increases the data overhead that has to be sent if many clients edit the same data set.

Vector Clocks have been designed under the assumption of a fixed, well-known set of participants. However, this is not the case in this very scenario – and also not in our requirements. To counter this problem, Almeida et al. introduced Interval Tree Clocks [ABF08], a clock mechanism that is usable in scenarios with a dynamic number of users in which the identities and number of processes in the computation is not known in advance. The mechanism has a variable size representation that adapts automatically to the number of existing entities, growing, or shrinking appropriately. This approach appears to be over the top for our requirements, since the creation of processes without global identifiers is not required for our application. So, this makes the required data overhead not necessary.

- **Timestamps:**
  Timestamps on data sets are used to keep track of the time when the data set was last updated. These are saved within the data sets and are considered during the data exchange. During the synchronization process, the entries can then be ordered according to the timestamp. This way, the synchronization of subsets of data is possible. However, it can lead to problems if the system times of the networked clients are not synchronous, for example because of different time zones or a wrong configured time setting.

While all of these methods are sufficient for many areas of application, 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 data sets, which are two of the most important requirements for archaeologists and bioarchaeologists. Therefore, we created a synchronization that fulfills all previously stated requirements, is independent of the database management system and hence can be used for any database system.

3.4 Synchronization

A simple way to create a database independent synchronization is to implement a type of timestamp synchronization. Still there are many different approaches how to achieve

this. A basic synchronization can be implemented rather quickly, but often many challenges arise. A commonly used method, is to use triggers that locally track the last time a data set was updated. During the synchronization in this approach the local time is used to commit all changes that happened since the last synchronization locally and send them to server. Then all changes on the server since the last synchronization are pulled from the server. This method bears the problem, that when the connected local databases have a different system time, the created timestamp is not correct and may lead to problems during the synchronization process, by possibly omitting entries from the synchronization. Our approach uses only the system time on the global database, which removes the problem with asynchronous system times on clients. In the following we describe the realization of our synchronization.

3.4.1 Realization in the Database

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

3.4.1.1 Database ID

One of the most important concepts is the Database ID. A column for this ID is added to each table a user can insert data. In addition to the existing primary keys, this column is marked as a primary key as well as to allow several distinct entries with the same ID from various databases. The value of this number for this database itself is stored in a separate table (in XBOOK we use "version") 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 will not be changed.

Because we want to enable a modular approach with projects, all tables also have to add the ID ("ProjectID") and the Database ID ("ProjectDatabaseID") of the project, otherwise the entry cannot be assigned to a specific project (see Figure 3.3).

3.4.1.2 Status

To achieve conflict management as well as identification which of the entries have to be updated, locally the column "Status" (cf. Figure 3.4) is added to each table. It stores the time the entry was modified on the server. This value is modified only on the server

ID	DatabaseID	ProjectID	ProjectDatabaseID	UserID	ExcavationNr	ArchaeologicalUnit	...
2	1073	1	1073	108	M-2011-13-1	1052	...
2	1079	1	1079	114	76/3724/230	630	...
2	1096	1	1096	132	HY.03.ISI	01.013	...
2	1147	3	1147	8	M-2013-732-1	4	...
2	1155	1	1096	132	HY.03.ISI	01.080	...
2	1174	1	1316	217	M-2007-58838	13819 C	...
2	1218	1	1218	113	HY.81	2275	...

Figure 3.3: *The table "inputunit" of the database of* OssoBook *[KLK+18a] as an example for the primary keys* ID, DatabaseID, ProjectID *and* ProjectDatabaseID. *These primary keys are necessary in every data table of* xBook.

via a trigger that updates the status as soon as the value is inserted or updated. Zero is added to the current time to convert the format from "yyyy-MM-dd HH:mm:ss" to "yyyyMMddHHmmss" to receive a sortable and numeric value. The trigger can be seen in Algorithm 2.

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

Algorithm 2: *Trigger to update the status column to the current time in entry tables.*

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. This means that in the meantime someone else modified the entry. Then this entry can be marked locally as a conflict.

In the archaeological context it is not a realistic scenario with more than one scientist collecting data about the same objects (e.g. bones of an animal) simultaneously, so conflicts will not occur frequently. However, conflicts may occur anytime, therefore a conflict management system is necessary. The one we implemented in xBook is described in Chapter 3.4.2.7. Other working areas may need more complex conflict management systems to avoid conflicts hampering the synchronization process.

3.4.1.3 Message ID

The next important addition is the column "MessageID" (cf. Figure 3.4). It stores the current status of the entry (locally). The different states 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.

...	Value	Status	MessageID	Deleted	...
...	24A45 H1	20150624145906.000000	0	N	...
...	26B34 A1	20150624145917.000000	0	Y	...
...	6773	20150624150044.000000	-1	N	...
...	Lab01 C1	20150624150058.000000	-1	N	...
...	Lab03 B12	20150624145945.000000	0	N	...

Figure 3.4: *Three important system columns in the input tables: "Status", "MessageID" and "Deleted". The entries with the message ID "–1" are conflicted.*

- **Changed (1):**
  Indicates that the entry was changed locally and has to be synchronized. This prevents the entry being updated with data from the global database to prevent overriding changes. In most cases, when the marked entry was to be updated from the global database, this would also mean that a conflict occurs, but conflict checking is already done when the entry is committed, so this is ignored. 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 cannot be committed or updated by the synchronization. A conflict occurs when an entry that is changed locally has an older (lower) status than the entry on the server. This means that 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.

3.4.2 Realization in the Application

With the changes to the database a big step in achieving the requirements is completed. Still the synchronization has yet to be executed and additional requirements have to be met, which can only happen inside the application. Therefore, several changes had to be implemented into the XBOOK framework as well.

3.4.2.1 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 do not match, the server is queried for an update. This check is done after the connection to the database is established, but before the user starts working with the data. If the user has

no Internet connection of course the update cannot be made, but since an upgrade could not be made without an Internet connection, this should not be a problem.

3.4.2.2 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 called “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 also be done during the usage of the application. To find out which values are changed, all tables have to have the column “Status”. Just like for entries, the value of the status is the timestamp of the last global change and is updated with triggers on insert and update (cf. Chapter 3.4.1.2) as seen in Algorithm 3.

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

Algorithm 3: *Trigger that updates the status column to the current time for Code Tables.*

Then only the values that have a higher status than the highest value in the local table have to be updated or inserted. To avoid unnecessary update checks in each table if no change was made, the last update can not only be found out by run through all tables, but has to be saved directly in the “version” table. To ensure that this has always the newest timestamp on the server, the trigger is extended by another command to set the last update, so that the complete trigger now looks like in Algorithm 4.

```
1 SET NEW.Status = NOW() + 0;
2 UPDATE version
3    SET version.LastUpdate = NOW() + 0;
```

Algorithm 4: *Trigger that also updates the status column in the version table.*

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 done if the local value is lower than the global value, and only when, the Code Tables are out of date and therefore have to be updated. A example of a Code Table can be found in Figure 3.5.

id	language	value	LB	Status	Deleted
14	0	Felis catus	16	20110118140546.000000	N
15	0	Canis familiaris	4	20110118140546.000000	N
16	0	Equidae	2	20111004103152.000000	N
17	0	Equus caballus	2	20110118140546.000000	N
18	0	Equus asinus	2	20110118140546.000000	N
20	0	Sus domesticus	3	20110118140546.000000	N
21	0	Ovis aries	1	20110118140546.000000	N
22	1	wahrscheinlich Ovis aries	1	20110707080440.000000	N
22	2	probably Ovis aries	1	20110707080358.000000	N
23	0	Capra hircus	1	20110118140546.000000	N
24	1	wahrscheinlich Capra hircus	1	20120719102831.000000	N
24	2	probably Capra hircus	1	20110707080142.000000	N
25	0	Ovis aries/Capra hircus	1	20111004103009.000000	N
26	0	Bos taurus	13	20110118140546.000000	N

Figure 3.5: *The Code Table "animal" in* OSSOBOOK *that defines the available values for species. Adding the column "language" allows using terms in different languages: 0 for general terms, 1 for German, 2 for English, etc.*

3.4.2.3 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 have to 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 Manager*s 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 Manager*s 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. Tables represented by the Base and Extended Project Managers only have ”ProjectID” and “ProjectDatabaseID” columns, no “ID” and “DatabaseID” columns.

3.4.2.4 Data Structure

To store the data and retrieve a complete entry we created some classes to easily load, save and update the data (cf. Figure 3.6).

- **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.

3.4.2.5 Synchronization Process

The actual process of the synchronization consists of three different steps.

- Check if the user has the required rights to access the project and therefore is allowed to synchronize it.

- All uncommitted, not conflicted entries in project and entry tables are retrieved, one by one from the database and sent 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

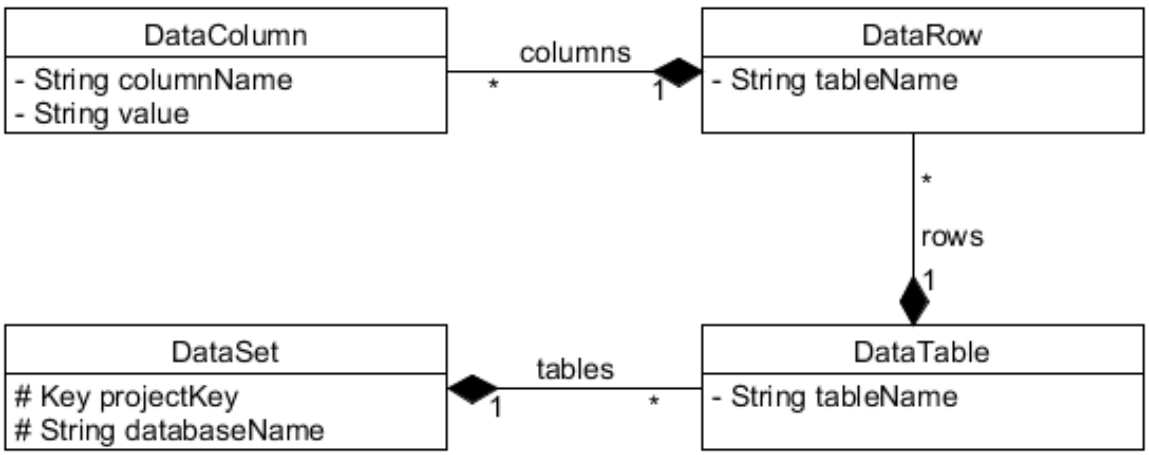


Figure 3.6: *UML visualization of the data types handling the data in* XBOOK.

the entry already exists in the global database, then the server checks the sent timestamp of the entry with timestamp that is saved in the global database. If the global timestamp is newer than the local one there is a conflict and the client is notified (cf. Chapter 3.4.2.7).

- 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, i.e. the last entry that was synchronized, is transmitted. To prevent a 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 does not already exist or is not conflicted or changed (see Figure 3.7).

3.4.2.6 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 (cf. Figure 3.4). 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

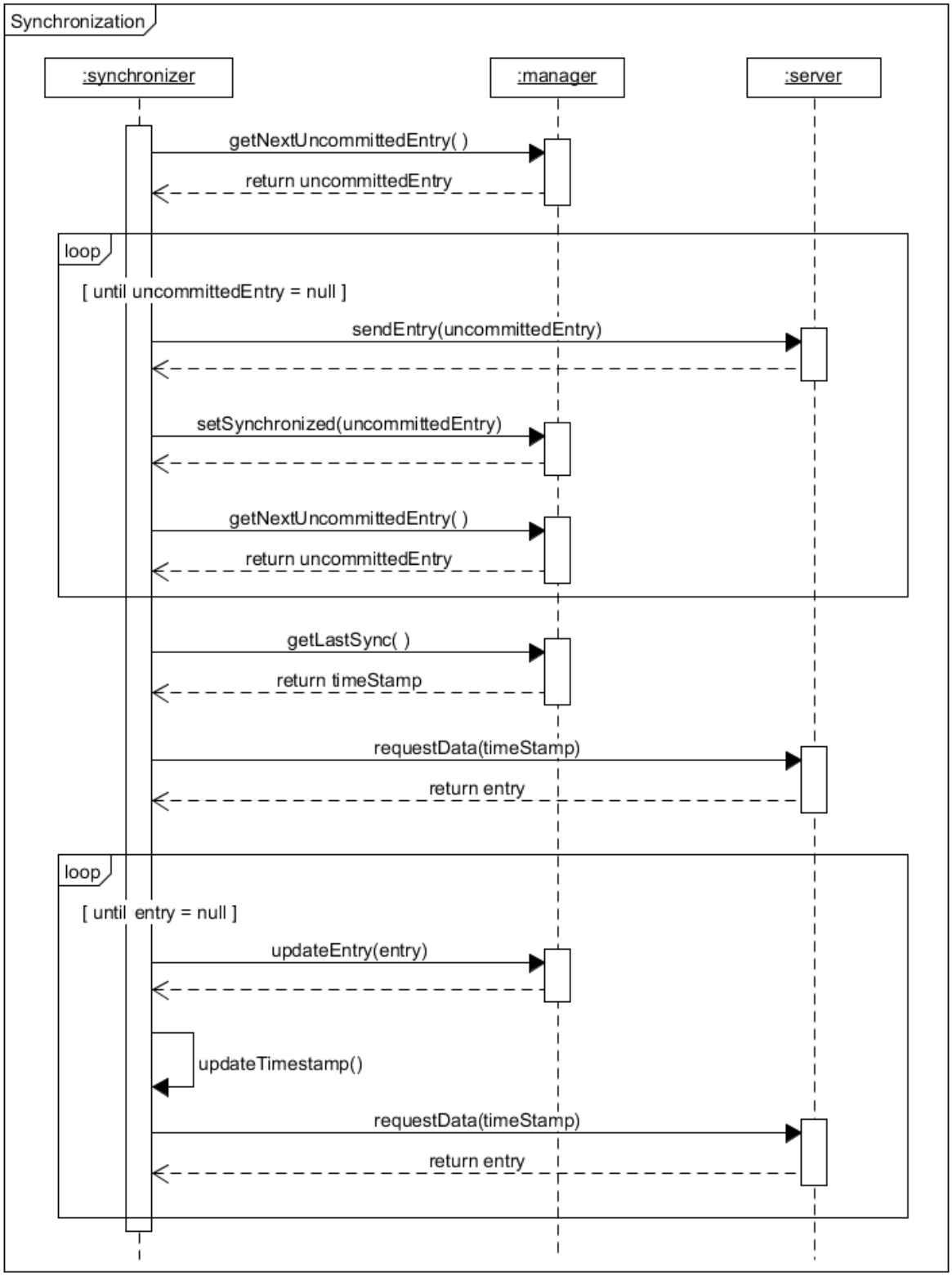


Figure 3.7: *Simplified visualization of the synchronization, displaying the commit of changed entries to the server and new data from the server.*

deleted, because the information about the change has to 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.

Furthermore it is also possible to remove projects from the local databases. These are only deleted on the local machines, but not on the server. So they can be loaded by using the synchronization any time.

3.4.2.7 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. Therefore, the application displays that the project contains conflicts that the user can solve manually by using the Conflict Management Screen.

The Conflict Management Screen provides both the global and local entry and allows the user to select the diverse values. To solve the conflict, the merged entry is saved to the global database with the timestamp of the global entry. If the entry was updated between the solving of the entry and committing the entry to the global database, this ensures that this change will not be overridden, but a new conflict is generated. If no new conflict was generated, the local entry is updated with the merged values, but the local timestamp is not updated, and the entry is marked as synchronized. Then the entry will be updated as usual when the synchronization process is executed.

Conflicts do not have to be solved immediately during or after the synchronization. It is possible to continue working on the projects even if there are still existing any conflicts in the project. This makes it also possible to inform the project owner about existing conflicts if the users are not able to solve them on their own. However, conflicted entries will not be synchronized to the global server until they are solved

Since the conflict management is a topic of its own, we only describe the solution that was implemented in XBOOK. In general conflicts can be solved in many different ways.

3.5 Synchronization in the Graphical User Interface

As presented in Chapter 3.4 the synchronization consists of a powerful, but complex architecture. However, the realization in the application has to consider that most of the archaeologists are not used to work almost exclusively with a computer.

Therefore, it is absolutely necessary to hide the complexity of the synchronization behind an intuitive user interface. We have to ensure that the synchronization can easily be used and that for every user even if not technically skilled could execute it.

Here we describe how the synchronization is integrated into the graphical user interface of XBOOK:

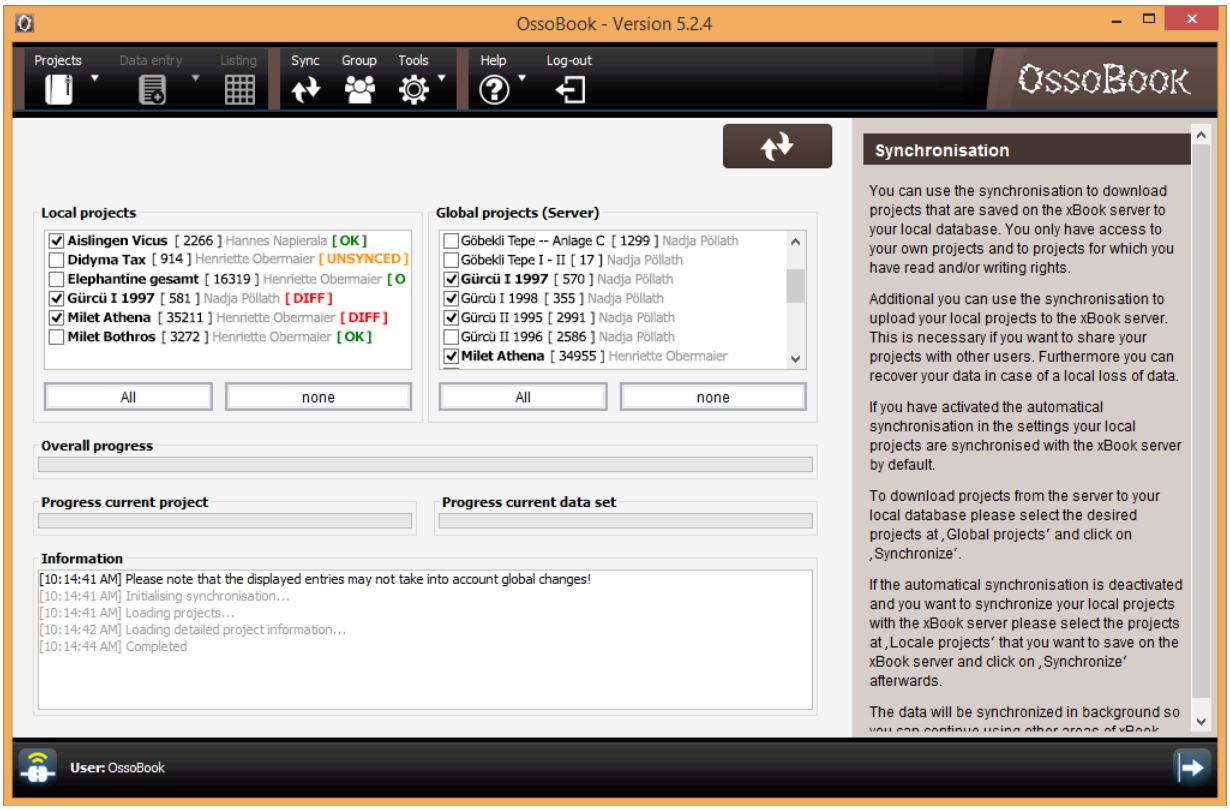


Figure 3.8: *The synchronization panel in* OssoBook *5.2.4. [LKK+16b]*

3.5.1 Manual Data Synchronization

To exchange project data there are three basic procedures that have to be possible in the synchronization panel (cf. Figure 3.8).

- **Global projects** for which a user has read and/or write permission have to 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 have to 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) have to be updateable. For this purpose the corresponding projects have to be selected, like explained above. However, the application has to recognize if a project was selected 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. Depending on the Internet speed and the number of projects and data sets, the synchronization may

take several hours. Thus, user feedback is displayed in message boxes and progress bars (each one for general, project and data set layer).

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

3.5.2 Automatic Data Synchronization

The automatic data synchronization can be activated in the application settings. Thereby, the project information and data sets are synchronized with the server automatically in the 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.

The automatic synchronization can be a big advantage when collaborating on one project in a group. Due to the fact that the automatic data synchronization is updating the data in the background, all members of the group have less unsynchronized data sets scattered in several local databases. Every collaborator immediately gets the current data and does not have to wait until the data from other collaborators is manually synchronized.

Another advantage of the automatic data synchronization is the possibility to use the synchronization as a backup method. In case of a hardware failure (in particular a hard disc crash) every synchronized data set can be restored by reloading the data from the server. When synchronizing the projects manually, there might be projects and/or data sets left that were not synchronized before, and therefore may get lost because of a hardware failure. Because the automatic data synchronization always commits data in the background, the chances for a data loss are minimized.

Data sets that are conflicted will not be synchronized in general, thus the automatic data synchronization will not commit them as well. Conflicts have to be solved manually before the data is able to be synchronized to the global server.

3.5.3 Code Table Update

The code table update that was mentioned in Chapter 3.4 is run automatically by the application after the login if it is necessary. However, the user can also reload all Code Tables manually by using the option in the “Tools” navigation element.

3.5.4 Conflict Management

During the synchronization process data conflicts may occur. One simple scenario for this case as an example: Two different users download data from the server by using

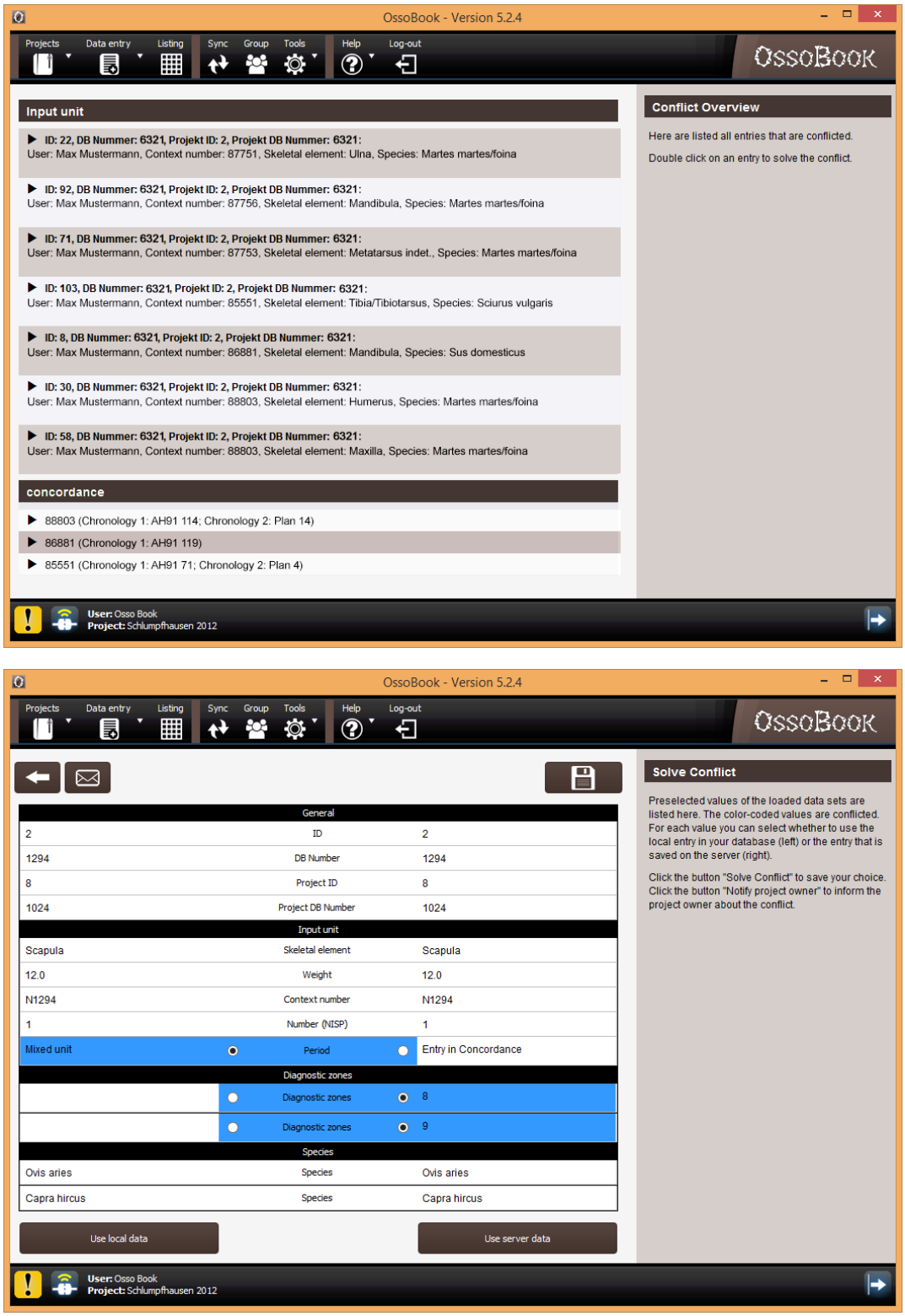


Figure 3.9: *Top: The Conflict Management Screen that displays all conflicted entries of the loaded project. Bottom: The Solve Conflict Screen allows the user to use the local or server values for a specific conflicted entry.*

the synchronization and update an identical data set. If user A uses the synchronization first to upload the changes to the server there will be no problem. If user B submit his changed data afterwards, it is not clear anymore whether the data of user A is the correct one or the data of user B. A conflict occurs that has to be solved manually.

For this case XBOOK highlights conflicted projects in the project overview screen to inform the user about current conflicts. Once the project is loaded the user can click the conflict button in the footer bar to open the *Conflict Management Screen* (cf. Figure 3.9 top). The user gets displayed all current conflicts, sorted by the table of the conflicted entry; e.g. on project information, entries, etc. Selecting a single conflict allows the user to solve the conflict screen.

The *Solve Conflict Screen* (cf. Figure 3.9 bottom) lists all important information about the conflicted entry in a diff table: Some values that are unique (like IDs and database IDs) to allow correlating the specific entry, and the conflicted values of the entry. On the left there are displayed the entry values that are currently saved in the local database, on the right there are displayed the ones on the server. The user can select for every conflicted value whether to use the one of server or the one of the local database. A click on the "Solve Conflict" button saves the selection in the local database, that then can be synchronized to the server as well.

In case that users do not know how to solve the conflict they can with a click on the "Inform Project Owner" button let the project owner know about the conflict. The project owner will receive an email about the conflicted entry and then can help the user to solve the problem.

3.6 Discussion

In this chapter, we described a list of requirements that should be fulfilled by e-Science infrastructures for a synchronized distributed data management and data analyses. We discussed existing solutions for synchronized distributed data management in relation to the previously listed requirements. We then 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.

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:

- **Data overhead:** A big data overhead is generated due to the way the data is sent and retrieved from the server. Some additional (and required) information like column names is sent that increases the amount of data that needs to be transferred.

It might be possible to use a better data type for the transfer that reduces the information, like, if a series of entries is sent, the column names are only sent once. This would decrease the size of data, but would require a check to prevent that entries get saved in the wrong column due to a value not being sent or an additional value being sent.

- **Unnecessary data transfer:** Additionally, because of the possibility of an incomplete synchronization, at the beginning of each synchronization process, the entries, that have the same status as the local highest status have, to be sent again. This could mean that many entries that are already in the local database, have to be transferred. The amount of data to be transferred could be reduced by only sending the key of the entries in question. Only if the corresponding entry does not exist locally or has a different timestamp, the complete entry would have to be resent.

- **Detecting duplicate entries:** Currently it is not possible to detect directly duplicate entries due to the local databases that have to work in offline-mode, too. A helpful addition could be to generate a list of similar entries from the global server within the application. This could be used to identify possible duplicates. But since the similarity of entries is context specific, a general best approach cannot be specified.

- **Compression:** Also, since for each entry the rights are checked and one entry is sent at a time, the data is synchronized very slowly in comparison to directly viewing the data in a SQL viewer. A possible speed increase could be achieved by compressing the transmitted data. This should fasten up the transport a lot - especially if large data packages are sent.

- **Improvement of conflict detection:** While the synchronization is able to detect conflicts, it may detect false positives due to only comparing the timestamp and not the actual value that was changed. To improve the detection of conflicts, and also to provide an automatic merge, the synchronization could be enhanced to be able to keep track of changes in tuples, so that each value has its own timestamp and can then be checked for conflicts.

- **Deletion of entries:** Currently entries that are no longer valid cannot be deleted from the database, but are only marked as deleted. So this change can be synchronized to connected databases. This generates a data overhead in the database since all columns of the entry are still occupied. A solution could be a designated table containing only the key entries of deleted entries which are synchronized and then used to delete the entries in the connected databases.

Chapter 4

Retrieving Data: Introducing the Reverse-Mediated Information System

Attribution

This chapter uses material from the following publications:

- Johannes-Y. Lohrer, Daniel Kaltenthaler, Peer Kröger, Henriette Obermaier, and Christiaan van der Meijden. Reverse Mediated Information System: Web-based Retrieval of Distributed, Anonymous Information. In *16th International Conference on WWW/Internet 2017, Vilamoura, Portugal, 2017*, pages 63–70, 2017 [LKK+17]
- Daniel Kaltenthaler, Johannes-Y. Lohrer, Florian Richter, and Peer Kröger. ReMIS Cloud: A Distributed Information Management System for Interdisciplinary Knowledge Linkage. In *8th International Conference on Internet Technologies & Society 2017, Sydney, NSW, Australia, 2017*, pages 107–114, 2017 [KLRK17]
- Johannes-Y. Lohrer, Daniel Kaltenthaler, Florian Richter, Tatiana Sizova, Peer Kröger, and Christiaan van der Meijden. Retrieval of Heterogeneous Data from Dynamic and Anonymous Sources. In *8th IEEE International Conference Confluence 2018 on Cloud Computing, Data Science and Engineering, Noida, Uttar Pradesh, India*, pages 592–597, 2018 [LKR+18]

See Chapter 1.2 for a detailed overview of incorporated publications.

In the previous chapters we discussed the collection of data (cf. Chapter 2) and methods to share and archive data (cf. Chapter 3). These are necessary and useful if the

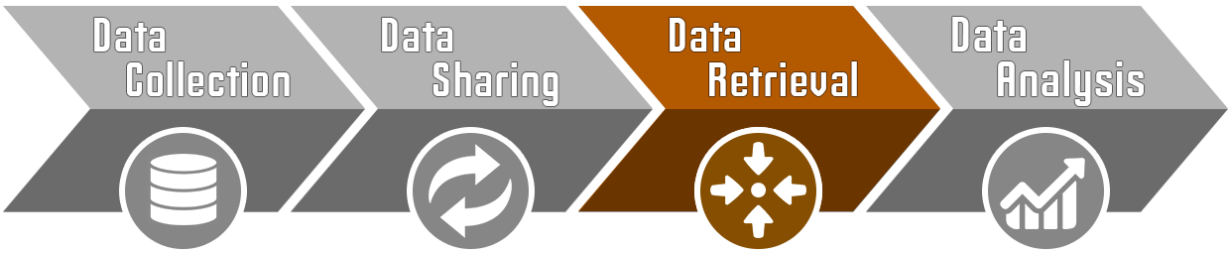


Figure 4.1: *Retrieving data is usually the third step in any scientific workflow.*

scientists work in the same working environment and/or use the same programs. The next step is to retrieve data from different sources (c.f. Figure 4.1).

However, data from one scientific domain is often spread over different institutions, organizations, companies, etc. Still, it might prove useful to, for example, analyze data from an extensive area or to include information from similar domains in an analysis. This data uses different programs and program structures, different database schemes, data sources, data types etc. To consider all this might prove challenging.

Still, to limit the amount of data, and to specify which type of information is desired, some parameters have to be specified which are generic enough so that they are valid for different scientific areas. They need to be specific enough in order to allow to distinguish desired data. These parameters also need to be commonly used so that the information they provide actually is useful.

In this chapter, we introduce a new architecture called Reverse-Mediated Information System (REMIS) that allows the provision of data from data sources by data owners while still remaining control over which data sets to share. Then, we also expand the idea of the architecture to allow a search in different fields to create powerful queries which would otherwise require multiple queries.

4.1 Introduction

Most information is distributed very unstructured over a wide community in many areas. Using crowd-sourcing technologies is a very common trend to utilize the knowledge of many sources. If data is provided uniformly, it can be queried right away and data mining techniques can be applied. In reality, these data sources are used locally and are often very heterogeneous. The reasons are the lack of well-defined standards and the peer-to-peer organized communities of the data suppliers. However, the combination of information systems can yield beneficial insights. If we are able to overcome the variety of data structures and retrieve data dynamically, we can access the knowledge contained in many cross-connections.

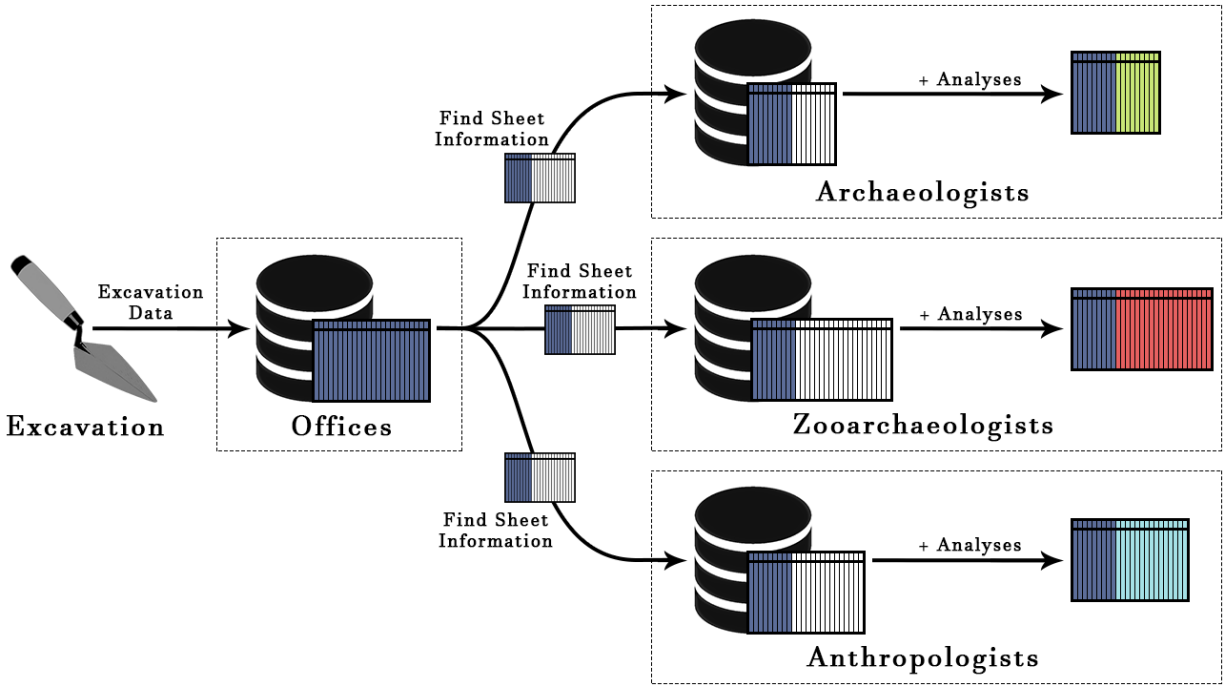


Figure 4.2: *The basic flow of archaeological data: The data from the excavation is partly passed from the offices to the specialized collections and specialists, who perform individual analyses on the findings and save the results in their databases. The results of these analyses are not accessible from the offices. In sum, neither the offices nor the specialized collections have all information about their findings.*

For example, in archaeology and bioarchaeology the different offices, institutes, and freelance researchers each have their individual databases with data from their specific research area (i.e. zooarchaeology, anthropology, or archaeology) as sketched in Figure 4.2. Collating all available data provides versatile analyses. Results from different excavation sites can be compared and set into a broader context, which is not limited to the spatial proximity of the excavation or the field of interest of one research group.

In the medical sector, each clinic uses its own workflow and information system to store patient treatment data. Integrating a state-of-the-art information system (SAP, Oracle) of a medical center and low-tech spreadsheets of a suburban clinic is a complex task. Especially legal and privacy issues can cause the data to become rather volatile. Therefore, databases have to be added or removed dynamically. However, if one is able to combine different medical data sources dynamically, diseases can be detected faster and more accurate, which is worth the effort.

In addition to the research area, a system to embed databases without a centralized manager can also be beneficial for economic purposes. In e-commerce, small businesses often have problems to compete with middle-class businesses and large enterprises. These can offer a wide product range and outperform the smaller players. Keeping the

available goods up-to-date in a central platform is time-consuming and error-prone. A system without a managing instance enables a dynamic decentralized integration of product databases, which offers a great opportunity for the small enterprises. They can collaborate without much effort and compete with large companies.

As a final example, eLearning is a typical crowdsourcing application. Teachers, lecturers, and students have access to many exercises, exams, data sets, sources, and documents. Offering a dynamic system, which allows each person to register his unstructured data into a large unmanaged information system is beneficial for every learner in the community.

To the best of our knowledge, there is no architecture that supports this kind of decentralized connection system for information retrieval. The well-known Mediator-based architecture [Wie92, Wie94, Wie13] supports only a centralized managed way of heterogeneous databases, as sketched in Figure 4.3. This architecture depends strongly on the knowledge of the data sources and their existence to access the contained information. In an agile and dynamic environment like the mentioned cases above, this is a big disadvantage, which we want to overcome with our approach.

Our novel approach, the Reverse-Mediated Information System (REMIS), is based on the concept of Mediator architectures. As sketched in Figure 4.4, it enables the provision of new data components without the need of a central administrator to manage the connections manually. Thereby, the data owners keep control over the third-party data access. It enables the users to search for more information about a specific context that is spread through a diversity of databases. In summary, the main contributions are as follows:

1. Modeling of an architecture (REMIS) to reverse-mediate a set of data sources
2. Decentralized and anonymous database registration as a server application
3. Client-side connector application to translate the user request depending on the data source

The remaining chapter is organized as follows: First, we demonstrate the problem a distributed architecture faces on the example of archaeology in more detail (cf. Chapter 4.2). Then, we list a set of requirements that are extracted from these problems. These should be addressed by an architecture for distributed data management of anonymous data sources (cf. Chapter 4.3). Then, we describe the concept of the REMIS architecture, including the process of the initialization, registration, and the user search (cf. Chapter 4.4), and describe the concept of a compact rights management for the data (cf. Chapter 4.4.4). Afterwards, we explain the necessary configuration for administrators of data sources (cf. Chapter 4.4.5). Then, we compare existing solutions and

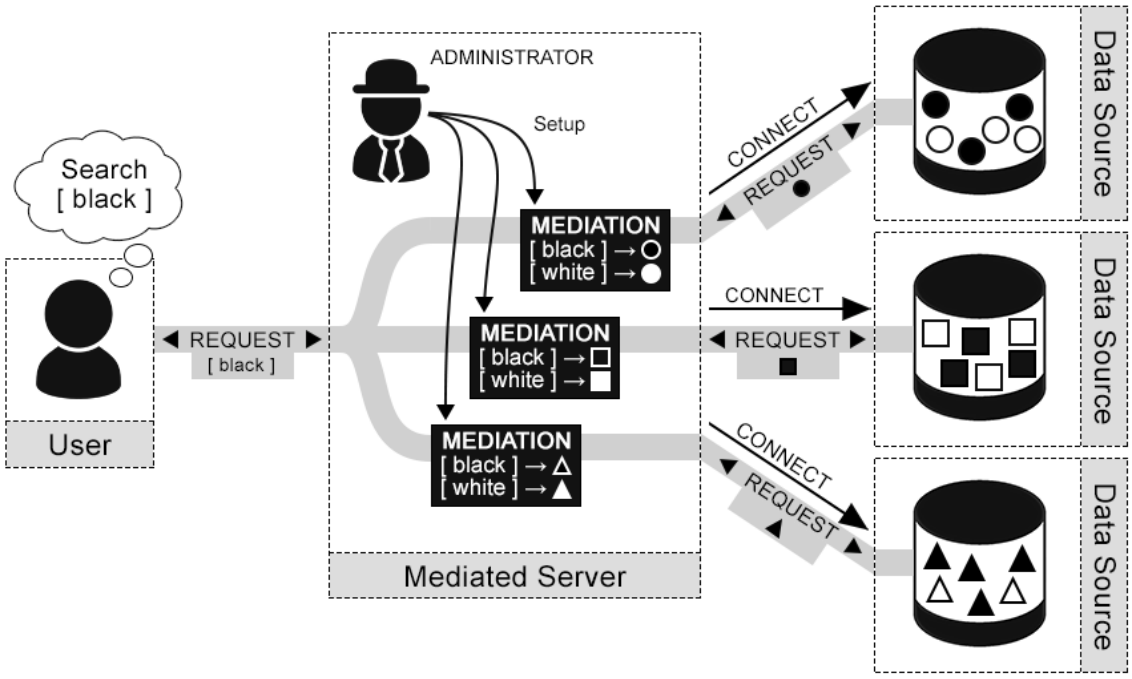


Figure 4.3: *Sketch of the well-known Mediator-based architecture. A central administrator is required to connect the data sources and to mediate the requests from the user. The administrator has to know each data source to be able to connect them.*

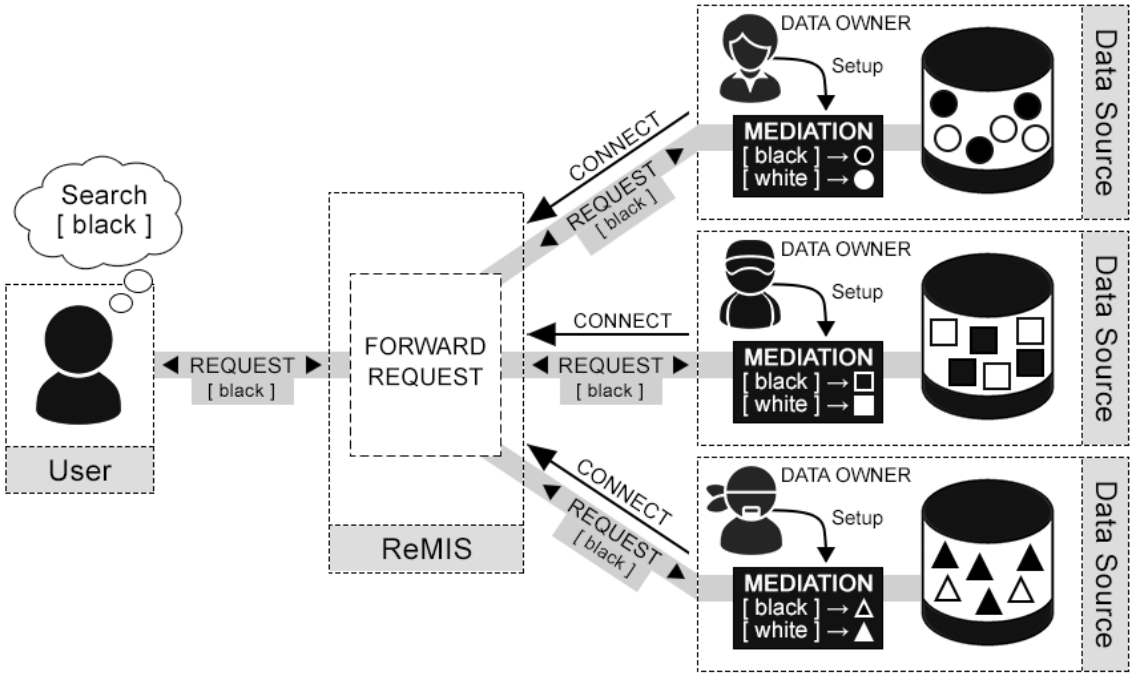


Figure 4.4: *Sketch of the Reverse-Mediated Information System. The data owners can register their databases to the system on their own. The necessary mediation setup is executed by a wizard dialog. The architecture forwards the user request to the data sources where the request is mediated.*

approaches for systems for distributed data management with the requirements and the REMIS architecture (cf. Chapter 4.5). Based on that, we introduce an extension called REMIS CLOUD (cf. Chapter 4.7). REMIS CLOUD allows the combination of different categories to allow querying information which is otherwise not trivial to achieve. We present some use cases for the REMIS CLOUD to demonstrate the potential application area for this architecture.

4.2 Problem Formulation

For many scientific analyses it is not only interesting to analyze data from a single sub-domain or location. Considering the data from other sub-domains and other locations would enable more complex and more comprehensive analyses than currently possible.

In principle, this data is often available, but spread over different databases that are run by different institutions. While in some research areas like biology exist some centralized platforms for sharing information like Global Biodiversity Information facility (GBIF)[25], Atlas of Living Australia (ALA)[26], Virtual Biodiversity Research and Access Network for Taxonomy (ViBRANT)[27], DataONE[28], and the US Integrated Digitized Biocollections (iDigBio)[29]. However, these are not always suitable for everybody, as they often have a limitation on scope.

Therefore, the problem of distributed databases also is valid in these areas. In practice, data is not easily accessible for every scientist because of the following reasons.

- **No direct access:**
  There is no direct external access available to data of most of the specialized databases, as sketched in Figure 4.5. Data has mostly manually to be requested which means the composition of this data occurs requesting a specific, usually generally available attribute or set of attributes. Obviously, a more detailed data selection with complex search parameters is difficult to apply without any direct access.

- **Anonymous databases:**
  Data is spread over databases that – in some circumstances – are not known at all. Scientific data is not always gathered by well-known institutes. There are also numerous smaller institutions, clinics, collections, and freelancers that are gathering data as well, but do not exchange their data with the state offices. Still, it would

[25] http://www.gbif.org
[26] http://www.ala.org.au
[27] http://www.vbrant.eu
[28] http://www.dataone.org
[29] http://www.idigbio.org

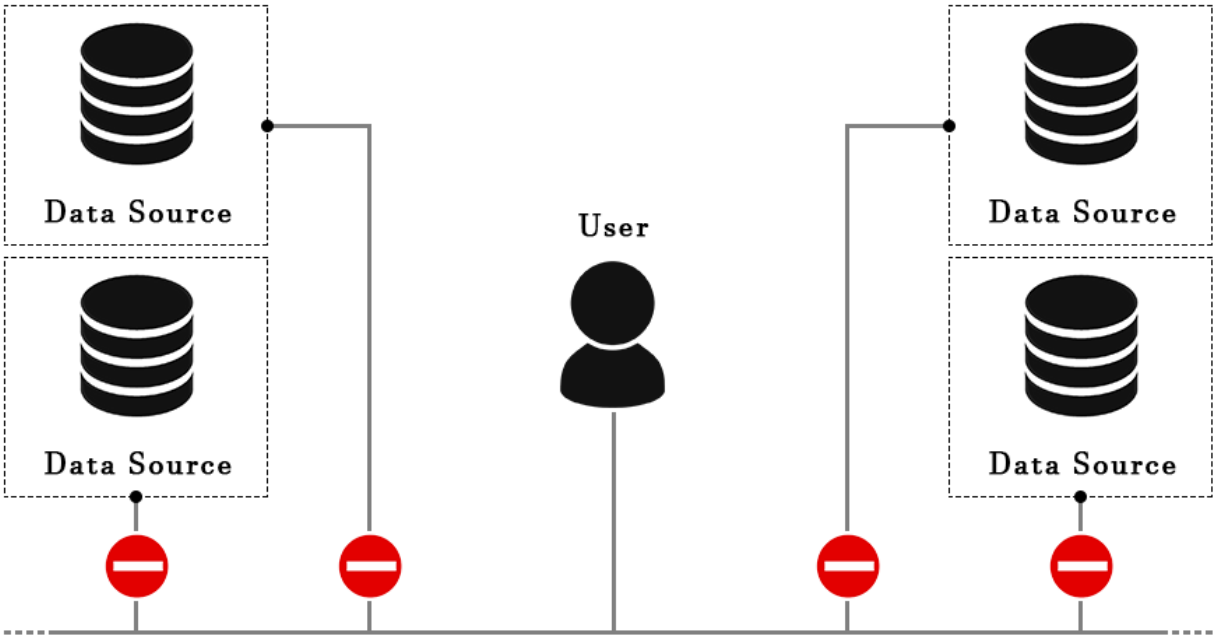


Figure 4.5: *The digital data exchange is hindered. Institutes and specialized collections and specialists each have no digital access to the detailed or individual data of other databases.*

be interesting to be able to include this data in analyses, even if the archaeologist does not know that there exists any data in these data sources.

- **Individual database schemes:**
  The databases of the institutes, the specialized collections, and self-employed workers have their own architectures and individual database schemes. Although they usually have identical basic information, these are saved in different ways and cannot be standardized to a commonly used database scheme.

- **No consistent way to exchange data:**
  As it is common practice, data is exchanged by using Excel or CSV files that have to be formatted before being able to use them for analyses. This method can be aggravating, time-consuming, and error-prone, but it is still common practice.

- **Regional and individual approaches:**
  There are regional and individual approaches for the filing of findings. They do not only vary in the way they are sampled, but also how and which data is recorded in the databases. This differs from country to country and even from state to state which is extremely frustrating for researchers that deal with cross-border research. Furthermore, there are numerous individual solutions even on a local level.

Especially the consideration of geodata, if applicable, would enable users to ask much more complex questions than it is currently possible. Therefore, we want to describe a solution to bring spread data together again and to achieve this without the need of a single central database, but by keeping the necessary system of distributed databases.

4.3 Requirements

We define a set of primary requirements that should be considered for our solution. This is described below.

Our architecture hast to dynamically connect new data sources that are heterogeneously distributed. Each of them can have their own data scheme and contain different types of data. Additionally, the physical location of the data is not known in advance, neither by the users nor the responsible administrators of the architecture. Therefore, it has to be possible to connect to data without having to update the information of the central server.

We state these requirements as R1 and R2:

R1 *Heterogeneous Data Sources.*

R2 *Anonymous Data Sources.*

External data sources have to be dynamically connected and added to the architecture. This process shall be performed individually by the owners of the data source. Therefore, no central administrator should be required to connect a data source to the existing information system. The access to the data by third-party users should be controlled directly by the data owners – i.e. they should be able to connect, and disconnect their data source at any time, and also control which data they provide.

We state this requirement as R3:

R3 *No central administrator.*
Administration directly by data owners.

The users have to be able to search through all connected data sources independently of their underlying database scheme. The database scheme must not influence what the users can search, but instead map the search parameters to the local database scheme.

We state this requirement as R4:

R4 *Database Scheme Independent Search.*

Finally, data sources can contain sensitive information that is not intended for everyone. Still, data sources may contain searchable data. Therefore, the architecture has to provide the possibility to restrict the access to sensible information or data that shall not be made public (yet).

We state this requirement as R5:

R5 *Rights Management.*

Let us add that the data of the distributed databases is not intended to be edited by the users. This infrastructure is solely aimed for retrieving data.

In the following chapter, we describe our novel information retrieval system which respects the defined requirements stated above.

4.4 Reverse-Mediated Information System

In this chapter, we want to describe the basic concept of the Reverse-Mediated Information System to achieve a search through different, anonymous databases. Therefore, we distinguish the three different layers of the architecture as sketched in Figure 4.6:

- **User Layer:**
  The users who search for specific data and want to retrieve the information corresponding to the entered parameters. This can be a web service, but also a search mask embedded to an existing application.

- **Server Layer:**
  Runs a stand-alone mediator-like Server Application which accepts requests from the users and relays these to the connected databases. In addition, it receives the result from the connected databases and sends them back to the users. New databases can register themselves to the Server Application.

- **Data Layer:**
  The server where the data sources, mostly databases, are saved. Each data source runs an independent Connector Application that is configured for the database it is connected to. This Connector Application accepts forwarded requests from the Server Layer and translates the request to a query to fetch the data from the data source. Once the data is fetched, it transforms the data to a uniformed data structure which will be sent back to the Server Application of the Server Layer.

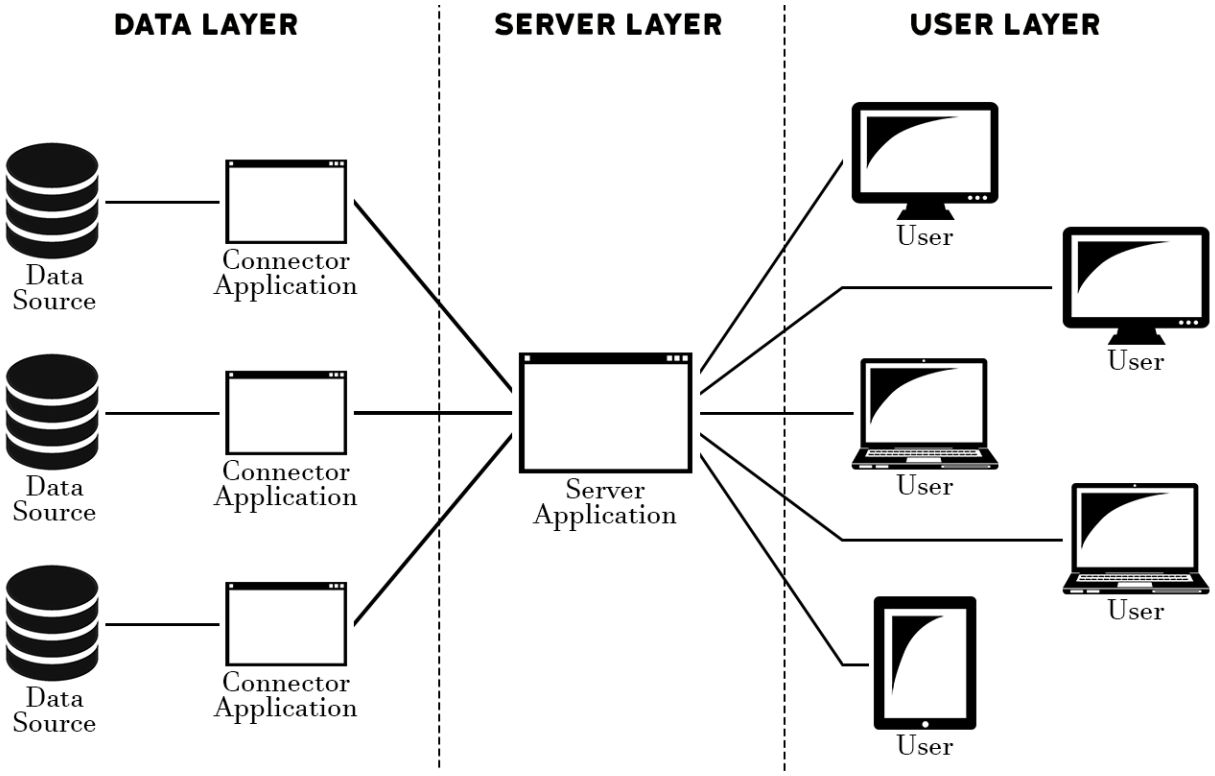


Figure 4.6: *The three layers of the architecture: Data Layer, Server Layer, and User Layer.*

In general, a Mediator provides data from a number of heterogeneous data sources with other users. “The autonomy of the participants enables the overall system to grow, since new sources [. . .] can be inserted.” [Wie94] Still, a central administrator is necessary to manage the participating data sources. Therefore, an extern connection of a data source is not possible without the integration work of the administrator.

The Reverse-Mediated Information System swaps this approach to achieve an open system for shareable data. Data sources can be connected to the system to share data with other users, independent of the actual data type.

The usage of the Reverse-Mediated Information System can be categorized in three different steps to describe the workflow that are described in detail below.

4.4.1 Initialization

To be able to connect the data of a specific data source, an initialization process has to be executed for the data source in the Data Layer. The administrator of the server, where the data source is located, is guided by a setup assistant. In general, the required steps in the initialization process can depend on the type of the connected data source. Hereinafter, we use a relational database as an example for a data source. A sequence diagram of the

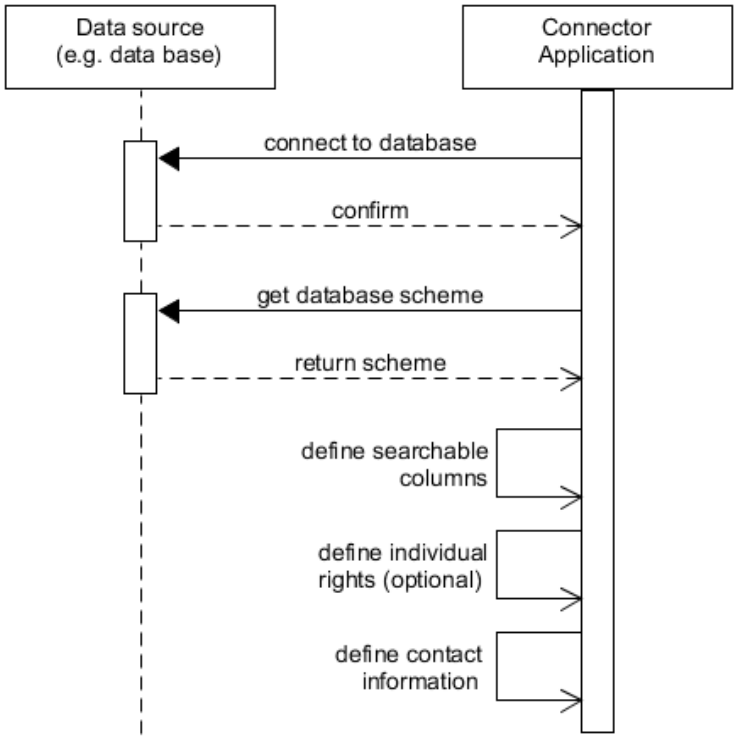


Figure 4.7: *Sequence diagram of the initialization process.*

initialization process is shown in Figure 4.7.

First, it is necessary to connect the Connector Application to the database in order to be able to access the data. Once this is done, the administrator can select the tables and columns to define which data will be sharable in general. This step determines which parts of the data are private and only visible within the database itself and which data might be transmitted to the users later.

Optionally, individual rights can be defined that specify the visibility of the data for the users. These are discussed in Chapter 4.4.4.

Finally, the administrator has to specify a set of contact information. That allows either contacting the data owners for questions, or to request additional information of entries that cannot be accessed due to rights settings.

4.4.2 Registration

To become a part of the Reverse-Mediated Information System, the initialized Connector Application has to be registered to the Server Application in the Server Layer. The sequence diagram of the registration process is shown in Figure 4.8.

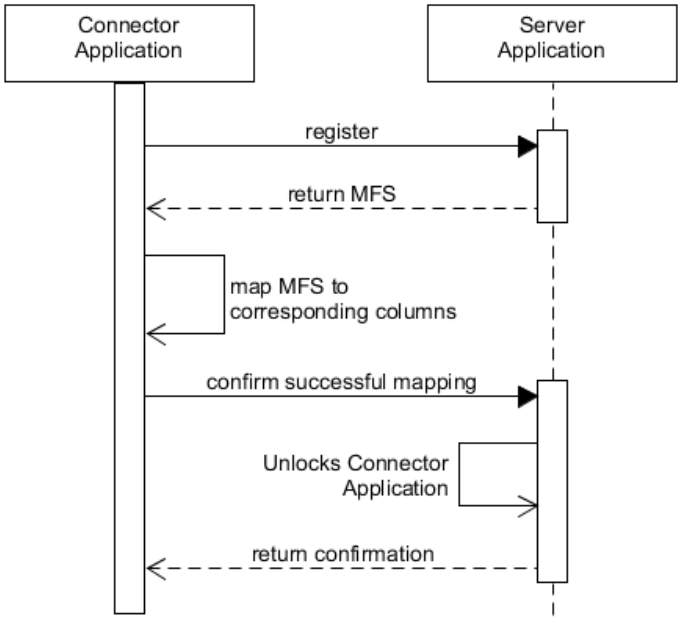


Figure 4.8: *Sequence diagram of the registration process.*

First, a "register" command is sent to the Server Application. The Server Application then sends the "Minimal Search Parameter", abbreviated with *MSP*, to the Connector Application, a set of minimal information that are required. These are as minimal as possible and guarantee that every connected data source has saved any information about each of the parameters. The Minimal Search Parameter also forms the options the user can search for.

$$MSP = \{p_1, \ldots, p_n\},\ n = |MSP| \tag{4.1}$$

In general, the parameters p of the Minimal Search Parameter are defined in the Server Application which has access to all connected Connector Applications and their databases. Each database D can have an individual database scheme with its own columns, but every parameter of the Minimal Search Parameter has to be mapped to a specific column in the database. Each database consists of a set of columns c.

$$D = \bigcup_{i=1}^{m_D} \{c_i\},\ m_D \geq n \tag{4.2}$$

For the registration process, the parameters are sent to the corresponding Connector Application. Then, the administrator has to map each of the parameters p of the Minimal Search Parameter to the corresponding columns in the database. To describe this mapping, we define the injective function κ_D that stands for a projection of the parameters of MSP to the corresponding columns in the database D.

$$\kappa_D : MSP \to D \tag{4.3}$$

We define r as the user search request. These may have different appearances, for example tuples containing the parameters p and the corresponding values v of the user inputs.

$$r = (r_1, \ldots, r_k),\ r_j = (p_j, v_j),\ j \leq n \tag{4.4}$$

Furthermore, we define $\tilde{\kappa}_D$, a replacement function as canonical extension over the query language, while all search parameters $r \in MSP$ are mapped according to κ_D and the remaining parts stay unchanged. Let $\epsilon = ()$ the empty search request.

$$\begin{aligned} \tilde{\kappa}_D(\epsilon) &:= \epsilon \\ \tilde{\kappa}_D(r \neq \epsilon) &:= \begin{cases} \kappa_D(r_1) \circ \tilde{\kappa}_D((r_2, \ldots, r_k)), & \\ \qquad\qquad\qquad\qquad r_1 \in MSP & \\ r_1 \circ \tilde{\kappa}_D((r_2, \ldots, r_k)),\ r_1 \notin MSP & \end{cases} \end{aligned} \tag{4.5}$$

Finally, r_D is yield by applying κ_D as a replacement function to r:

$$r_D = \tilde{\kappa}(r) \tag{4.6}$$

As soon as the mapping of all Minimal Search Parameter values is complete, the Connector Application informs the Server Application that it is ready to accept requests by the user. To complete the registration process, the Server Application set the Connector Application to be ready to be searched.

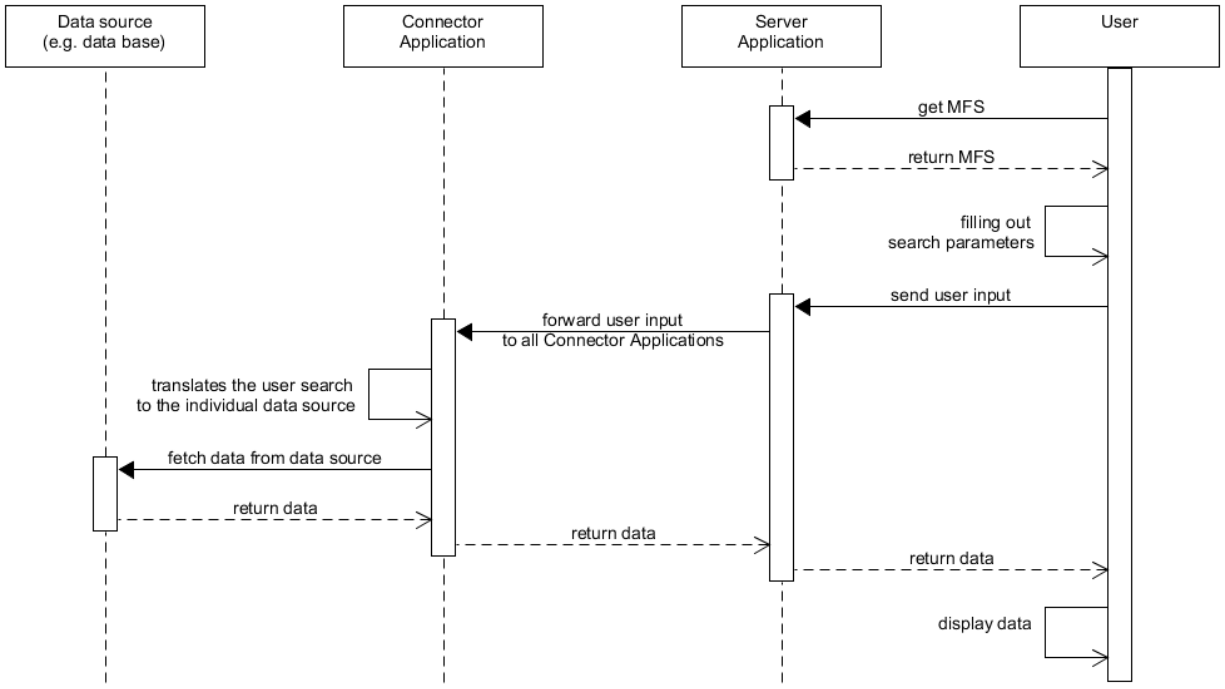


Figure 4.9: *Sequence diagram of the process to fetch data from the single data sources.*

4.4.3 Search

The existing federated systems are sufficient if the users need to query data from a fixed set of (internal) databases, like sale statistics or customer information. But now, we want to provide the users the option to search for information, they might not even know that it exists.

For this, we define the Minimal Search Parameter being the set of information the user can search for by default. All connected Connector Applications know how to handle these information.

Certainly, the right settings set during the initialization process have also to be considered. Since the right management has to be configured according to the underlying data structure and data, we do not discuss the implementation of the rights management in detail in this thesis. The process of fetching data from the databases is figured in the sequence diagram in Figure 4.9.

The search enables the retrieval of entries that include values which are defined as the Minimal Search Parameter. Since this Minimal Search Parameter information is saved on the Server Application, the Server Application is first queried to get the Minimal Search Parameter MSP to be able to display its information in a search mask for users. This is necessary to avoid input masks that are embedded to any applications getting incompatible when the definition of the Minimal Search Parameter changes. The users

enter the data and define which parameters MSP* they want to use for their search. For these applies:

$$MSP* \subseteq MSP \tag{4.7}$$

The parameters MSP* and their inputs are sent to the Server Application that forwards the request to all N connected Connector Applications. Each Connector Application translates the incoming parameters to the local scheme of their corresponding databases, considering the given mapping κ_D. Each Connector Application queries the database with the modified search request r_D and checks if the search request of a user (this means, the inputs of the parameters MSP*) matches with the corresponding columns. Let $R(D, r)$ the result of this request.

$$R(D, r) = \sigma_{\tilde{\kappa}_D(r)}(D) \tag{4.8}$$

The entries that have been found in the database are then returned back to the Server Application. Finally, the user gets the merged data σ_r^{Σ} from the Server Application considering the search request r:

$$\sigma_r^{\Sigma} = \bigcup_{i=1}^{N} R(D_i, r) \tag{4.9}$$

The responds from the single Connector Applications are not bundled within the Server Application, but are sent directly to the users. This improved the necessary waiting time for the users, especially if a database has a slow connectivity or currently no connectivity at all.

4.4.4 Rights

We now have established the Reverse-Mediated Information System for users to be able to query for data from databases they did not even know exists. This architecture was explained in Chapter 4.4. Since we want to encourage database owners to provide their data, we have to provide a right management for their data. Scientists want to be able to manage the visibility of their data, not only because data of unpublished work often

should not be shared until the time of publication, but also because parts of the data must not be shared at all, like sensitive data.

Therefore, we provide the possibility to limit the visibility of the data (or parts of it) of their database. The first step for the right management was already defined during the initialization process, as described in Chapter 4.4.1. The data owner defines which data (tables and columns) is set private and is not accessible for the Connector Application at all.

As well, the data owner can set individual rights that specifies the visibility and searchability of the data for the users. Therefore, we distinguish five different types of right that can be applied to the entries. These specify the level of detail of the returned values that include the range from non-restricted at all to most restrictive.

- **Full-Public:**
  The default right which is used if no restrictions were applied to the entry during the initialization process. All columns of the entries are returned that are defined in the initialization process.

- **Partially-Public:**
  Limits the returned columns according to criteria specific to the data set. Still, all columns can be searched by the user, but not all of them are returned as a result. This can be achieved by excluding single columns from the visibility, but can also be a more complex query, like checking if a specific value is set (or not) to a column.

- **Numeric-Public:**
  Setting this right would effect that the entries are searchable, but only the count of the found entries is returned. The details of the information about the entries is not communicated with the users.

- **Hidden:**
  To reduce the visibility even more, the entries can be hidden. A search request would only return a boolean value whether any matching data was found, or not. The details, how many data sets are found and any further information are hidden from the users.

- **Private:**
  No information about a data set is returned at all. All data from the database is completely private and cannot be searched.

In general these rights can be set on single data sources like tables in databases. But if the data source has saved right information within the data sets, these can also be used

to define entry-specific rights. Therefore, the data of the result set can be put together by considering different rights.

In any case, the data owner can be contacted by the user, e.g. for further questions for the request to access more detailed information about the data set. This is especially interesting for the numeric-public and hidden rights. Therefore, the contact information is used that was defined during the initialization process.

4.4.5 Configuration of the Connector Application

Since the configuration of the Connector Application is the most important and responsible task of the Reverse-Mediated Information System, we describe the practical configuration process in more detail. Here, the owner of the data source defines the settings for the data and the server, and for the basic privacy settings for the data. Therefore, administrators are led by a wizard dialog to setup the application.

First, the data owner has to select the type of the data source. Currently MySQL databases and Excel files are supported. The Connector Application can be extended by other types to be supported in future. In the next step, the data owner has to establish a connection to the data. This depends on the selected type of data source. In case of a MySQL database, the administrator has to enter connection details to the database, like database name, port, user, and password. Then the Connector Application connects to the desired database and the available tables are shown in the setup dialog. In case of Excel tables, the data owner has to select the desired files, where the data is stored, in a file dialog.

Then, the Connector Application retrieves the Minimal Search Parameter from the Server Application. Normally, the column names in the data sources are not identical to the corresponding parameters of the Minimal Search Parameter – reasons for that might be a varying naming of the columns or different languages. The data owner has to map each parameter of the Minimal Search Parameter to a specific column in the database, respectively the Excel file. A screenshot of this dialog is shown in Figure 4.10.

The mappings (“Minimal Search Parameter” $\rightarrow$ “table name” / “column name”) are saved in a XML configuration file on the server. On an incoming search request, these mappings are used to translate the Minimal Search Parameter to the corresponding column information.

The next step in the dialog is a listing of all available columns, where the data owner can select all those columns which data should be displayed to the users. This screen is shown in Figure 4.11.

Only selected columns are considered when composing the result set. Though, the data owner can use this screen to set specific parts of data to private which will be communicated neither to any user nor to the Server Application.

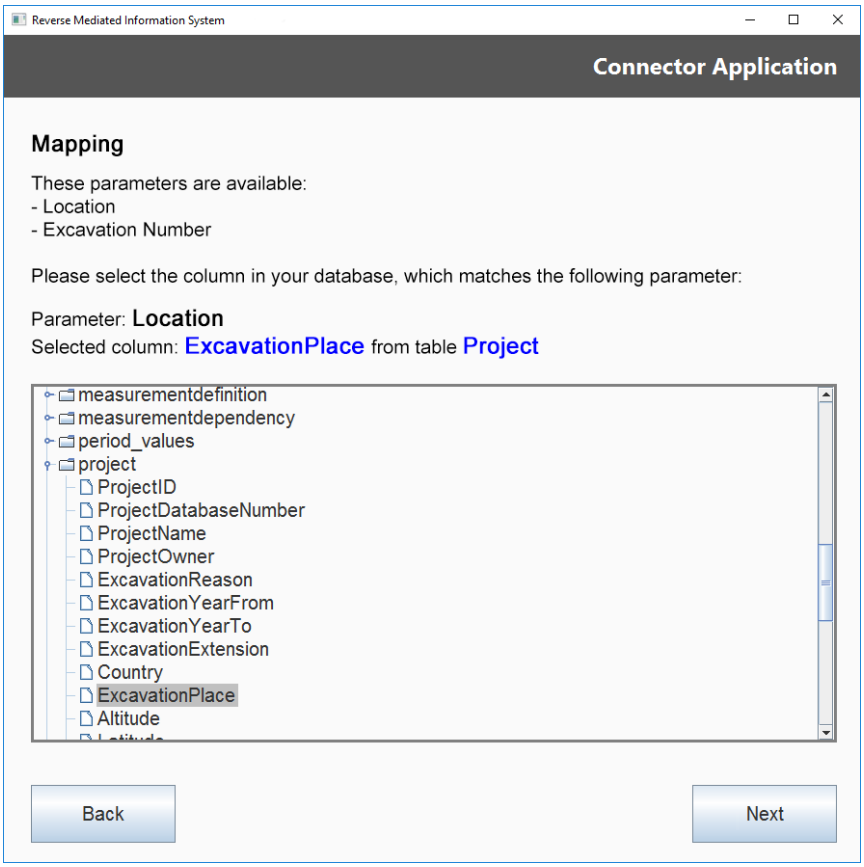


Figure 4.10: *Screenshot of the wizard dialog where administrators can map the specified parameters of the Minimal Search Parameter to the actual data of their database.*

If data is not saved in one single table, e.g. by using IDs and value tables, it is necessary to define dependencies between tables within one database. The data owner should define foreign keys for each table. Multiple foreign keys are also allowed. The screen of these dependencies is shown in Figure 4.12.

The settings for the result sets and the dependencies are both saved in the XML configuration file together with the mapping information of the Minimal Search Parameter.

4.5 Related Work and Comparison

In the traditional database search, information is stored in single databases that are managed by Database Management Systems (DBMS) – but these do not meet our requirements R1, R2, R3, and R4. In general, the idea of merging data from numerous (physical or virtual) databases, is not new. In the following, we recapitulate existing approaches for such infrastructures in the context and compare them with our requirements derived in Chapter 4.3.

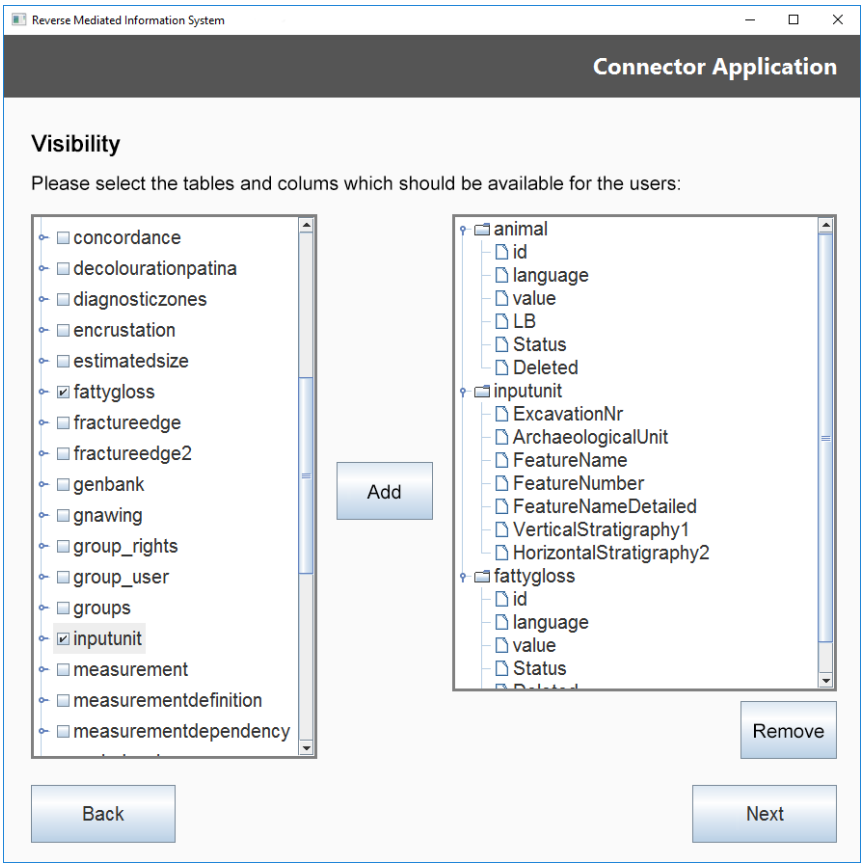


Figure 4.11: *Screenshot of the wizard dialog where administrators can determine the columns from the connected data source which should be communicated and transferred to the user.*

- **Data Warehouses:**
  Data is stored in different databases to be able to store more and bigger amounts of data, or to separate logical information. Each database may hold different sets of information. To make these heterogeneous databases searchable, the information is merged in a Data Warehouse, a central repository that retrieves and updates the integrated data from the diverse sources. The users only query the central repository.

  Data Warehouses do not require having to update existing databases to store new types of information, but they produce a large stream of data to query and update the central database if data is updated frequently in one of the heterogeneous databases. [Inm05]

  In regards to our specified requirements, Data Warehouses violate the requirements R2 and R3.

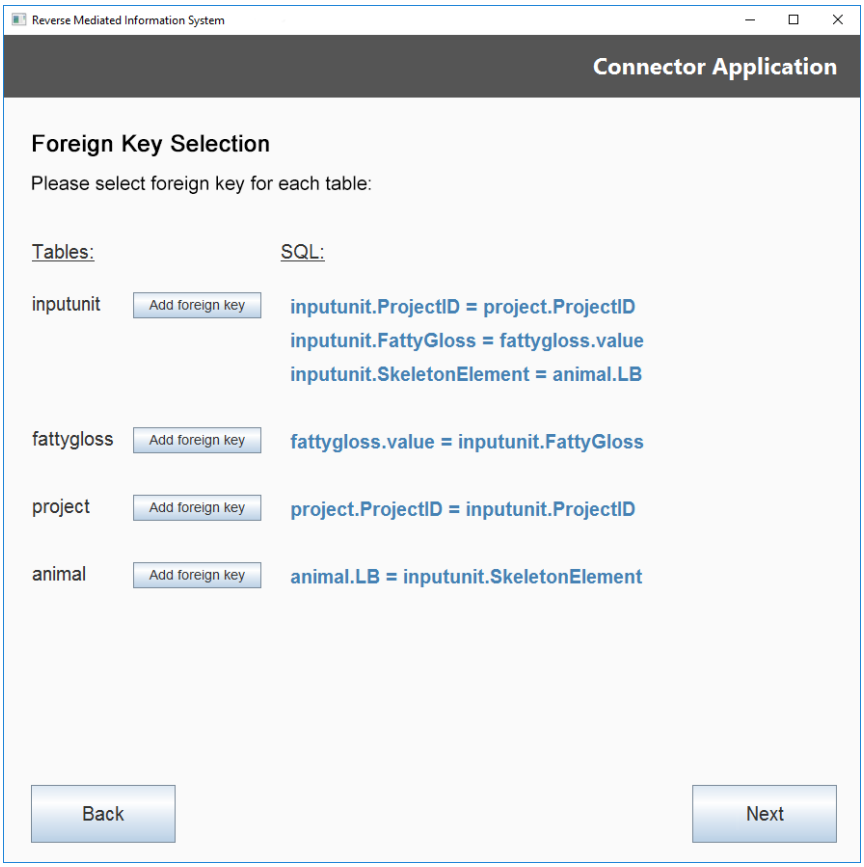


Figure 4.12: *Screenshot of the wizard dialog where administrators can define foreign keys to define related columns in separated tables, e.g. for the use of IDs and value tables.*

- **Federated Information Systems (FIS) / Federated Database Management Systems (FDBMS):**
  These systems map multiple database systems into a single federated database. There is no actual data integration in the consisting disparate databases as a result of data federation. The virtual data of the FIS/FDBMS is queried directly by the user. It decomposes the query into subqueries for each of the federated databases. Finally, it composites the result sets of the subqueries. [SL90, HM85] Here again, the requirements R2 and R3 are violated.

- **Mediator:**
  An extension to the Federated Information Systems is the Mediator concept, “a software module that exploits encoded knowledge about some sets or subsets of data to create information for a higher layer of applications” [Wie92]. Mediators allow the addition of new databases unknown to the user by adding a mediator layer which is responsible to translate the queries to the designated database. While the user can still query data without knowing the amount and structure of the

queried databases, the Mediator needs to know all connected databases. If a new database is added, the Mediator is extended. [Wie94, Wie13]

The mapping of the mediation within these systems can basically be implemented in two different ways [Lev00]:

- **Global-as-View:**
  The global data scheme is the sum or union of the local schemes. For this the schemes of all connected data sources are taken and a global scheme is defined, either by using all entries in the scheme or by only using the entries that are available in all data schemes.

  However, this requires that the global scheme has to be updated as soon as the data scheme of a local source changes. This approach is not suitable for systems in which not all sources are known in advance, data sources can be added or removed, or data schemes continuously evolve. Otherwise it can pose a risk of asynchronicity and incoherence with the applications that are using the platform, since it can not be guaranteed that the local scheme of the data sources stays compatible to the global scheme.
- **Local-as-View:**
  In the Local-as-View concept, the global scheme is defined independently of the connected data sources. Instead, a mapping from the global to the local schemes has to be defined. This method is preferred to the Global-as-View approach since the global scheme is only defined once and only the mapping to the corresponding source has to be updated if the scheme of the local sources changes.

However, the Mediator concept needs a central administrator who connects existing databases to the system which does not satisfy R3.

- **Apache Hadoop:**
  A different approach (which is very commonly used to store large amount of data) is Apache Hadoop. It is a framework that allows the usage of a network of many computers to solve problems involving massive amounts of data and computations. The default file system, called Hadoop Distributed File System (HDFS), is a distributed, scalable, and portable file system. It usually stores large files in the range of gigabytes to terabytes [Fou] across multiple machines.

  Still, Apache Hadoop duplicates the data on multiple data sources and, at the same time, all data sources are managed by a central instance. This means Apache Hadoop violates all of the our defined requirements stated above.

Different applications exist with different approaches to allow the retrieval of data sets from distributed sources. In the following, we want to give an overview of some of the applications and how they work.

- **CLARIN:**
  The Common Language Resources and Technology Infrastructure (CLARIN) is an infrastructure that "aims to provide easy and sustainable access for scholars in the humanities and social sciences (HSS) to digital language data (in written, spoken, video or multimodal form) and advanced tools to discover, explore, exploit, annotate, analyze or combine them, independent of where they are located." [KH14]. CLARIN uses data centers which are distributed all over Europe. These 'CLARIN centres' store reference data sets – to be used for analyses – and also allow the creation of new resources which have to be mapped to appropriate data formats and be described with metadata. This approach allows querying (text) data in a standardized way. However, this method violates the requirements R1, R3, and also R4.

- **SchizConnect:**
  SchizConnect [WAC+16] allows the querying of multiple data repositories for schizophrenia neuroimaging studies with the help of mediators. New data sources can be added, and with the help of mediators their model can be mapped to the global SchizConnect domain model. This allows users to query data with the global terms. The search is translated to the individual data schemes and the databases can be queried concurrently. However, the connected databases have to be connected with the SchizConnect Mediator (a central platform) for this approach. SchizConnect is built upon XNAT [HHO+16], an open-source imaging informatics platform. Due to using the mediator approach, requirements like and R1 and R4 are fulfilled, however, SchizConnect violates requirement R3.

- **Neuroscience Information Framework & NIF DISCO:**
  There are other solutions that provide a broad range of capabilities to store data in distributed locations. As an example, in the neuroscience disciplines the Neuroscience Information Framework (NIF) [GAA+08] is available to handle data. The extension NIF DISCO is a framework to facilitate the automated maintenance of the connected data sources. Whenever "a resource changes the scope of its contents, the resource developers can make corresponding changes to a local DISCO file describing their resource. The information in this file is then "harvested" by a central DISCO server on a regular basis and incorporated into the NIF Registry entry describing the resource." [MWSM10] The concept of the DISCO framework helps data owners to describe and to update their scheme by updating a registry

file. Therefore it fulfills most of our requirements. However, a rights management system is missing, therefore requirement R5 is violated.

- **The Australian Repositories for Diffraction Images:**
  The Australian Repositories for Diffraction Images (TARDIS) [ASB+08] was created as a solution to store large X-ray diffraction images. It uses a federated approach to distribute the uploaded images over several databases. The images can be uploaded from a central website along with their standardized metadata. Later, users can search through the databases for the metadata to get the desired information. Other applications are also built upon the TARDIS data management system, like the Online Ancient Genome Repository (OAGR) [30], an open access repository for ancient human DNA data. However, since the data is collected from a central platform, this violates several of our requirements like R1, R3, and also R4.

In contrast to these existing approaches, the Reverse-Mediated Information System fulfills all defined requirements from Chapter 4.3. REMIS connects heterogeneous data sources (R1) whose physical location is not known beforehand (R2). There is no need for a central administrator (R3) and the data sources can be searched independently on the database scheme (R4). The provided user rights management allows the data source owners to protect (maybe sensitive) information from unauthorized access (R5).

4.6 Case Study: Retrieving Archaeo-Related Information

For a case study, we use real data of three database applications in archaeo-related sciences: OSSOBOOK [KLK+18a], EXCABOOK [KLK+18e], and ARCHAEOBOOK [KLK+18d].

For each of these applications a Connector Application was configured to connect the data sources to the REMIS architecture. For our case, we required the research field "Archaeology" to be supported. The Server Application was initialized to require the Minimal Find Sheet information as the values for the Minimal Search Parameter. These are values that are always known in an excavation. They include the Feature Number of the excavation (which is centrally assigned by the *Bavarian State Office for Monuments and Sites*[31]), the Find Sheet Number, and location information (excavation place and x-/y-coordinates of the excavation).

The interested user can use the REMIS web interface to request data from the connected data sources, but they do not need to know that they exist. The web interface is shown in Figure 4.13 (the search mask) and in Figure 4.14 (the result view).

[30] http://www.oagr.org.au
[31] http://www.blfd.bayern.de

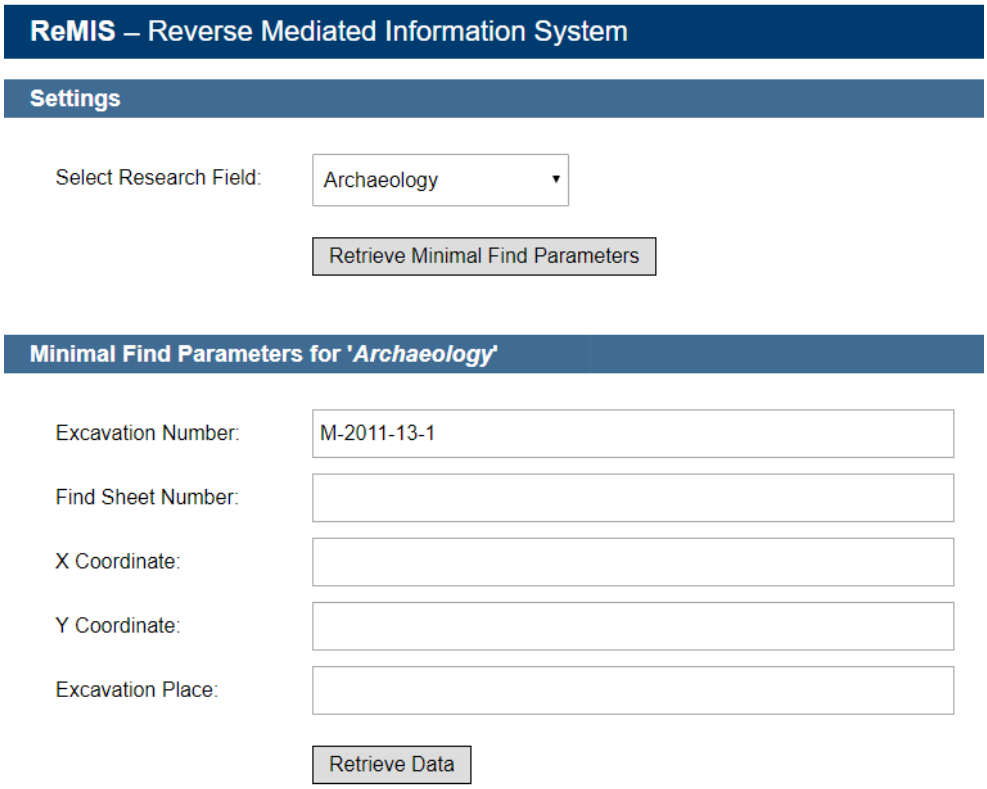


Figure 4.13: *Screenshot of the search mask of* REMIS.

The search mask offers the opportunity to select the research field "Archaeology". Then the corresponding Minimal Search Parameter is loaded and can be entered which actually consist of the values of the Minimal Find Sheet.

We want to retrieve information about the excavation "Marienplatz-Haltepunkt" in Munich, Germany, of which we know the excavation number "M-2011-13-1". We enter this number into the corresponding input field of the web interface and click the "Retrieve Data" button. In the background, the REMIS architecture will now query information from all connected databases – this means the Server Application will send the user request to the Connector Application of each of the registered databases. In our case, the Connector Application of OSSOBOOK, EXCABOOK, and ARCHAEOBOOK will each receive the user request and will then translate it to the local database scheme considering the mapping of the configuration. Then the query with the translated mapping is executed, and the retrieved information is sent back to the Server Application.

Then, the result is passed to the web interface which displays the information in table form to the users. Due to the different data schemes of each data source, the information from each data source is displayed in an own, collapsible block. In our search, we received information from the databases OSSOBOOK and EXCABOOK which

ReMIS – Reverse Mediated Information System

Search Parameters

Back **Research Field:** Archaeology
Excavation Number: M-2011-13-1

Print Export XLS Export XLSX Export CSV

OssoBook

ExcavationNumber	ArchaeologicalUnit	FeatureName	FeatureNumber	FeatureNumberDetailed	VerticalStratigraphy1	VerticalStratigraphy2	HorizontalStratigraphy1	HorizontalStratigraphy2	AddInfoArch
M-2011-13-1	4014	Stadtgraben	1229	Verfüllung	Schnitt 5		Fläche 4		Abschnitt 1 Süd, siehe auch Fz 3898
M-2011-13-1	2630	Schacht 11	1470	Verfüllung	Schnitt 11		Fläche 5	Planum 2-3	Abbau Planum 2 zu 3; Schlämmen
M-2011-13-1	2969	Stadtgraben	1508	Verfüllung	Schnitt 5	Profil 120	Fläche 4		Abbau Profil
M-2011-13-1	1760	Schacht 5	360	Verfüllung	Schnitt 3 W		Fläche 1		
M-2011-13-1	2131	Stadtgraben	1229	Verfüllung	Schnitt 5		Fläche 4	Anlage Planum 2	
M-2011-13-1	2630	Schacht 11	1470	Verfüllung	Schnitt 11		Fläche 5	Planum 2-3	Abbau Planum 2 zu 3; Schlämmen
M-2011-13-1	2969	Stadtgraben	1508	Verfüllung	Schnitt 5	Profil 120	Fläche 4		Abbau Profil
[...]									
M-2011-13-1	2106	Stadtgraben	1016	Verfüllung	Schnitt 5		Fläche 4	Planum unter 1	Abbau Befund unter Planum 1
M-2011-13-1	1344	Schacht 5	746	Verfüllung	Schnitt 3 W		Fläche 1		
M-2011-13-1	1066	Schacht 5	746	Verfüllung	Schnitt 3 W		Fläche 1		
M-2011-13-1	2726	Schacht 11	1470	Verfüllung	Schnitt 11		Fläche 5	Planum 3-4	Abbau Pl. 3-4

ExcaBook

FeatureNumber	Excavation Number	BriefDescription	Description	Comments	Date	NN_Up	NN_Down	Length	Width	Depth	Diameter	Interpretation	PreliminaryDa
4	M-2011-13-1	Befund	Befund		2011-06-07							Befund	
1	M-2011-13-1	Erschließungsstraße	Fläche 1		2011-08-03							Erschließungsstraße	
2	M-2011-13-1	Humushalde	Humushalde		2011-08-03							Humushalde	
1	M-2011-13-1	Grube	Pl. 1: rundlich, deutlich erkennbar mit verwaschenen Kon...		2011-10-27						40	Grube	
1	M-2011-13-1	Mauer	07.06.2016 Das aufgehende Mauerwerk besteht aus mi...		2011-05-22			511	45	35		Mauerwerk	NZ
2	M-2011-13-1	Wasserleitung	07.06.2016 evtl. Wasserleitung. Das aufgehende Mauerwe...		2011-05-24			134	53	33		Wasserleitung	NZ
1	M-2011-13-1	Grube	Pr. A-B (W-O): flach wannförmig, veraschene Kontu...		2011-05-24				62	7		Grube	
3	M-2011-13-1	Befund	Befund		2011-06-06							Befund	

Figure 4.14: *Screenshot of the (shortened) retrieved result of* ReMIS.

is displayed to the users in the browser. However, there was no information stored in ArchaeoBook for the selected excavation number and therefore no data was returned from its Connector Application– so it is not included in the result view.

In the result view, the users can now view and sort the results directly in the browser, but can also export the data to XLS, XLSX, or CSV files for further analyses in spreadsheet application or other tools.

4.7 ReMIS Cloud

Whilst the ReMIS architecture allows the retrieval of distributed information from different data sources in one scientific domain, it is not or at least not easily possible to combine data from different domains. An interdisciplinary information system requires an architecture that cannot be achieved by the initial ReMIS. The decentralize architecture of ReMIS provides a solid basis. The heterogeneous data sources remain in their origin data sources. However, we provide a central architecture in which multiple ReMIS are embedded. We call this architecture the ReMIS Cloud which is described below. An abstract overview of the ReMIS Cloud is sketched in Figure 4.15.

4.7.1 Structure

The basic functionality of the Server Applications in the REMIS is retained, but it should now serve as a central possibility to categorize a specific field of data. Instead of distributed Server Applications for separate topics, they are now managed on a central shared server for the REMIS CLOUD technology. We want to provide a unified category system which is considered by all connected data sources. This is essential because these categories form the basis for the search requests. To point this out, we use the term "Category" as a synonym for a Server Application.

If needed, new Categories can be registered to the REMIS CLOUD and need to be activated by a responsible person. Even if this is an additional step to be managed, the REMIS CLOUD technology benefits from central managed categories to allow a better quality control, like to avoid nonsensical and duplicated Categories. A Category basically consists of a name for the Category and the definition of the Category Information Definition (CID). The CID, which is comparable with the Minimal Search Parameters, guarantees that all connected data sources provide the defined parameters and therefore the information that is necessary for the corresponding Category. These are also the parameters the users can search for. We would like to point out that each data source is not limited to one single Category and specific columns of a data source can be mapped to several Categories. If all parameters of all CIDs are mapped, any number of Categories can be assigned.

Furthermore, the existing REMIS architecture is extended by a tagging system to provide further and more detailed filtering of data. This tagging system is integrated to the Server Application and the Connector Application. The administrator of the Server Application defines one or more appropriate tags which describe the general context of all connected data sources. In the Connector Application, the data owners can define individual and more detailed tags for their data during the initialization process. Additionally, each data source also inherits the defined tags of the dedicated Server Applications.

4.7.2 Data Retrieval

The REMIS CLOUD provides a central website where search requests can be defined by any users. Below, we describe the data retrieval that is split in three steps.

In the first step, the users can select from all available Categories that are registered to the REMIS CLOUD. Once one is selected, all parameters of the CID of the Category are loaded and displayed to an input mask. There, the users can enter their search parameters. This is sufficient to already run the data retrieval. All databases that are connected to the corresponding Category will now be queried to gather the data, dependent on the user input.

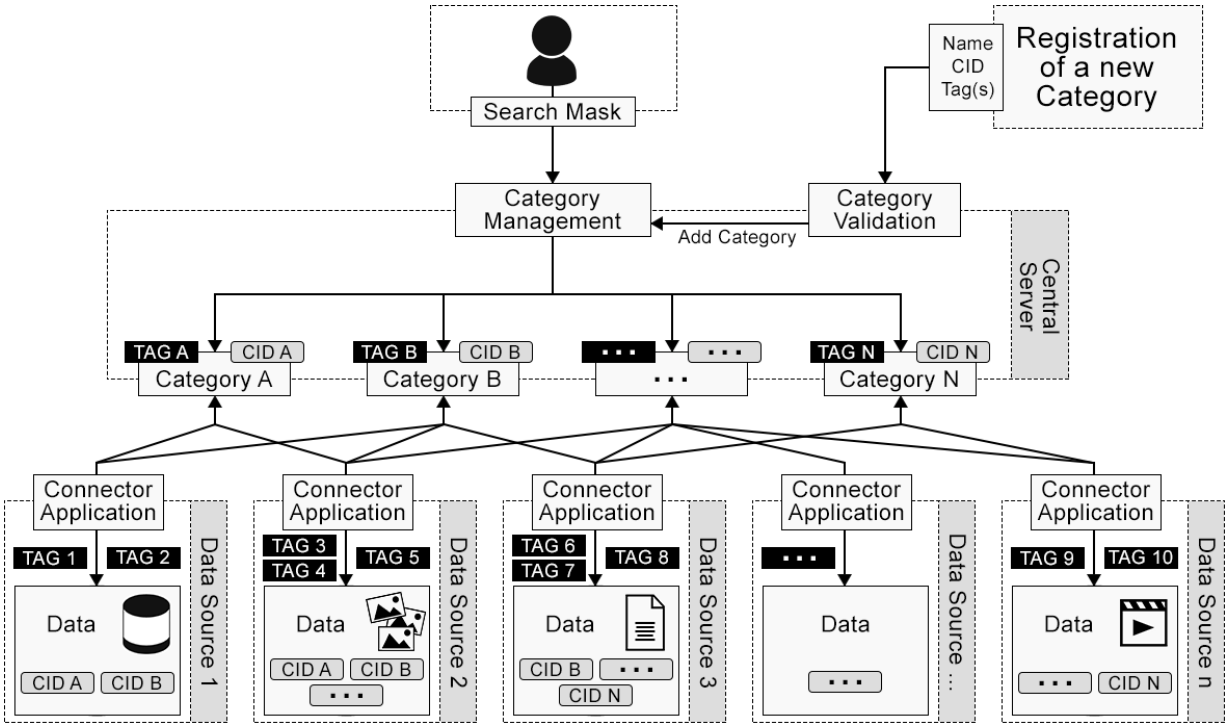


Figure 4.15: *An abstract sketch of the* REMIS CLOUD *architecture.*

So far, the process is comparable with the origin REMIS, but now it also allows multiple Categories to be searched. Thereby, we have to differentiate between an OR search and an AND search which the users decide. Using an OR search, data from the data sources is retrieved once at least one value of the defined search request matches, independent of the searched Category. Using an AND search, the architecture has to ensure that all search parameters match independently on the number of Categories.

All values of the filled search parameters by a user are now merged into a list. Sorted by the order of the selected Categories in the search, the Categories are now queried by the Category Management in the Central Server. Since all data sources that are connected to a specific Category are guaranteed to support the CIDs of the Category, the Category has only to check if the data source was already searched. If this is not the case, the list is passed to the Connector Application of the data source. There, all search parameters that were filled by the user (also the ones of the other selected Categories) are mapped to the local scheme of the data source which is then queried and returns the list of results.

The second step is basically optional, but provides the possibility to search for further information about existing data sets, even if the information is spread through different data sources. Here, the search mask provides a selection of Categories – without the definition of any specific search parameters. This selection displays all available registered

Categories from the REMIS system. A preselection of the available Categories of the data sources is not possible in advance without an additional search to determine them. Due to the accessible data sources may be different for every new search requests, it is not intended to execute several iterations for each search run. If the users select any number of Categories, these will be considered to search for further information dependent on the CID values.

The list of result, which was returned by each data source in the first step, is checked for values of the Categories that were selected by the users for further information. These values are now used to search for further information with the values of the result as parameters. Thereby, a search request can be sent to the corresponding Category. The values of the parameters of the CIDs are not defined manually by the users in this case, but are read out from the data source. The results are sent back to the users. Finally, the results from the first step and the additional information retrieved from the second step are displayed.

In the third step, the users can filter the result for specific tags or data sources. While retrieving the data, the returned search results are supplemented with the tag information and the name of the data source. This enables a filtering for this information. Therefore, the users get a selection of all available tags from the search result and names of all data sources. A selection of the tags and data source names filters the data.

4.7.3 Use Case: Scientific Example (Archaeology)

Archaeo-related sciences deal with distributed data which is not commonly standardized. There are countrywide and statewide, but also regional and individual approaches how to gather and store archaeological data. This makes it hard to consider all available data for large-area analyses. The distributed data sets can hardly be set in relationship due to different data schemes and standards. In contrast to this, analyses of archaeo-related disciplines would benefit strongly from considering scientific data from other disciplines as well. But up to now, the retrieval of interdisciplinary data is cumbersome, time-consuming, and error-prone.

The REMIS CLOUD supports the archaeologists and bioarchaeologists in their work by providing an architecture that allows retrieving spread and interdisciplinary data.

As a vivid example, which is illustrated in Figure 4.16, we basically consider three databases with archaeological data. In Bavaria, Germany, the zooarchaeological database OSSOBOOK and the archaeological database EXCABOOK follow the guidelines of the *Bavarian State Office for Monument and Sites*[32]. Therefore, these databases are connected with the Category “Archaeology BY” (with the tag “Archaeo”) which requires that the parameters of its CID are mapped to the databases, that means: The find sheet number

[32] http://www.blfd.bayern.de

and the excavation number. The database *ADABweb*[33] is used in Baden-Württemberg and Lower Saxony, Germany, and could be connected with the Category "Archaeology BW/NI" (also with the tag "Archaeo") of which the CID requires other parameters to be mapped. Because all three databases have also saved x- and y-coordinates of the excavation places, they could also be connected to the Category "Geographic Coordinates".

Basically, archaeological and bioarchaeological scientists are interested in considering all available findings of an excavation from all databases for their analyses. Using the origin ReMIS architecture, it was already possible to gather all data of a specific excavation number from the databases OssoBook and ExcaBook, but the corresponding findings of the database *ADABweb* are not considered. With the ReMIS Cloud, the users could now search for a specific excavation number in the search mask and define to search for further information about the search results. By selecting the Category "Geographic Coordinates" the system could retrieve all information with the same coordinate information from all data sources that are connected to that Category. In our example, we would retrieve matching information from the database *ADABweb* and also interdisciplinary data, like climate and time information from a weather database. The scientists could now filter the result for the tag "Archaeo" and they would only get displayed the data from the archaeological databases with this tag. As a reminder, the data sources inherits the tags of the connected Categories.

The scientists can also consider the interdisciplinary data from the result. If they filter the result for the tag "Climate" in our example, they retrieve all available climate data with the matching geographical information for the searched excavation number. This could be very useful for large-area analyses of findings where climate conditions should be considered. Of course, this is not limited to climate data. The possibilities depend on the data sources that are connected to the ReMIS Cloud.

4.7.4 Use Case: eLearning Example

Imagine some students have to answer the following question: "The year 2017 marks the 500th year after Martin Luther nailed his theses to the door of the Castle Church in Wittenberg. We assume that he had access to a mobile audio device. What music would he had listened to?" This question is not so trivial to answer by just using a query in the search engine of one's choice. The information is available, but it is not preassembled somewhere. Using classic search strategies, one would try to use consecutive search queries to refine the results and search in different sources. With our information management system, this can be done in one step. We assume that we have the necessary data sources registered in our system. As sketched in Figure 4.17, there is (A) a database containing

[33] `http://www.adabweb.info`

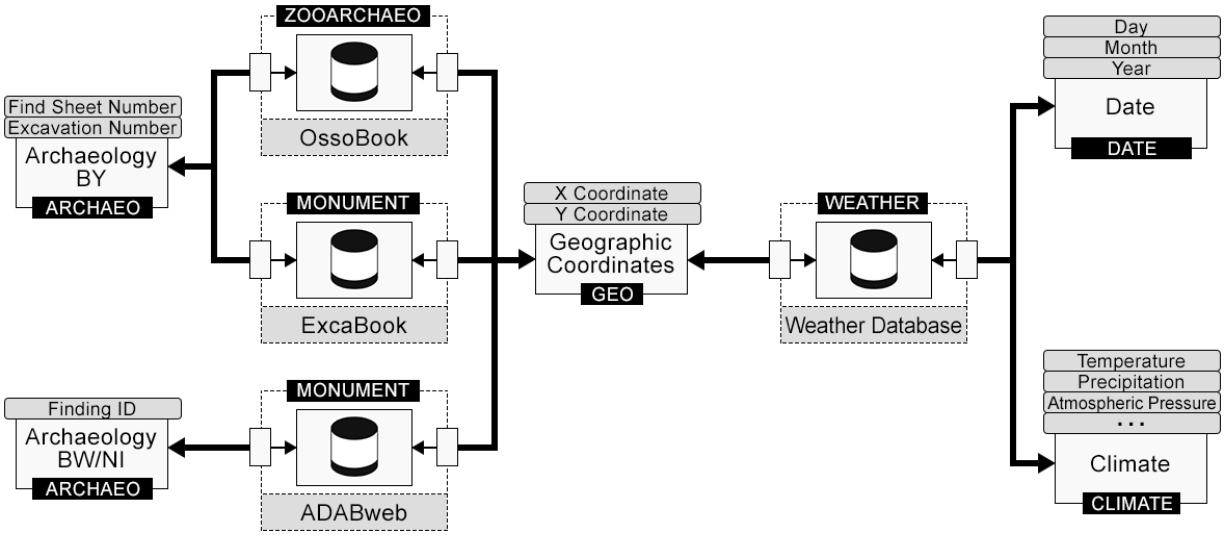


Figure 4.16: *A simplified sketch of the* REMIS CLOUD *architecture for archeo-related sciences.*

historic events, corresponding persons and dates, and (B) a file system containing music files.

First, we query "Luther" in the Category "Historic Events". (A) contains the Category "Historic Events" so this data source is examined for matches with "Luther". The result set could be stated as shown in Table 4.1.

Event	Person	Year
Birth in Eisleben	Martin Luther	1483
Start studying in Erfurt	Martin Luther	1501
Magister artium in Erfurt	Martin Luther	1505
Travel to Rome	Martin Luther	1510
Posting of theses	Martin Luther	1517
Excommunication	Martin Luther	1521
Marriage with Katharina Bora	Martin Luther	1525
Translation of Bible	Martin Luther	1534
Death in Eisleben	Martin Luther	1555
...	...	...

Table 4.1: *Extract of the result for the example query "Luther" in the Category "Historic Events"*

Our tool allows to further inspect the set of data sources for links with the current result data. So we can start the second step with one click. (A) also inherits the category "Dates" and for (B) it is also very usual to do this as well. Music files contain metadata like the publishing date in most cases. In a uniform database environment, we would have two database tables and can join tables using the "year" attribute. Using heteroge-

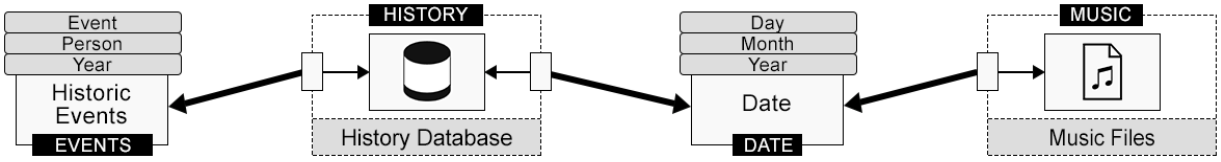


Figure 4.17: *A simplified sketch of the* REMIS CLOUD *architecture for the eLearning example.*

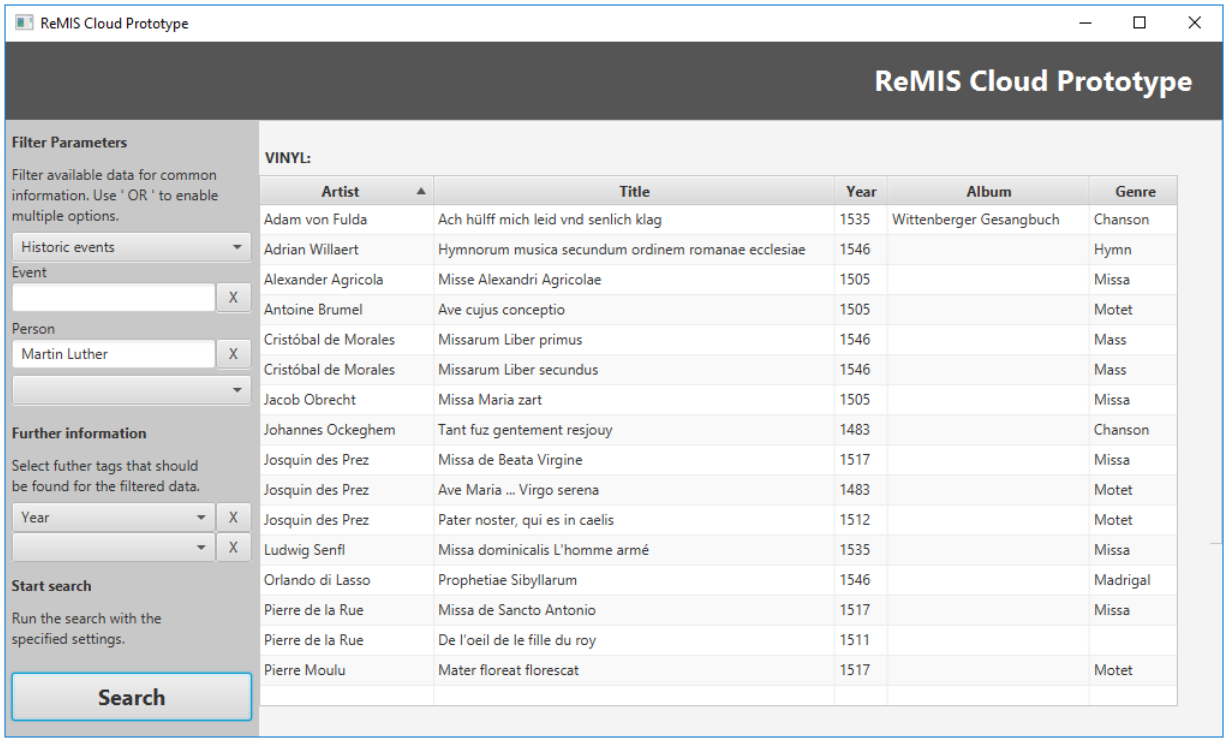


Figure 4.18: *Screenshot of the* REMIS CLOUD *Prototype displaying the result described in Chapter 4.7.4*

neous data sources makes it harder to define join operations.

Our approach utilizes the Category here. Since the data owner had to define the Categories during the registration of the source, the common semantics of “Dates” are already embedded in the system. The system matches other sources’ data with the Category “Dates” and especially with every date according to events in Martin Luther’s life. This allows us to quickly find connected information between Martin Luther and music composed in these times. The result can be seen in the screenshot of our prototype search engine in Figure 4.18. Obviously, we need the data and a correct registration to obtain such results. On the other hand, the individual domain knowledge can be acquired as music repositories and event databases exist.

Availability

A prototype of the REMIS CLOUD architecture can be downloaded from
`http://remis.dbs.ifi.lmu.de`

4.8 Conclusion and Discussion

In this chapter, we introduced the concept of a new architecture, the Reverse-Mediated Information System, motivated by the need to search distributed data together with further contexts from multiple heterogeneous and anonymous data sources. We described the concept of the architecture that allows users to retrieve data without having to know about the existence of specific data sources where the data is actually saved. Then, we extended the architecture to allow multiple disciplines to connect their knowledge which enables querying enriched information. This could not easily be aggregated otherwise.

While this architectures already has many benefits for the users to gather data from distributed databases, there are also some improvements and extensions that can be addressed in the future that would improve both, REMIS and REMIS CLOUD.

In the first approach of the concept, the search possibilities for data are limited to the information of the Minimal Search Parameter. In future developments, different and more complex search options are possible. The Server Application could request all connected databases for their individual database schemes. The columns of this schemes could be displayed in the user interface as own input fields. Therefore, the user has more options and the possibility of a more detailed and individual search through the connected databases.

Currently, it is only possible to search for exact matches of parameters. For certain types like dates, numbers, or coordinates more flexibility would be gainful. For example, range queries would improve the usability for the REMIS by considering matches with locations in a spatial proximity or timestamps in a certain time interval.

Data can be saved in different formats, e.g. geodata can be stored by defining coordinates or the name of the place, or different time or date formats. Currently, we imply that data is entered in the same format, but this might not be the case in other working environments. Therefore, it is necessary to provide a possibility to translate these different formats so that a uniform comparison is possible.

At present, the system is designed to be used for the clear excerpt of structure for archaeo-related data where most of the data sources are trustworthy and known. In future, the system should also be adaptable by extern scientists and institutes – maybe also a full-public approach. Therefore, a verification should be considered to avoid wrong configured Connector Applications and spam.

Currently, the architecture requires a Server Application which is necessary for the

communication between users and data sources. It could be considered to change the system to a Peer-to-Peer environment in which the Connector Applications are communicating directly to each other.

The data sources might also include sensible data, like in diary information or in meta information of pictures. However, this data could also contain interesting information for scientists, but especially personalized data should not be shared in plain text. A way to provide this data is using the concept of k-anonymity [Swe02] that could be implemented for this data in the Connector Application.

Finally, a more complex rights management system could help sharing data selectively to specific users who could create a user account for the Server Application. Administrators of the data sources could authorize access to more detailed data that is set to more restricted privacy settings. This would also be a possibility for administrators to pass data to a user that was contacting him via the contact information (cf. Chapter 4.4.4).

However, different data owners have access to data of different quality levels. Even wrong data can be inserted into the system. An extension to report untrustworthy data sources or mark data sources as spam, so that they are not or less considered during the data retrieval, might be useful.

One server might not be sufficient if the system would grow large enough, that many search requests have to be handled. It might be necessary to apply the same or similar optimizations that are already common for traditional systems, like load balancing, caching, etc.

Additionally, an issue that may be of concern in REMIS CLOUD is that if a data source, which is frequently queried and used as an intermediate link between different Categories, is temporary offline, it might disrupt the information retrieval. The information management system should be improved to cache often queried parameters and their results. This could probably also improve the retrieval time.

Chapter 5

Analyzing Data: Tools for the Scientific Field of Application

Attribution

This chapter uses material from the following publications:

- Johannes-Y. Lohrer, Daniel Kaltenthaler, and Peer Kröger. Leveraging Data Analysis for Domain Experts: An Embeddable Framework for Basic Data Science Tasks. In *7th International Conference on Internet Technologies & Society 2016, Melbourne, VIC, Australia, 2016*, pages 51–58, 2016 [LKK16a]
- Daniel Kaltenthaler, Johannes-Y. Lohrer, Ptolemaios Paxinos, Daniel Hämmerle, Henriette Obermaier, and Peer Kröger. TaRDIS, a Visual Analytics System for Spatial and Temporal Data in Archaeo-related Disciplines. In *13th IEEE International Conference on e-Science, eScience 2017, Auckland, New Zealand, October 24-27, 2017*, pages 345–353, 2017 [KLP+17]

See Chapter 1.2 for a detailed overview of incorporated publications.

Analyzing data is usually the final and most important step in the workflow of scientific work (c.f. Figure 5.1). The collecting, sharing, and retrieval methods that were presented in the previous Chapters 2, 3, and 4 build the basis for analyses of data from disciplinary and inter-disciplinary domains.

Now, with all the necessary data available for the scientists, the next step is to support them with their analyses. Many scientists are experts in their area of expertise. However, they often lack the technical know-how to create complex analyses. They would greatly

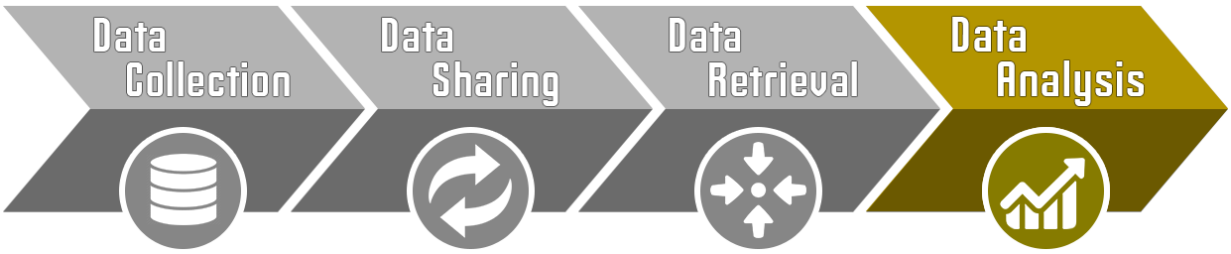


Figure 5.1: *Analyzing data is usually the final step in any scientific workflow.*

benefit if they can, without insight to the structure of the database and the data, create analyses and visualization of the data completely without programming or data querying knowledge.

Therefore, it is important to offer scientists a set of modules which are integrated to the program and are matched to the corresponding data scheme they are using to collect the data. This allows them to easily carry out the analyses of their data.

In Chapter 5.1, we introduce an embeddable ANALYSIS TOOL which allows the creation of different analyses in a familiar working environment. The ANALYSIS TOOL also aims to encourage scientists to try new analyses or slightly vary the default analyses to enable the creation of new knowledge.

In Chapter 5.2, we describe a specialized analysis for the visualization of temporal and spatial information of findings of an archaeological excavation. This tool allows to order and visualize the layers both in their physical location as also the temporal sequence in which the layers were created.

5.1 An Embeddable Analysis Tool

In a database application for scientific data, the accessibility to analyses of existing data is as important as the easiness of correct input. Entering and analyzing data is the most important part of scientific work, but often scientists require a high degree of time and patience to learn, evaluate, and validate an external tool. This effort drains resource from their research work. Scientists working in areas that are not related to IT technologies often do not have the motivation and the resources to get familiar with external applications. A built-in tool would provide the needed data analysis features relevant for this scientific areas.

Therefore, exporting data into a spreadsheet (like Microsoft Excel, LibreOffice Calc, etc.) or CSV file and importing this into a separate analysis application can in the worst hinder them completely from analyzing the data since the process maybe too complex, time-consuming, or error-prone. Also, generating the analysis in a spreadsheet directly

is not always possible since complex analyses might require additional functions that are not provided by a generic spreadsheet application and cannot easily be added without programming skills. Clearly, the process of the data analysis should be easier for the researcher. The best case would be if the feature to analyze the data is already built into the application where all data is stored and new entries can be added because the scientist using the built-in application is already very accustomed to this working environment.

But even if the analysis tools are built into the application, a crucial problem still remains: The scientists who are carrying out the analyses are often not responsible for creating the analysis applications nor are they even involved in the process of creating the tools. Not all variations of analyses can be known beforehand since there are often tasks specifically dependent on the scientific work. So the domain experts have to be able to create exactly the analyses they require.

To allow this, the application has to provide not only predefined analysis methods, but also offer dynamic generation of analyses by chaining together different simple configurable modules. However, some scientific tasks are so special that these modules are not sufficient. Therefore, there has to be a way to add own, specific modules as well.

In this chapter, we provide a ANALYSIS TOOL for data analysis that can be embedded and thus can be used in different applications. The framework has to be extendable, easy to be integrated into an existing application, and provide multiple operations that are necessary to analyze data. The ANALYSIS TOOL has to be flexible and easy to be used by scientists who are not willing to get familiar with external technologies to support them in their work.

In summary, the main contribution is as follows: We list a set of requirements that should be addressed by an embeddable ANALYSIS TOOL for scientific data analysis (cf. Chapter 5.1.1). We discuss methods of existing analysis applications in Chapter 5.1.2. Then, we describe the data structure, the functionality of the modules, and the classification that describes the values in the database in Chapter 5.1.3. We describe the integration of the framework to another base application (cf. Chapter 5.1.4) and the definition of new, custom modules (cf. Chapter 5.1.5). Finally, we introduce some of the most common modules as examples (cf. Chapter 5.1.6) and use them to present an example of a workflow (cf. Chapter 5.1.7).

5.1.1 Requirements

Allowing the users to generate their own analyses out of arbitrary data offers a challenge because neither the input nor the output is known. All steps in between are also up to the users. Still, the users have to be able to work in a responsive environment which allows them to carry out the steps they want and need. Additionally, not all operations should be allowed or are possible at a given step of the analysis pipeline.

As a consequence, we have defined several requirements that have to be met to allow a dynamic generation of analyses.

- **Dynamic data structure:**
  We need a data structure that allows data to be easily added, removed, merged, and accessed, since the data has to be generated, transformed, searched in, and of course also be displayed. This data structure has also to be usable in every part of the analyses independent of the previous steps.

- **Modular components:**
  Since the users have to define the operational steps, the order in which specific tasks are run is not known beforehand. Therefore, we need modular components that can be linked together and define a "workflow". The components have to be able to accept the data structure – no matter what components were used before, but not all inputs have to be valid for the component. Therefore, it is possible that none of the inputs can be used by the component, but still the data structure itself has to be accepted. To make the component more dynamic, it also should provide settings like which input to use or the order to sort. These settings should, if applicable, dynamically change depending on the input.

- **Classification:**
  All fields, for which the data can be retrieved for, have to be defined in a way that each component can decide if or how the data for this field can be processed. For example, a date value may have to be treated differently than a pure text value. With this classification, a component can also decide if it is able to work with the field or not.

- **Extendability:**
  It has to be possible for everyone to extend the application with new components that offer new, possibly very specific, functions. Since there are many special analyses that cannot be put together with generic modules, it has to be possible to include these in the list of available workers.

- **Open Source:**
  The area of application might be in an open source environment, therefore the analysis has to be open source that it is possible to include it into the application. Some other licenses might be applicable as well, but still this is a requirement that is very important due to legal issues.

- **Embeddability:**
  Since the analysis shall be done with the data the users might just momentarily

have entered, we require the analysis to run directly from the program without first having to extract the data into spreadsheet, CSV files, or similar. This means that the analysis has to be able to connect itself to another program and use the structure defined inside the program for its analysis.

5.1.2 Existing methods

Now, we want to discuss if some of the most commonly used tools meet our requirements. Since there are many different analysis applications, we cannot cover and evaluate all of them. Therefore, we do not claim to offer an exhaustive comparison, but still think we covered some of the most used tools that best fulfill our requirements. These can also be found at kdnuggets.com [Jon16].

- **Tableau Public:**
  As a commercial service, *Tableau Public* [Sof03] allows creating interactive data for publications in the web that can also be used for analyses of any data. The tool supports the analysis of text, numbers, dates, and coordinate values and offers extensive visualization methods.

  The free edition of the tool is provided on a limited scale. As a data source only a few file formats are supported like Microsoft Excel, CSV files, or some files types for statistical data. More data sources, like the connection to database systems (MySQL, Oracle, Microsoft SQL, etc.), only come along with the professional edition which is fee-based. *Tableau Public* cannot be included directly into another application. Files are always saved on the own profile and cannot be downloaded or saved on the own computer in the free edition. Furthermore, it is not extendable. The existing analysis methods cannot be extended with own, specific ones.

- **KNIME:**
  A modular approach with graphical nodes that allows many different input methods including tables, comma-separated values files, and even images is used by *KNIME*. It enables the combination of "simple text files, databases, documents, images, networks, and [...] Hadoop based data" [KNI06] and integrates the use of modules for data blending, transformations, math and statistical functions, and predictive algorithms. The extension of new modules is supported with an API. But still, the application is an external one that cannot be included into another application.

- **RapidMiner:**
  Just like *KNIME*, *RapidMiner* [Rap06] also uses graphical nodes for the representation of the data. It is an open source project in the basic version and can be ex-

tended with own nodes. It supports connections to databases. But, like the others, this tool has the disadvantage that it cannot be used inside the main application and has to be run as a separate instance.

In general, all of the mentioned tools are very good in what they do. They allow a wide range of different analysis methods and also can be partially extended with new functions. Additionally some of them allow a connection directly to the database without having to extract the data first into an own file. Also, they all offer some kind of classification of the different fields. Still, none of the tools allows an integration into an existing application.

5.1.3 Realization

In Chapter 5.1.1 we discussed the requirements for a dynamic analysis. Now, we take up these requirements and present our approach to meet them. Of course, especially for the data structure, there can be more than one valid solution.

5.1.3.1 Data structure

To allow a dynamic and flexible restructuring and access to the data, a special data structure is required. Therefore, the data structure for our framework consists of several elements.

- **Column Header:**
  The Column Header holds the important information about the specific field, such as the type and the display name. Since a field is basically a column in the database, we use the name of the field and the column as synonyms. The Column Header itself does not store any information about the value of the field, but serves more as information and meta data for the column. The additional information that is stored in the Column Header is described later in more detail.

- **Entry Value:**
  The smallest data type to hold the values is the Entry Value. It holds a list of strings which represents the values for a specific entry. This is necessary if a field has several options, e.g. different values for specific measurements. Therefore, every distinct value is one string and all of them are saved inside the Entry Value.

- **Entry Map:**
  The Entry Map maps the Column Header to the Entry Value inside a map. This map now represents a complete entry. It can easily be accessed because of the nature of the map. Therefore, writing and reading is no problem. Also editing values can easily be done.

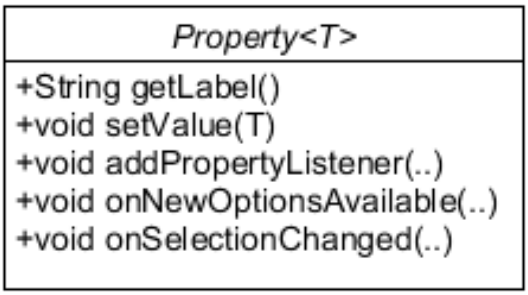


Figure 5.2: *Schematic representation of a Property.*

- **Data List:**
  The Data List is a list of all Entry Maps which represents a complete data set. It also contains the list of all Column Headers that are in any of the Entry Maps. This serves as a utility method to allow components easy access to the list of all fields without having to iterate over all entries. The list is generated by adding all unknown Column Headers whenever a new Entry Map is added to the Data List.

5.1.3.2 Worker

Each component, or "Worker" as we call it, can have multiple inputs and outputs of data. Some Workers do not necessarily have inputs, like Workers that retrieve data from the database, since they generate data without manipulating it. At the same time, there can be Workers without outputs, like a Worker that allows the users to display the data as a diagram.

- **Properties:**
  The Worker consists of inputs and outputs, settings, and the actual logic of the Worker. The latter either uses the input(s) or creates a new Data List, runs its logic considering the settings, and finally outputs the data or displays it.

  The Worker can define a list of Properties which can represent a setting. So for every setting type there has to be an own property. Typical properties are text properties where the users can enter text (for example to describe a name) or combo properties where the users can select one value from a list of values. The schematic representation of the property can be seen in Figure 5.2.

  Below, we describe the methods a Property has to implement:

  - `getLabel()`: Returns the label to describe the property. This should make clear for the users which setting they can manipulate.

  - `setValue()`: Called by the graphical representation with the specific value. This is used to tell the Worker that the value has changed and therefore has to update its data structure and possibly additional properties.
  - `addPropertyListener()`: All elements that are dependent to this property can register themselves as listeners which have to be notified if this property has changed. It may be called when either new options are available that can be selected or the value of the property itself was set.
  - `onNewOptionAvailable()`: This is called if new options for this property are available. This causes all property listeners to be notified, so that the graphical user interface can now display the updated values.
  - `onSelectionChanged()`: Almost identical to the method `onNewOptionsAvailable()`, but instead it is called when the value of the property is changed, for example after the `setValue()`method was called.

  The Worker also defines the number of inputs and outputs. The users connect different Workers which basically is a directed graph or multigraph with the nodes representing the single Workers and the edges representing the connections between the Workers. The number of inputs can vary as some Workers require no input some require an exact amount, and some can work with an arbitrary number of inputs. If the Worker provides an output this can be used to pass the data on to other Workers.

- **Data handling:**
  Each Worker fulfills a predefined task like retrieving or merging data. But still, the output of the Worker is only defined after the input(s) and settings for the Worker are set. Therefore, it is not necessary to instantly generate the output since the input(s) may change if a setting in a previous Worker was changed. But at the same time it is important to know at least the Column Headers of the data to enable updating the settings of the successive connected Workers. This is the reason why the output is separated into two parts:

  - **Output Scheme:**
    The list of all Column Headers is returned by this Worker. This list is instantaneously generated as soon as an input or setting is changed. The successive connected Workers are immediately notified with an event that the Output Scheme of this Worker was changed and enables checking themselves if their Output Scheme changed as well. This list has the same value that the list of Column Headers inside the Data List should have in the real output. The-

refore, Workers only need this list to define which settings they provide and what Output Scheme they return.

– **Output Data:**
The Data List with the "real" output containing the data. This is only generated if necessary since the composition of the data could take some time. A complete recalculation is not required if only the Output Scheme is important. Of course, the real data has to be generated if a diagram representing the data has to be displayed or the values should be listed. This is done recursively. Each Worker, beginning at the end, requests the Output Data of the Workers that are connected with its input. Of course, this means the starting Worker has to be able to generate data without any input.

This separation into Output Scheme and Output Data allows a fast applying of updated settings and rearranged inputs or outputs while still ensuring that the correct type is used.

- **Interaction with the Graphical User Interface:**
The Worker itself is only responsible for the logic. But since the users need to be able to easily arrange, connect, and configure the Workers, there has to be a graphical representation. The graphical user interface is notified with events of changes in the properties. At the same time it uses the properties to notify back to the Worker if any value of the settings was changed, such as entering a text or selecting a new value. This allows the graphical user interface to be created independently of the logic and therefore is not limited to a specific format. The exact design of the graphical user interface is not part of this thesis. Here we want to describe the technical connection to the graphical user interface. A discussion about the benefits of different GUI designs was done in [KLK17].

5.1.3.3 Level of Measurement

Since typically the database containing the data consists of multiple tables with a variety of different types of entries, not all columns can be used to carry out every operation. For example, a text cannot be used for a numeric ordering of entries or a numeric value should not be alphabetically ordered. For every column it has to be defined which types of operation are compatible with the column.

This is achieved by using the *Level of Measurement*, a classification that describes the nature of information within the values [Ste46]. It defines four different basic scales of measurement:

- **Nominal Scale:**
  The different values can just be differentiated by their names. Therefore, it can only be used to get the quantity and to check if a value equals another or not.
  Possible modules: Filter, counter.

- **Ordinal Scale:**
  The values can be ordered, but do not allow to measure the degree of difference between different values.
  Possible modules: Sorter, median, mode, and all modules of Nominal Scale.

- **Interval Scale:**
  The values can be measured by degree of difference, but not the ratio between them.
  Possible modules: Arithmetic mean, standard deviation, and all modules of Ordinal Scale.

- **Ratio Scale:**
  In addition to a degree of difference, values have a meaningful zero value.
  Possible modules: Elementary Arithmetic and all modules of Interval Scale.

Since all columns now have a definition of which scale they have, modules can define which scale they can work with. Therefore, they only work with columns that have the required scale. For our intention, every column has to be assigned to one of the scales. The corresponding Level of Measurement of the columns is saved inside the Column Header. Additionally, we require information how to order the entries for every column. The default ordering is alphabetically. Still, numeric values should be ordered accordingly and also text values that have an ordinal scale should be ordered correctly. This requires a Comparator to be set inside the Column Header for the corresponding column.

5.1.4 Integration

To be able to reuse the ANALYSIS TOOL in different base applications, it is important that the integration requires as few changes to the base application as possible. But since the ANALYSIS TOOL cannot know how the data is stored or how the connection to the data is realized, the base application has to implement some wrapper methods.

The ANALYSIS TOOL itself is composed in a `JPanel` or `JFXScene`. The base application can either create a new `AnalysisSwing` or `AnalysisFX` instance which both require an `IController` (cf. Figure 5.3). The `IController` is the interface that serves as the combination of the base application to the analysis. It provides the following methods that the base application has to implement:

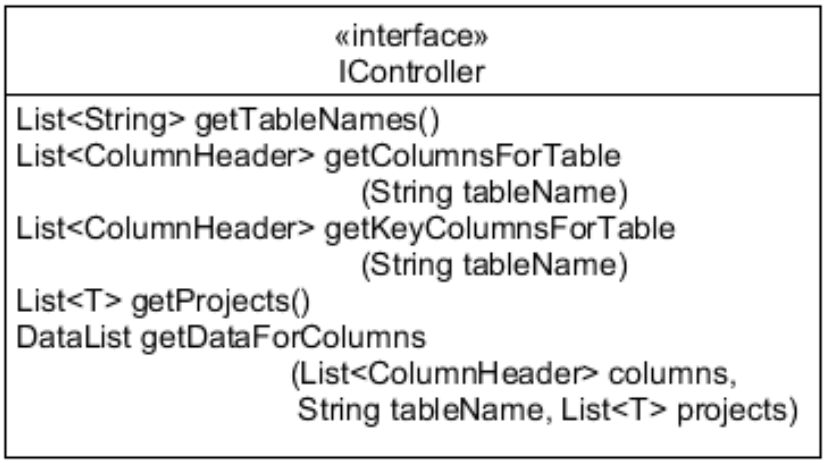


Figure 5.3: *Schematic representation of the IController.*

- `getTableNames()`: Returns the list of different names of tables that can be selected for the analysis. This should only return tables in which the users have entered data.

- `getColumnsForTable()`: Returns the columns for the given table that can be included in the analysis. This should return only columns that are important for the users, but no columns that contain additional information that provide no information for the users. The expected return value is a list of Column Headers. Therefore, all columns have to be transformed into this format. This is important because all future analysis options are based upon the information stored inside the Column Headers.

- `getKeysForTable()`: Returns the key columns for the given table. This can be used for example in the Combiner to provide default mappings. Also this method requires the returned values to be a list of Column Headers.

- `getProjects()`: The database can possibly be structured in different "projects" which is a logical separation of different data sets. To also allow this separation inside the analysis framework, a list of unique identifiers can be returned. This could be just a name or an integer, but can also be a more complex data structure in which the `toString()` method returns the name of the project, so that his can be later displayed for the users. If the separation into different projects is not desired or supported, this method can return `null`.

- `getDataForColumns()`: Returns the Data List for the given columns in the given table for the optional project. Since the ANALYSIS TOOL has no knowledge about the structure of the database, the base application has to generate the Data List. The list

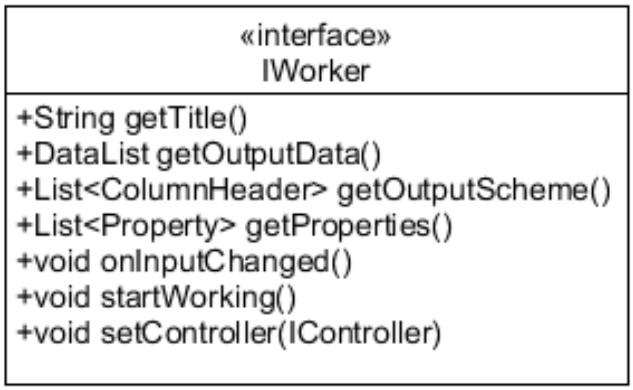


Figure 5.4: *Schematic representation of the Worker.*

of projects is only required if `getProjects()` returns a value and can therefore be ignored if it is not applicable. The values that are returned should not be the values as they are stored in the database, but already translated into human readable form, e.g. by translating IDs into the appropriate values.

If necessary, all Workers can use the `IController` to retrieve the data they require. This is the only connection data-wise needed for the analysis, as all further analyses is built onto the retrieved data. Therefore, the base application has not to know any internals of the analysis or other way around. Since the analysis is done inside one panel, it can be easily be included into the base application which can decide where and when the analysis shall be displayed.

5.1.5 Definition of Custom Workers

While creating an ANALYSIS TOOL not all use cases can be known beforehand since the area in which the analysis is used is not always known in advance. Therefore, it is very important to be able to add new Workers accordingly to the requirements of the individual analysis that shall be carried out.

For this, we defined a very simple API to allow new Workers to be easily implemented.

To add a new Worker, it has to implement the interface `IWorker` (cf. Figure 5.4). It defines the basic functions that are required for the integration in the workflow.

- `getTitle()`: This method expects the name of the Worker to be returned as a string value. This name is displayed for the users inside the graphical user interface. It should be a short but meaningful name which allows the users to instantly comprehend the function of the specific Worker.

- `getProperties()`: Returns a list of Properties which define the settings of this Worker. With these settings, specific aspects for the Worker can be set (cf. Chap-

ter 5.1.3.2). This list of Properties also includes Properties for the input and output which define the connection to other Workers. The Worker is notified over the Properties if the input or a setting has been changed. Since the Properties are abstract classes, all methods of these classes have to be implemented when creating a new Worker (cf. Figure 5.2). This includes the displayed name, the logic for setting and retrieving data, and additionally which values the representation in the graphical user interface of the Property shall display, depending on the Property.

- `getOutputData()`: Returns the Output Data of the Worker after it has completed its job. If neither the input nor the settings of the Worker have changed this can return the generated values from the previous call. Otherwise the process has to be run again. Therefore, in this case the method has to call in this case the `startWorking()` method and return the generated values after it has completed.

- `getOutputScheme()`: Returns the Output Scheme of the Worker that would be returned in the Output Data if the Worker would return its generated data set. The Output Scheme should be calculated with regards to the Output Scheme of the connected previous Workers (if there are any) and the settings of this Worker. If the input and settings of the Worker did not change, this method does not have to recreate the Output Scheme again, but can just return the previously generated data. Also, this method should not run the complete calculation method, but only calculate the Output Scheme.

- `onInputChanged()`: This method is called by the Properties to notify the Worker that something has changed and the Output Scheme and Output Data have to be recalculated, if requested. However, this method should not start the update process itself.

- `startWorking()`: In this method the actual logic is carried out. The input is collected by iterating over all input Properties and getting the Output Data of the connected Workers. This triggers them to generate their results themselves if needed by carrying out the same logic recursively. Then the result of the Worker is calculated with regards to the settings defined in the Properties. This method is usually only called inside the `getOutputData()` method.

- `setController()`: This method is used to set the `IController`which is used for interaction with the base application. This is required to be an own method and not part of the constructor since all Workers are created with reflection. Therefore, the constructor needs to be the empty constructor. With the `IController`, the Worker is able to query the base application for information that is possibly required for the Worker to carry out specific operations.

After the Worker has been created, there are two options to register it to the application.

- **Direct registration:**
  The API provides a registry class which allows Workers to be registered for inclusion in the ANALYSIS TOOL.

  In addition to the Workers available by default, the registered Workers can then be selected by the users. This method requires of course access at runtime and therefore can usually only be done from the base application.

- **Indirect registration:**
  The indirect registration allows externally created Workers to be included in the ANALYSIS TOOL. For this, all .jar files inside a specified folder (default "./analyses/-workers") are analyzed if they contain classes that implement `IWorker` and if so, they are added to the list of available Workers for analyses.

Since the Workers use Properties to tell the graphical user interface what to display, it is important to also add new Properties if the available Properties are not sufficient. This consists of two parts: The definition of the property itself and also of the definition of the graphical representation of the Property.

All new Properties have to extend the `Property` class (cf. Figure 5.2). They can define additional methods that are used by the representation of the graphical user interface. Then the newly created Property can be registered together with the graphical representation in the registry.

5.1.6 Example of Provided Basic Workers

We have already mentioned that there are types of Workers that fulfill different tasks. Here we want to give an example of the most common Workers that are sufficient for a basic analysis. A screenshot of these Workers from the ANALYSIS TOOL can be viewed in Figure 5.5. Of course, there are far more different Worker types possible, but we only want to give an overview over the possibilities the Workers provide.

5.1.6.1 Retriever

The Retriever is the most basic Worker that is used in every analysis. The Retriever, as the name suggests, retrieves data from the database. The users can define the fields and projects they want to include in their analyses. The available fields and projects are retrieved from the local database. They are a representation of the data structure of the application the ANALYSIS TOOL is embedded in. This would normally be the fields the

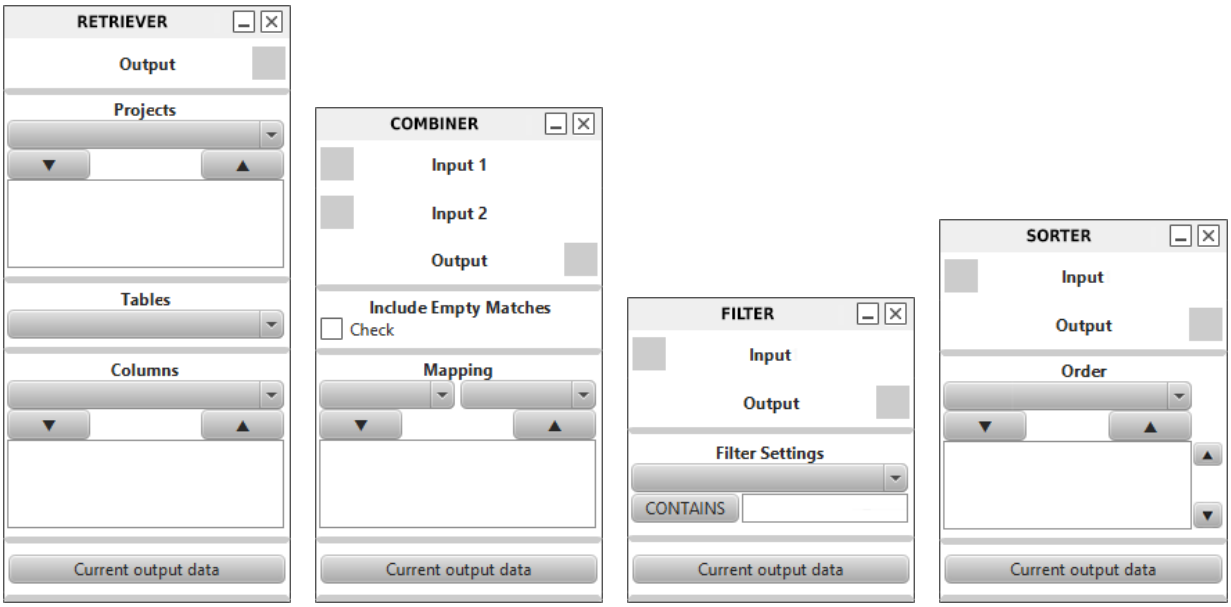


Figure 5.5: *From left to right, the screenshots of the Retriever, Combiner, Filter, and Sorter, as they are represented in the graphical user interface of the tool.*

users either want to filter, display, or further analyze. The Retriever is the only Worker that needs a connection to the database to get any data. It uses the `IController` to get the required data. The retrieval from the database is most likely a simple SQL query for all fields selected by the user.

5.1.6.2 Combiner

The Combiner allows different data sets to be combined. The users can define the fields which should be considered when combining the data sets. The list of available fields is defined by the Output Scheme of the previous Workers as described in Chapter 5.1.3.2.

For example, the Combiner can be used if a user has already created two different analyses with different data sets and now wants to combine the data for a third analysis. The Combiner searches for entries in the given list of data sets that are equal on the fields the user entered. The result of the combination can be compared to the SQL join. There are two possibilities that either only entries having a match or also entries that have no match are combined. In this case, the other columns are filled with empty values. Then these entries are combined into a new entry.

5.1.6.3 Filter

To filter out entries that are not required in the analyses, the Filter can be used. For example, an analysis about specific animals may only require the entries containing data

of these animals. So the users can filter out all other animals by using the Filter.

The users have several options: They can specify the fields for which the data shall be filtered. The list of available fields is defined by the Output Scheme of the previous Worker, as described in Chapter 5.1.3.2. Depending on the column header, they can specify whether the value of the field contains or equals a specific text. For dates or numbers it is possible to check if the value is smaller, greater, or equal than a specific user input. It is also supported to define a combination of different fields that can be filtered at the same time.

5.1.6.4 Sorter

The Sorter sorts entries according to the comparator set in the Column Header (cf. Chapter 5.1.3.3). It only has one input, but allows more than one column to be set as sorting column. This allows applying different priorities if the values of the column with a higher priority are equal. This can for example be useful if in a diagram a specific order of entries is required.

5.1.6.5 Diagram

The Diagram is the umbrella term for Workers that display the result in a graphical representation. It can be used in different and complex ways.

As an example, we describe a two-dimensional, axis-oriented bar chart. The users can select the field which values shall be used for the x-axis. The list of available fields is defined by the Output Scheme of the previous Worker as described in Chapter 5.1.3.2. The different values of this field are then listed as values of the x-axis. The number of entries for the given value are displayed on the y-axis. An example of a Diagram can be viewed in Figure 5.6.

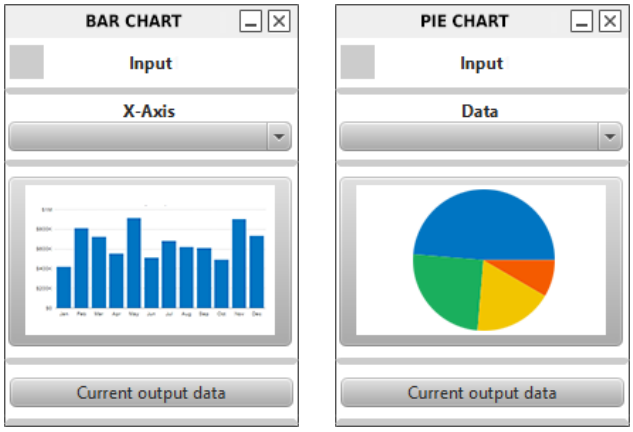


Figure 5.6: *Examples for different Diagrams*

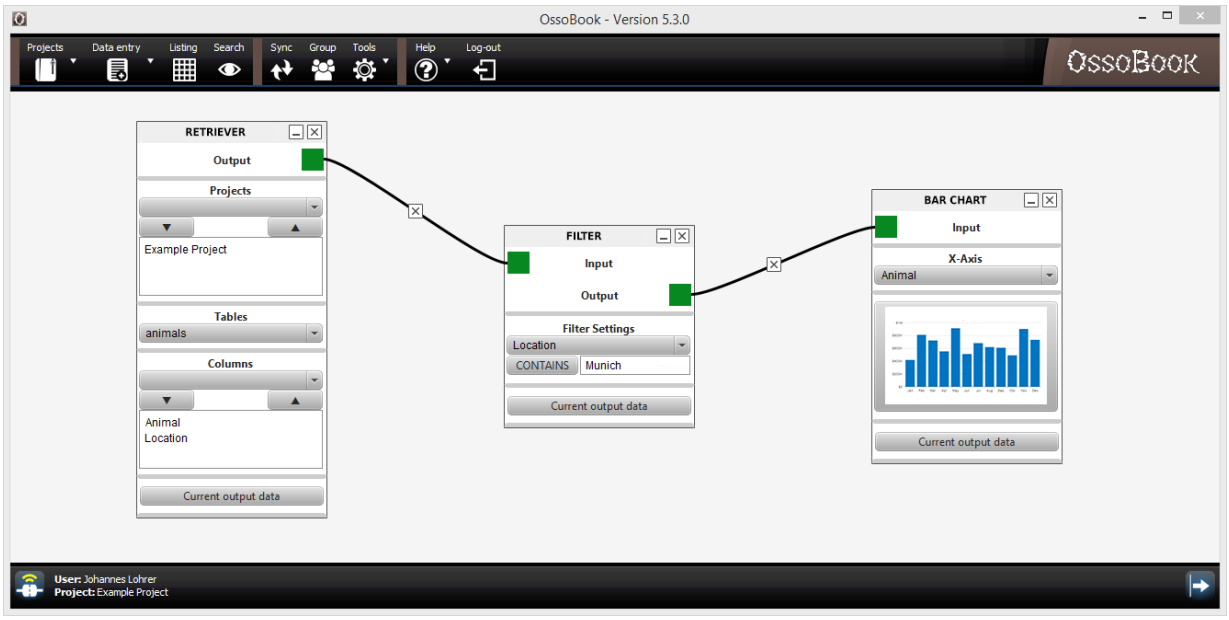


Figure 5.7: *Composition of the analysis for animal distribution in Munich, described in the first example of Chapter 5.1.7. The analysis framework is embedded to the zooarchaeological database* OssoBook *[KLK+18a].*

5.1.7 Workflow

We discussed the structure required for the Analysis Tool and also gave examples of the most basic Workers. In the next step, we want to put the pieces together and describe how a typical analysis is built. We will show two examples: The first one is a very simple, but still frequently used analysis to get the distribution of animals. The second one is a bit more complex and deliberately abstract and requires a custom Worker to be implemented.

For the examples below we take a sample table with the columns "*ID*", "*Animal*", "*Location*", and "*Size*" as shown in Table 5.1. Of course, in real scientific databases a table like that would contain several more columns and data sets. Because of reasons of clearness we reduce the example data to a minimum.

5.1.7.1 Basic analysis

In the first example, we want to calculate a distribution of animals in Munich, and create a graphical representation of the data as a bar chart. The composition of the data in the Analysis Tool is shown in Figure 5.7.

The analyzing process can be divided into three different steps:

- **Planning and collection of data:**
  Gathering all required data from different sources and combining it so it is able to be

ID	Animal	Location	Length	...
1	Dog	Munich	85	...
2	Mouse	Los Angeles	74	...
3	Dog	Guarujá	14	...
4	Cat	Baltimore	21	...
5	Dog	Munich	42	...
6	Cat	Baltimore	32	...
7	Mouse	Los Angeles	98	...
8	Penguin	Baltimore	569	...
9	Dog	Munich	65	...
10	Penguin	Guarujá	851	...
...	...	...	...	...

Table 5.1: *Animals*

used for further processing.
We first identify the fields we need in our analysis. These are: "*Animal*" and "*Location*". The fields "*ID*" and "*Size*" are not important for our goal and can be disregarded.

The first Worker we use is a Retriever to get the necessary data. We configure it to get the fields "*Animal*" and "*Location*" only.

- **Processing data:**
  The collected data is processed to remove unimportant data or structure the data, e.g. by filtering or sorting it.
  Next, we only want to filter all animals from Munich. Therefore, we add a Filter. The input of the Filter is connected to the output of the Retriever, which makes the fields "*Animal*" and "*Location*" available as options for the Filter. There we select "*Location*" and define only to allow data sets in which the "*Location*" value equals the term "Munich".

- **Generating result:**
  The processed data is used to display a result like a diagram or any other visualization.
  The last Worker is a Diagram. It takes the filtered data from the Filter and displays a graphical representation of it. In our case, we choose the bar chart. We can now select the column its values we want to use as the x-axis. We select the "*Animal*" column and the Worker generates the chart as seen in Figure 5.8.

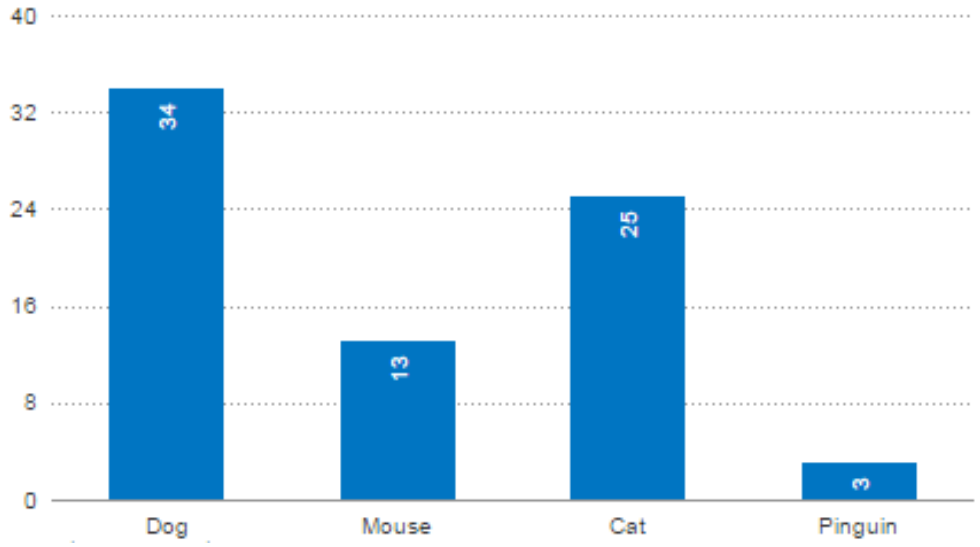


Figure 5.8: *Possible result of the first analysis.*

5.1.7.2 More complex analysis

We now want to calculate the most common animal related to the average temperature in our sample data. For this, we take data from Table 5.1 of the previous example and introduce a new table with the average temperatures of the locations, as shown in Table 5.2. The composition of the analysis can be seen in the screenshot of Figure 5.9.

ID	Location	Avg. Temperature
1	Guarujá	21.8
2	Munich	13.0
3	Baltimore	18.4
4	Los Angeles	21.3

Table 5.2: *Average Temperatures*

- **Planning and collection of data:**
  Like in the example before, we start with a Retriever that returns the fields "*Animal*" and "*Location*" of the table "*Animals*". Additionally, we need a Retriever that returns the fields "*Location*" and "*Avg. Temperature*" from the table "Average Temperatures".

- **Processing data:**
  Now we only want to get the most common animal per location. Therefore, we have to create a custom Worker. This Worker accepts only one input and provides

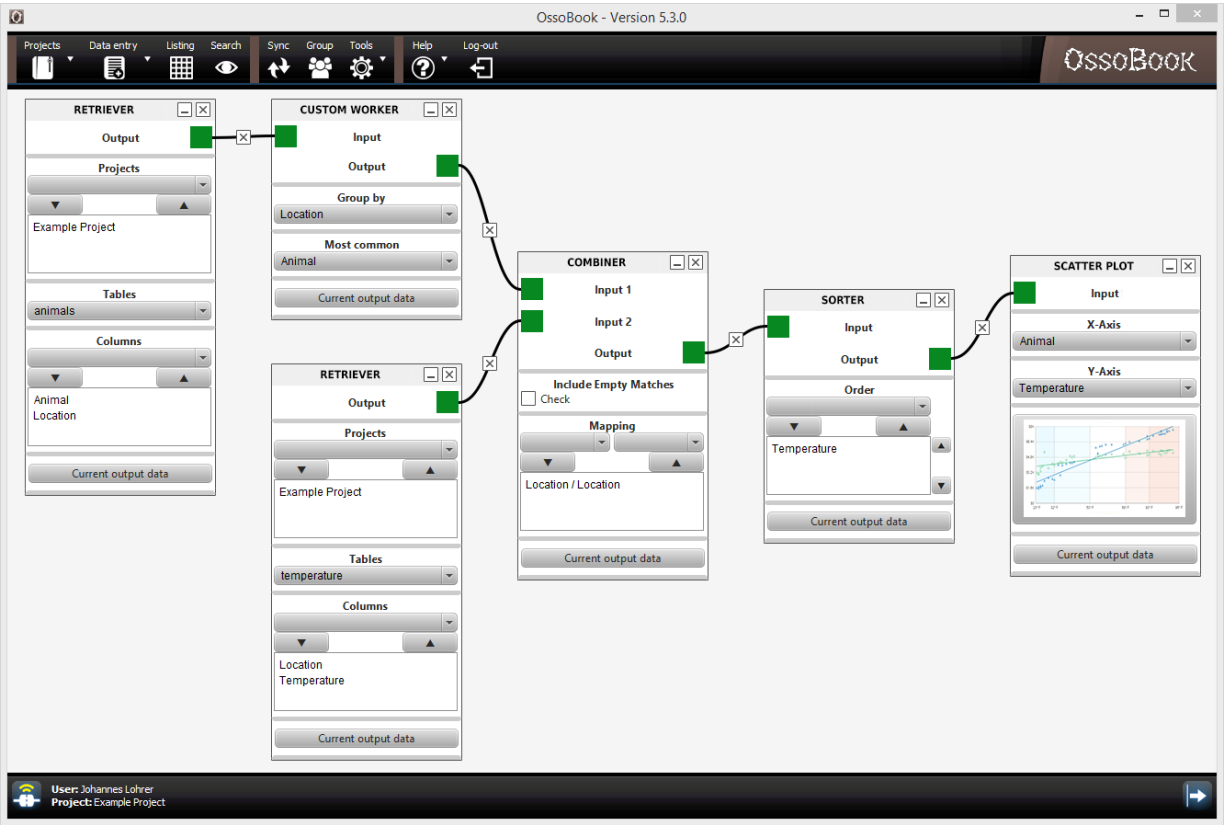


Figure 5.9: *Composition of the analysis for the most common animal, dependent on the average temperature, described in the second example of Chapter 5.1.7.*

one output. The Worker consists of two `ComboProperty` elements. The first one defines the column for which the values are aggregated. The second one defines the column for which the most frequently occurring element is searched for, depending on the value selected in the first Property. In our example, we select "*Location*" as the column to be aggregated and "*Animal*" as the column for which we want to calculate the most common element. Then it returns a `DataList` containing only the two selected columns with the aggregated values to the output.

To merge the two `DataLists` with the grouped values and the temperatures, we use the Combiner. In it, we define the two "*Location*" columns as the key, so that the Combiner joins the two lists into one `DataList` with the columns "*Animal*", "*Location*", and "*Avg. Temperature*".

As a demonstration, we want the result to be ordered by temperature, so we need the Sorter as another Worker. The Sorter uses the comparator of the selected column which would be a numeric comparison in this case.

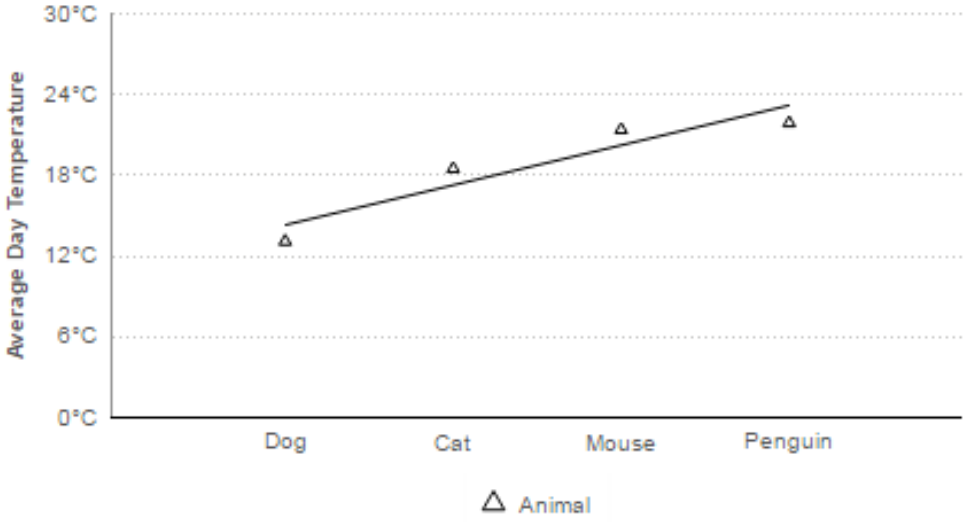


Figure 5.10: *Result of the second analysis.*

- **Generating result:**
  To display the results, we again use a Diagram. This time we us a scatter chart, that can be seen in Figure 5.10. We select the "*Animal*" column as the x-axis, and the "*Temperature*" column as the y-axis. Since the y-axis can only display numeric values, only columns are valid for the y-axis that are at least in interval scale.

5.1.8 Discussion

In this chapter, we described an ANALYSIS TOOL for generating analyses that can be embedded into a base application. The framework aims to be intuitively used by scientists without any IT background, inside their accustomed working environment. The ANALYSIS TOOL can easily be extended and integrated into a base application. We identified the requirements for the framework which have to be fulfilled to allow working with it. Then, we discussed the applicability of the requirements with some existing analysis applications. We described the realization of our tool to be able to meet the requirements and discussed how the framework can be integrated into a base application and extended to the demands of the user. Finally, we presented some of the basic Workers and used them in two examples to show the analysis workflow.

While the ANALYSIS TOOL can already be used for generating interesting analyses, there are still some issues that could be addressed in the future to make working with the framework go more smoothly.

Due to the nature of the implementation, it can only be used in a Java environment which is the main limitation of this analysis framework. While the concepts themselves work in any environment, the implementation would have to be specific for the language of the base application. An application that is not written or compatible with Java cannot currently use the ANALYSIS TOOL. So either there would have to be an instance of this tool for every programming language or a wrapper has to be developed to be able to use the framework in other programming languages as well.

While the integration into the base application is straight forward and can easily be done, it still requires both access to the source code and knowledge how to program. Therefore, the typical users cannot do the integration themselves. This means that the developer of the base application has to integrate the ANALYSIS TOOL to provide the functionality to the users. For these cases, the framework could be extended to run as a standalone application which can be connected to the database directly. This would require additional settings which handle the connection to the database itself. A stand-alone tool could benefit from the same level of flexibility and extendibility that the framework offers while being integrated into a base application.

Additionally, the issue still remains that for creating a new Worker some programming skills are required. As opposed to the integration into the application. However, in this case no access to the source code is required, as shown above. Still, the Worker itself has to be written by a software developer.

There are already a wide range of analyses possible, but still many additional Workers have to be created. These Workers should be part of the framework itself since they can be used for analyses in different areas. Possible Workers include clustering algorithms, different diagram types, etc.

The logic handling the data sets is currently focused on processing speed. For this the `DataLists` are often just copied and remain in the primary memory. For small data sets this is a fast and efficient way, but for complex databases with thousands or millions of entries this method can quickly reach the computer's capacity limit. Several solutions would be possible to meet this problem. The generated result could only be kept in memory during one operation until succeeding Workers have used the Output Data. This would slow down the analysis process since, even for a small adjustment, all Output Data would have to be recalculated. Additionally, the generated Output Data could be saved in a temporary file, so that main memory is freed because the Output Data is not used for calculation. Again, this would slow down the calculation process since disk operations are relatively slow.

5.2 A Visual Analytics Application for Temporal and Spatial Data

The creation of graphical analyses for a specific use case sometimes requires special methods or algorithms which are not possible or reasonable in a generic environment. In this case, a special analysis application created exactly for that particular use case is more reasonable than to try to integrate the features into a generic environment. One example for this is the tool which is explained in this following chapter.

The archaeo-related sciences deal with the analysis of rests of buildings, artifacts, human burial remains, or faunal and botanical remains. Excavated are settlements, such as houses, workshops, wells, waste heaps, latrines as well as sanctuaries and burial grounds with related artifacts and bones. Those are all remnants of human activities. They describe daily life, about nutrition, city planning, crafting techniques, interactions of different cultures, fashions, preferences and taboos, trade, forms of economy, migration, and animal kingdom.

In the process of archaeological excavations, ancient monuments are gradually excavated and thus destroyed. The task of archaeologists consists of documenting each step *lege artis* to reconstruct and describe each situation as exact as possible.

For their analyses, not only the data of the findings is important, but also the spatial position and temporal information since it can easily happen that during an excavation findings from completely different epochs are revealed. The position of findings in chambers, graves, etc. gives an insight into the temporal circumstances of the excavation site (typically, findings from older times are found below findings from more recent times). The relationship of the spatial and temporal information of the archaeological data allows the reconstruction of history.

It is common practice that archaeologists and bioarchaeologists analyze their findings by using statistical analysis applications. However, in archaeology and bioarchaeology it is not possible to combine the analysis results with the related spatial and temporal data of the findings automatically because this kind of data is usually not explicitly available in digitalized form (even though most excavation databases contain the necessary data to reconstruct this information). The model is usually prepared by hand in a time-consuming process resulting in drawings that represent the stratigraphy of the excavation site. These hand painted drawing sheets of the stratigraphy do not include any enriched information about the available findings in the layers. For each question the position of each layer has to be set in relation manually. This is time-consuming and error-prone.

Archaeologists and bioarchaeologists would greatly benefit from an automatic process to generate explicit stratigraphical models that can be visualized and analyzed by dedicated tools. This would not only save processing time, but would also allow further

analyses to be carried out for which the manual process is too complex.

In this chapter, we describe TARDIS ("Temporal and Relative Diagram Interaction System"), a tool that provides the possibility to present the extremely complex and multidimensional connections between place and dating as well as present development in time henceforth immensely simplifying the understanding of the connection of the data. It conveys an enormous amount of information in only a few descriptive and highly informative visualizations. TARDIS also features algorithms to make the implicit information hidden in common excavation databases explicit without a manual, time consuming, and error-prone generation (e.g. drawing).

In summary, the main contributions are as follows: First of all, we explain some basic archaeological terms that are related to this work (cf. Chapter 5.2.1). We discuss the background and the availability of data and the related work for this context (cf. Chapter 5.2.2). Then, we describe the structure of TARDIS and focus on the creation of the Harris Matrix as a key aspect (cf. Chapter 5.2.3). Afterwards, we use TARDIS in a case study to analyze real data from an archaeological excavation to determine the distribution of faunal remains (cf. Chapter 5.2.4). Finally, Chapter 5.2.5 concludes and presents some directions for future work.

5.2.1 Terminology

For understandability and relevance reasons, we introduce definitions from the archaeological context concerning the work below.

Archaeological notions

Excavation and spatial documentation of archaeological findings are – amongst others – usually managed in terms of areas, sections, and layers. According to internationally agreed guidelines for the documentation of archaeological excavations, see exemplarily [blf16]. These terms are defined as follows:

- **Layer**:
  Layers describe all structures that differ in color, consistence, and material of the directly neighbored structure, e.g. a monophased wall, a layer of earth, a discoloration of soil, but also a disruption. A layer is the smallest managed unit.

- **Area**:
  An area is a horizontal sector of the excavation that is defined by the archaeologist.

- **Section**:
  If required, single parts of an area can be split into sections.

Harris Matrix

The Harris Matrix [Har75, HIB91] is used to depict the temporal succession of archaeological contexts and thus the sequence of depositions and surfaces on an archaeological site. Two layers, i.e. their strata, can be set in temporal relationship dependent on their position.

All archaeological sites are subject to the laws of archaeological stratigraphy. Therefore, Harris formulated a set of four basic laws for stratigraphy for the archaeological context: [Har89]

1. **Law of Superposition** "assumes that the strata and layers are found in position similar to that of their original deposition".
2. **Law of Original Horizontality** "assumes that strata, when forming, will tend towards the horizontal".
3. **Law of Original Continuity** bases "on the limited topographical extent of a deposit or an interfacial feature".
4. **Law of Stratigraphical Succession**: "A unit of archaeological stratification takes its place in the stratigraphic sequence of a site from its position between the undermost (or earliest) of the units which lie above it and the uppermost (or latest) of all the units which lie below it and with which the unit has a physical contact, all other superpositional relationships being redundant."

By applying these laws, it is not necessary to sketch the relationships of each single layer to all the others. Instead, it is sufficient to consider three basic relationships between them: (A) Two layers are positioned on the same layer, but have no stratigraphic connection, (B) two layers are positioned one above the other, and (C) a layer is cut by another layer. A sketch of these relationships is shown in Figure 5.11. The representation of the Harris Matrix could also be called a diagram or graph, but we will continue to use it in this chapter since "Harris Matrix" was the chosen term by Harris.

5.2.2 Background and Related Work

In principle, our application TARDIS works on top of any excavation database providing implicit information on the spatial-temporal context of an excavation as described above. In this chapter, we will illustrate the capabilities of TARDIS on top of a real excavation database running at the *Bavarian State Department of Monuments and Sites*[34]: The database EXCABOOK [KLK+18e] is used to gather and store data of archaeological

[34] http://www.blfd.bayern.de

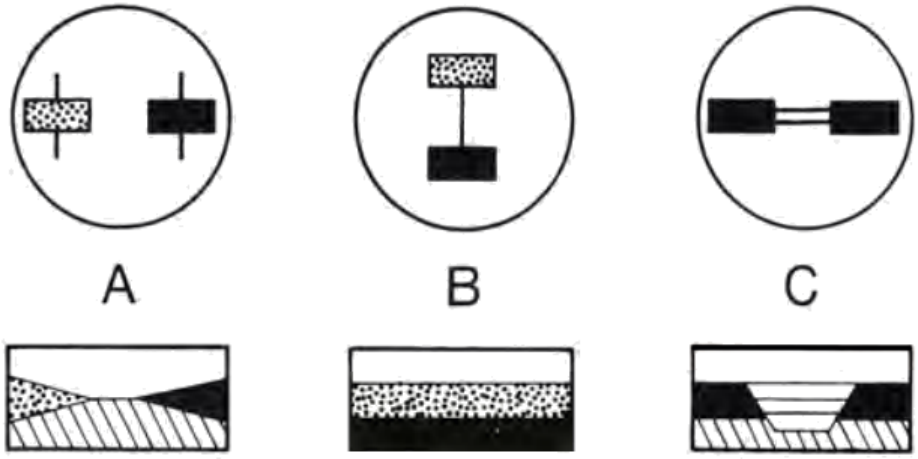


Figure 5.11: *"The Harris Matrix system recognizes only three relationships between units of archaeological stratification: (A) The units have no direct stratigraphic connection. (B) they are in superposition; and (C) the units are correlated as parts of an once-whole deposit or [layer] interface." [Har89]*

excavations. EXCABOOK includes detailed information about the layers and the single findings from an excavation.

Each layer is saved as an own data set that is assigned to a specific position correlation information which includes – amongst others – the information for area and section. Within each layer, this information can be set in spatial relationship by defining the position of the layer in relation to other layers, by defining which layer is positioned below/above another layer, which layer cuts/is cut by another layer, and which layers are disconnected, but identical. This information is sufficient to calculate the Harris Matrix [Har89] as described in Chapter 5.2.1.

In summary, temporal data is available in the archaeological data sets due to the definition of the stratigraphy, respectively to the temporal relation of single layers to each other. There is also available geodata for the data sets that can be taken from the spatial expansion and coordinates of areas and sections in excavations. This data is documented as precisely as possible. However, not all measurements have the same precision due to different measurement methods or level of accuracies. The data still can – at least – be used for a rough generation of a sketch of the excavation place.

The problems encountered from this context are the following:

- **No considering of spatial and temporal relationship:**
  This spatial and temporal information is not set in relation to the archaeological field of work. The archaeologists do either consider the temporal or the spatial point of view. However, combining both of the aspects would result in the exploration of new scientific questions in the archaeological context.

- **No interactive Harris Matrix:**
  The visualization of the data with a Harris Matrix does not offer any dynamic interaction for the user. It is a static result that is achieved mostly by the manual composition of data.

- **Finding information is not considered:**
  The information about the single findings are not considered at all in the Harris Matrix. If a user wants only to display a filtered data set – e.g. only a specific species, bone element, sex, etc. – the input of the data has to be filtered manually before loading them to a potential application.

Since the first proposal of the Harris Matrix in 1975 [Har75], some tools were developed to support the creation of the matrix. These tools allow the users to enter each layer and define the stratigraphy between them. Tools like the *Harris Matrix Composer* [Ser] or *ArchEd* [HMP+] allow the setting of these relationships directly in the applications. Other tools like *Stratify* [Her, Her11], alternatively support the import of a CSV file or dBASE database to define these information. The entered layers are checked for errors. If no errors exist, the resulting Harris Matrix is created. Still, these tools do not visualize the spatial position of the single layers. The actual findings inside these layers are also not considered in any way. In addition, none of these tools contain interactive elements.

Most often not all objects of an archaeological excavation site can be found. Natural circumstances (like trees) and man-made constructions (like buildings or walls), but also financial or time reasons, lead to selective or random excavations within one site. Therefore, statistical interpolation methods are used to get a representation of the distribution of objects on the excavation site. The one we focus in this chapter is the Kernel Density Estimation that is commonly used in the archaeological field of work. There exist many tools for Kernel Density Estimation that allow the generation of a distribution either directly online [Wes] or inside an application like MATLAB [Mat]. All existing tools offer various options to load data including CSV and database files. While it would be possible to filter the data by selected criteria, a 3D representation of the layers and the Harris Matrix cannot be displayed at the same time.

5.2.3 Illustration of the Spatial and Temporal Distribution of Findings

In this chapter, we introduce TARDIS, an application that supports the archaeologists in their data analyses concerning the challenges mentioned in Chapter 5.2.2. Its graphical user interface allows access to three major parts of the spatial-temporal context of the data from a given excavation.

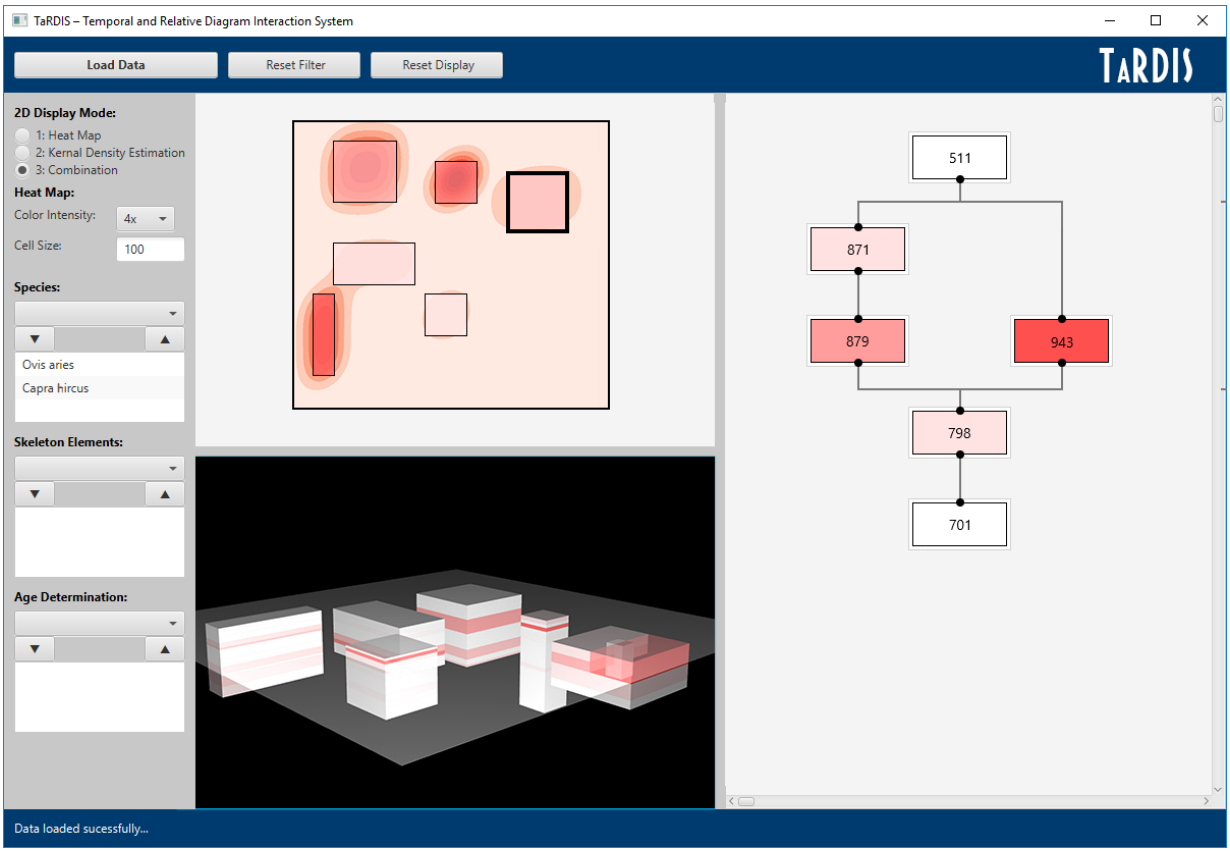


Figure 5.12: *Screenshot of the* TARDIS *application. The 2D representation (top center) shows the areas and sections and displays heat map colors for the absolute number of findings and the result of a Kernel Density Estimation in the background. The 3D representation (bottom center) shows the distribution of findings in the layers. The generated Harris Matrix is displayed on the right. Settings and filter options can be entered on the left.*

These includes:

- a 2D representation of the site,
- a 3D representation of the layers, and
- a (graph-based) visualization of the Harris Matrix.

Furthermore, an area for the selection of the data and filter possibilities are added to the interface so that each of the three views mentioned above can be customized according to which data should be displayed. A screenshot of this tool is shown in Figure 5.12 which also shows the elements described in this chapter. Below, we present the single areas of TARDIS in detail and describe the use of the elements.

5.2.3.1 2D Representation

In TARDIS, the 2D representation sketches the rough layout of an excavation site as archaeologists are used to. Therefore, the single sections are plotted by considering the corresponding coordinates that are saved in the database. Currently EXCABOOK only allows the definition of rectangle and circular shapes to describe the dimension of sections. Thus, TARDIS currently only supports the representation of these shapes, too. In future, other shapes – like the definition and use of custom polygons – will be considered as well to leverage the flexible definition of the shape of single sections.

This rough layout of the excavation site is complemented by a visual representation of the distribution of findings which is indicated by different colors. Therefore, there are three possibilities for the representation of the available findings that are described below.

- **Mode 1: Heat Map Colors**
  This possibility allows the visualization of the actual findings in the database by means of a heat map. The drawn rectangles and circles in the 2D representation are colored dependent on the actual available data in the corresponding area or section. The stronger the color, the more data of findings exist in the area or section. Therefore, the colorization represents the absolute number of findings.

- **Mode 2: Kernel Density Estimation**
  The actual position of the findings is taken to calculate an estimation of the distribution of findings on the excavation place. We use a Kernel Density Estimation as a well-established statistical method. The graphical representation of the estimation is drawn to the 2D representation as an own layer behind the rectangles and boxes. The border of the sections, respectively the forms, are still displayed.

- **Mode 3: Combination**
  A combination of the previous modes allows the visualization of the absolute number of findings and the estimation. The background of the visualization shows the result of the Kernel Density Estimation, but the absolute number of findings is still indicated by a colorization of the drawn rectangles and circles. This combination can be seen in the screenshot of Figure 5.12.

The user can decide which data representation is displayed. The 2D representation of the excavation site also serves as a selection of the areas and sections which information shall be displayed as an Harris Matrix.

The 2D representation simulates the traditional way how archaeologists represent the spatial context of an excavation. However, the archaeologists usually did this by manually drawing (simply because no tool was available to support a digitalized data

generation) so far. With TARDIS working on top of an excavation database like EXCABOOK, the generation of that representation is fully automated (data is gathered from the underlying database). In addition, TARDIS also allows the visual analysis of the spatial distribution of findings, implementing several filters, etc.

Thus, using TARDIS, archaeologists are now able to visually analyze the spatial relationships of findings in a large scale, e.g. considering different categories of findings, etc.

5.2.3.2 3D Representation

Similar to the 2D representation, the 3D representation does also sketch the layout of the excavation site, but it adds the depth information to the display. This is a completely new feature to many archaeologists since the sketch of the 3D spatial context of an excavation is recorded not too often because it has to be done again manually so far which is very complex and time consuming.

The x- and y-axis of the 3D representation is identical with the information of the 2D representation, but the z-axis was added to consider the height/depth information as well. These are taken from the data of layers automatically. However, for a realistic display of the stratigraphy the stored data of layers is not sufficient. Actually, only the rough dimension of a layer can be used. This can only be an approximation for the vertical value because a layer can have different thicknesses at each spot. According to this, the 3D representation is not to be seen as a detailed drawing, but as a sketch instead. However, the sketch provides a rough spatial impression of the distribution of findings for the archaeologists which is – for many projects/excavations – much more than previously available.

The distribution of findings can also be displayed in the 3D representation, but – in contrast to the 2D representation – for each single layer. Here, the stronger colorization of a layer also indicates a higher frequency/density of findings. Therefore, the user gets an overview of the distribution of findings through the physical position of the stratigraphy. This is supported by the possibility to arbitrarily move through the 3D scene with mouse and keyboard.

Like the 2D representation, the 3D representation is a huge step ahead for the archaeologist when analyzing the spatial distribution of findings. While so far most archaeologists are limited to 2D manual drawings that are not really flexible in terms of changing views through filters, etc., TARDIS now leverages visual analytics of the spatial context of an excavation at full scale.

5.2.3.3 Harris Matrix

In the context of an archaeological excavation, not only the spatial distribution of findings, but also the temporal distributions and relationships of items are very important. As described in Chapter 5.2.1, the Harris Matrix displays the temporal relationships of different layers. Each single box of the Harris Matrix represents a layer from a specific area or section. The higher a box is displayed in the diagram, the more recent is the layer. At the same time, an earlier layer can be recognized by a lower position in the diagram.

The Harris Matrix itself is an important representation of the temporal context of an excavation. However, since this representation is typically drawn manually, it is limited in the way it can be analyzed.

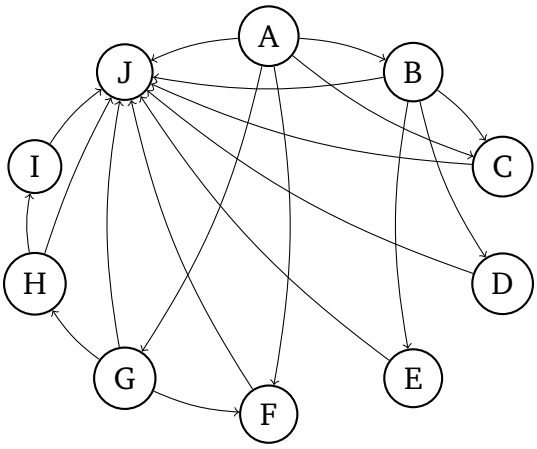
Thus, in TARDIS, the visualization of the Harris Matrix can be extended by displaying the distribution of findings over the elements of the diagram. By selecting a specific area in the 2D representation, the corresponding Harris Matrix is generated and displayed in the application. Each box in the Harris Matrix is highlighted with a color that represents the frequency/density of findings in this layer. In comparison to the 2D and 3D representation, the colorization in the Harris Matrix illustrates the temporal distribution of these findings. Again, appropriate filters allow the visualization of different categories, etc.

The Harris Matrix in TARDIS also features interactive elements. By selecting one box in the Harris Matrix, the 3D representation will highlight the corresponding layer. Therefore, the archaeologist can identify the spatial position of the layer and compare it with the temporal position of the Harris Matrix.

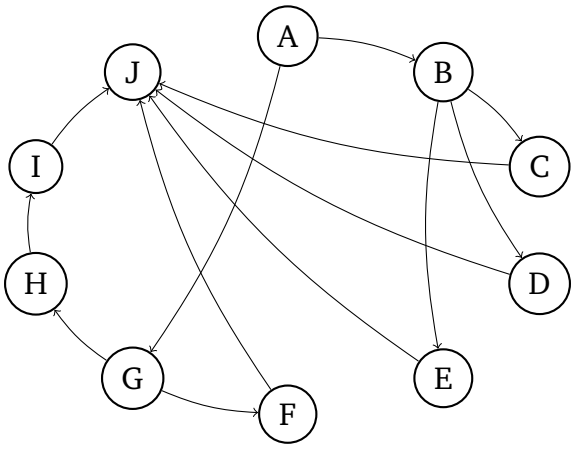
Below, we describe the set of algorithms necessary to generate the Harris Matrix. The processing of the algorithms is illustrated in Figure 5.13.

To generate a valid Harris Matrix, we need suitable data. Each layer has to be connected to the network, either by being earlier, later, or in the same time as another Layer from that data. Layers that are in the same time as other layers can be identified by having the same layers above and below. We start off by checking the available data (containing connections and names of layers) in the database by looking for loops and reporting them back so that they can be removed. This is done by checking if any layer is, via connections, later or earlier than itself. If so, this is a contradiction and indicates a loop that needs to be eliminated. Also in this step, we can already assign depths to the nodes of the diagram. Algorithm 5 implements the identification of loops and the assignment of depths to nodes.

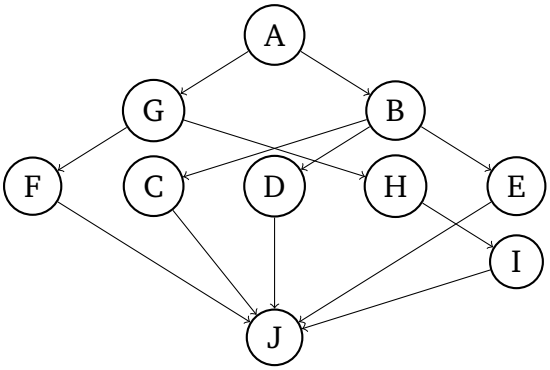
After we eliminated the loops, we eliminate redundant links. This is important to get a Harris Matrix that explains the excavations temporal relations with as little connections as possible. Otherwise, redundant nodes and connections produce a possibly huge overhead and an excessive visualization that is hard to comprehend. To ensure, we do not



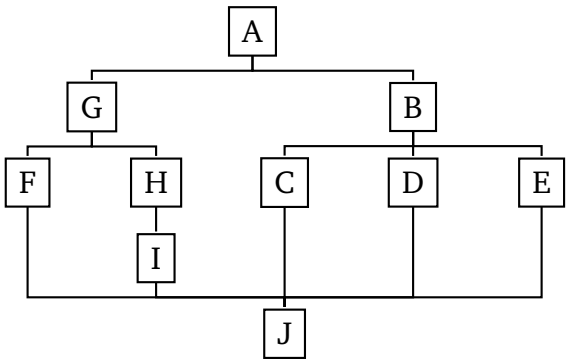
Step 1: Removal of loops from data



Step 2: Removal of redundant links



Step 3: Sorting for height of nodes



Step 4: Minimizing crossings

Figure 5.13: *Illustration of the algorithms described in Chapter 5.2.3.3 to create a Harris Matrix.*

have any links that are not necessary, we check each direct link if we can reach the same layer transitivity. If this is possible, we remove the direct link. Algorithm 6 implements this step.

Then we need to determine the height and the width that the Harris Matrix will need. In this process we have also to take into consideration whether there will be a connection going through to a lower height but does not have a layer at that specific height. This is necessary so that our Harris Matrix is clear and all connections are visible. This process is implemented in Algorithm 7.

Once we have the height and width of the Harris Matrix, we can start sorting the layers in a manner that requires as little path crossing as possible. For this purpose, we sort the layers below in a way that they lie underneath the layer above and, if needed, in the right position if they are below multiple layers. In the sorting process we also need to consider empty spaces which are required if a connection stretches above multiple

```
1  topnode ← all nodes without incoming connection(s);
2  foreach topnode do
3  |  assign height to node;
4  |  foreach node below do
5  |  |  goDown(height++, nodeID);
6  |  end
7  |  if not all nodes have a height then
8  |  |  throw error found nodes without connection;
9  |  end
10 end
11 def goDown(height: int, node: Node) : void:
12 |  if height > nodes.length then
13 |  |  throw error loop in data;
14 |  end
15 |  if height(nodeID) < height then
16 |  |  assign height to node;
17 |  end
18 |  foreach node below do
19 |  |  goDown(height++, nodeID);
20 |  end
21 end
```

Algorithm 5: *Checking connectedness and loops*

layers of depth without an actual layer in them. The final preparation of the data is implemented in Algorithm 8.

After we have the data well prepared, we can then finish the generation of the Harris Matrix by drawing the layers and their connections.

5.2.3.4 Filter

In general, the displayed distribution of findings is not filtered. The colorization of the elements in the 2D and 3D representation and in the Harris Matrix is reflecting the distribution of all available findings, independent on their diversity.

Still, TARDIS provides some basic filter settings that can be applied to the loaded data, e.g. – in the case of zooarchaeological studies – filter for specific species, skeleton elements, or age determinations. Once a filter option is set, the colorization in the visualizations only considers the findings matching the filter settings.

This allows multiple views on the spatial-temporal distribution of findings within an excavation and leverages visual analyses at large scale that enables archaeologists to find insights on a completely new level.

```
1 foreach node do
2 |  if there are multiple connections downwards then
3 |  |  remove one and save ID;
4 |  |  follow the other path(s) downward;
5 |  |  if ID is not encountered then
6 |  |  |  reinsert ID;
7 |  |  end
8 |  end
9 end
```

Algorithm 6: *Check for redundant links*

```
1  maxheight ← max(height);
2  sort nodes according to height;
3  start from top do
4  |  columnwidth ← 0;
5  |  check nodes one layer above for links to nodes below that are not in this
   |   height;
6  |  foreach of those do
7  |  |  increase columnwidth by 1;
8  |  end
9  |  foreach in this height do
10 |  |  increase columnwidth by 1;
11 |  end
12 |  maxcolumn ← max(columnwidth, maxcolumn);
13 end
```

Algorithm 7: *Sort for height of nodes*

```
1  insert first layer into Harris Diagram in random order;
2  foreach layer below do
3  |  add nodes in the order that they are referenced above;
4  |  if nodes are not in this layer then
5  |  |  add a filler;
6  |  end
7  |  if nodes are referenced in multiple nodes above then
8  |  |  put these nodes in the middle;
9  |  end
10 end
```

Algorithm 8: *Sort nodes to minimize crossings*

5.2.4 Case Study: Distribution of Faunal Remains

To prove and demonstrate the usefulness of the tool as well as the quick and efficient way to show the distribution of animal species in the different features of an archaeological excavation, we run a case study on real archaeological data. Our goal is to compare the spatial and temporal distribution of animal bones excavated in several shafts, not only within the features, but between them as well.

We took data of the excavation Marienhof-Haltepunkt in Munich, Germany, excavated by the archaeological excavation company ReVe during the years 2011 and 2012. The major part of the 110 $\times$ 95 m excavated area lies within the oldest city core from the 12$^{\text{th}}$ century whereas the northern peripheral zone is a part of city's expansion from the late 13$^{\text{th}}$ and early 14$^{\text{th}}$ century. Marienhof-Haltepunkt is a promising study object, due to the good condition of the animal remains as well as of the shafts themselves. The fact that flotation of material was practiced during the excavation is a great benefit for the zooarchaeologists. Thereby, smaller and more fragile bones were able to be recovered. This results not only in a higher amount of different species (e.g. rodents and small birds), but also of age groups which are usually underrepresented, especially the age groups "fetal", "neonat", and "infantile".

We chose shaft 5 of the excavation that is illustrated in the drawing sheet of Figure 5.14 and visualized the data of the shaft in the TARDIS application – the distribution of all (unfiltered) species is shown in Figure 5.15-A. Shaft 5 was constructed as an elaborate well in 1261 [Wes14], but it was abandoned short after and functioned as a latrine. Such an elaborately built well has to have belonged to a wealthy owner.

The bones of frog species depict a good example of a result due to a carefully done excavation. Two distinct species of frogs, *Rana temporaria* (the common frog) and *Pelophylax lessonae/ridibunda* (the pool/marsh frog), could be identified, although the majority of the frog bones could be identified only to the family level (*Ranidae*). In the TARDIS visualization we filtered the available data for frogs (as seen in Figure 5.15-B) and recognize a high amount of frog bones within the shaft. The high amount of frog bones from these layers is not such an unusual phenomenon in medieval archaeology. Frogs tend to fall in shafts and are not able to get out. Another possible explanation would be the consume of frogs even if its evidence in medieval complexes is not yet rendered. In the well of the excavation "Altstadt" in Villingen, Germany, from the 13$^{\text{th}}$/14$^{\text{th}}$ century 315 bones of *Rana temporaria* as well as 3 bones of *Bufo bufo* (the common toad) were found [vdDK14]. As the authors state, all of them seem to have been fallen in the well. This death assemblage which is not caused by predators or, as would be in our case, humans called *thanatocoenosis*.

Frog bones were found for the first time in layer 801 of shaft 5 although in small numbers ($n = 4$). The highest **N**umber of **I**dentified **Sp**ecimens (=NISP) however can be

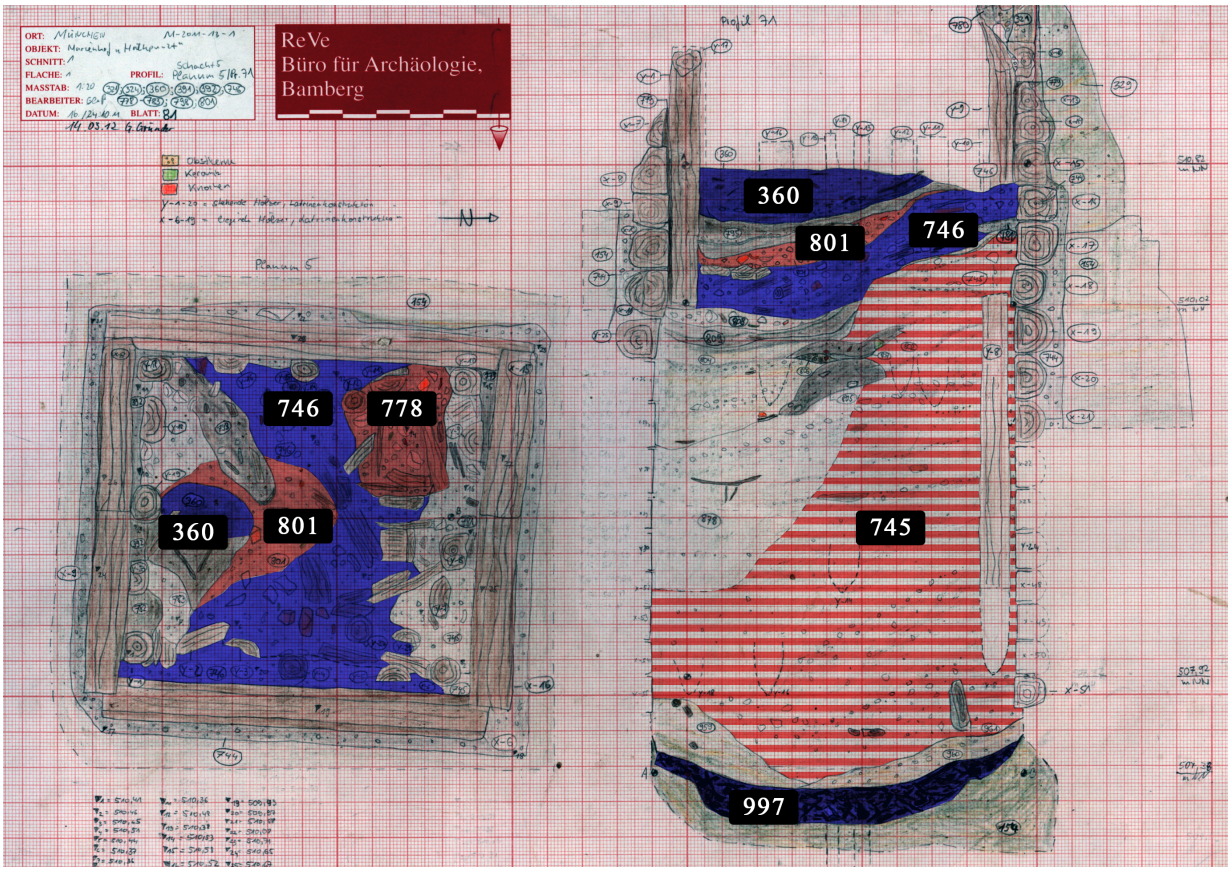


Figure 5.14: *Planum (left) and profile (right) drawing of shaft 5 of the excavation Marienhof-Haltepunkt in Munich, Germany. Drawing: ReVe, Büro für Archäologie, Bamberg, Germany. The layers 360, 745, 746, 778, 801, and 997, that are mentioned in Chapter 5.2.4, are highlighted. The mentioned layer 393 is not visible in the drawing.*

found in layer 746. Here the density of frog bones is at highest ($n = 464$). However, if we take a look on the faunal material of layer 360, which superimposes layer 746, we can see that there are only 37 frog bones. In the other layers, frogs are found as well, but also in lesser numbers. The high density of frog remains in layer 746 makes clear that many frogs or rest of frogs were fallen respectively thrown in the shaft in a specific time period. Since the shaft has a dual story (at the beginning as a well and afterwards as a latrine) we could hypothesize that the distribution of the frog remains reflect the special story of this shaft.

This dual story is best reflected in the composition of the layers 997 and 745. As can be clearly seen in Figure 5.15-C, the bones of cattle are most abundant in layer 997 while the bones of other species are underrepresented. In the superimposed layer 745 there is only a small number of bones ($n = 73$) with small ruminants being most abundant.

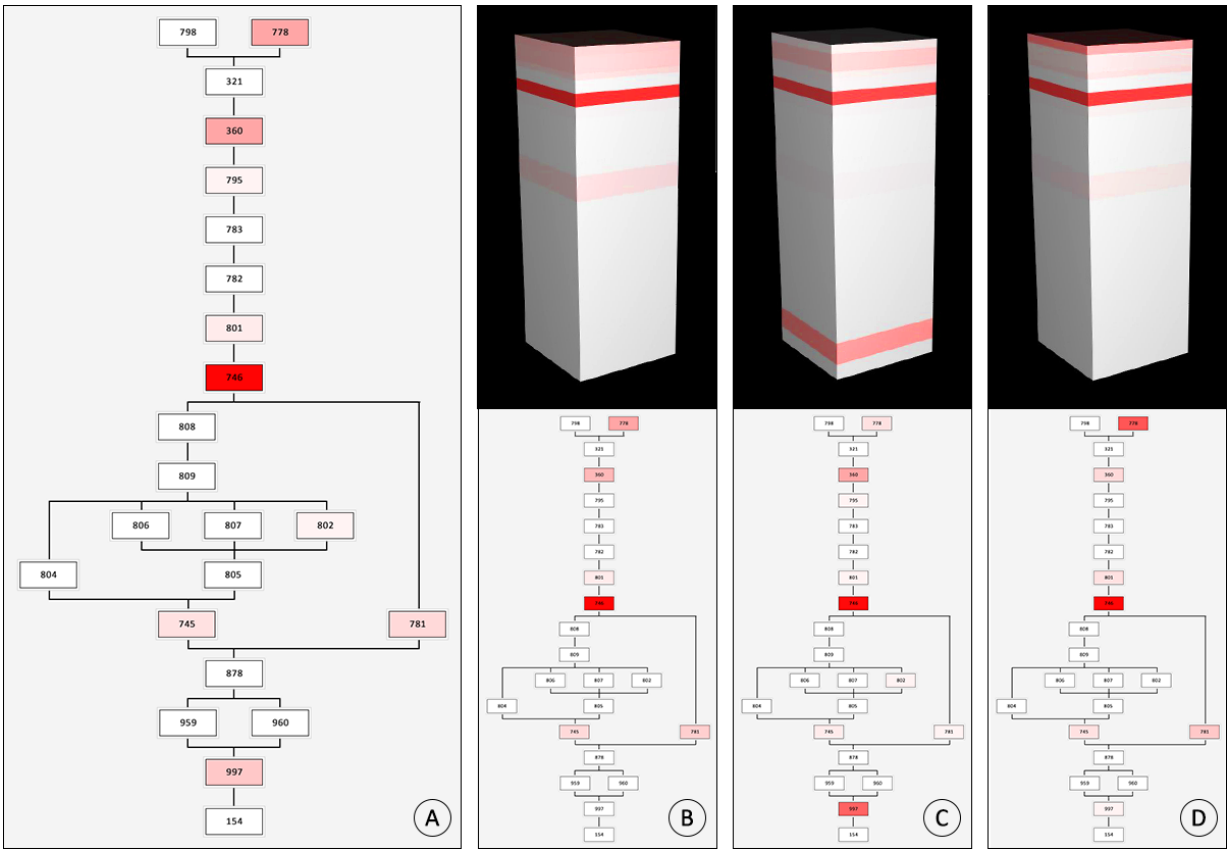


Figure 5.15: *The 3D representation and Harris Matrix of data from shaft 5 of the excavation Marienhof-Haltepunkt in Munich, Germany. The shown Harris Matrix (A) is the temporal structure of the drawing sheet in Figure 5.14 with the unfiltered distribution of findings. The other illustrations show the 3D representation of the shaft and the distribution of filtered findings of (B) frogs, (C) cattle, and (D) dogs/cats.*

Interestingly enough, the NISP is remarkably small even though layer 745 is the biggest of all. If we add the fact that there are virtually no other finds in the layer besides the bones one could wonder about the purpose of such a layer. The answer lies in layer 997. The cattle bones of layer 997 belong solely to one skeleton. This skeleton of a cow (as can be clearly seen on the pelvis) was fallen respectively thrown into the shaft when it was still a well. Because of water contamination, people decided to backfill the well, hence the superimposed thick layer 745 where almost no findings were made. The visual generation through TARDIS makes it easy to spot such deviations and allows the user to quickly uncover interesting findings.

In TARDIS, we also filtered the available data for remains of cats and dogs, as visualized in Figure 5.15-D. In the subsequent layers, such as layer 778 and especially layer 746, we identified many bones of cats and dogs. Again, this is a typical phenomenon in

medieval archaeology as many small animals were disposed in wells and latrines. This was for instance the case in the aforementioned well of "Altstadt" in Villingen, Germany, where six dog skeletons were uncovered [vdDK14]. In shaft 5 at least two partially preserved dog skeletons were identified. More abundant are the remains of cats. Many of them were juveniles and were disposed in the shaft, because they were "superfluous". Again, thanks to the quick visualization through TARDIS such findings can be quickly detected.

Another interesting aspect is the economic importance of livestock in the middle ages. The most important ones were cattle, pig, small ruminants (sheep or goats[35]) as well as, on a smaller scale, chicken. Trends, which can be quickly generated in TARDIS, are based on the distribution of the bones of each animal. The higher the number, the more (economically) important the animal.

In shaft 5 the raising importance of small ruminants is recognizable. While the NISP of cattle and small ruminants is balanced in layer 746, the relative share of small ruminants increases in the higher positioned layers 393 and 360. In layer 360 the domination of small ruminants becomes apparent.

As we have seen in this case study, statements on the development of the economic importance of the different livestock animals within subsequent time periods can quickly be done with TARDIS, a new application to visually analyze archaeological data.

5.2.5 Conclusion and Discussion

In this chapter, we described the application TARDIS, a visual analytic system for spatial and temporal data in the archaeological field, motivated by the challenge to set spatial and temporal data from excavations in relation. Therefore, we generated 2D and 3D representations of the excavation site that are enriched with data of findings to indicate their location. Furthermore, we generated a Harris Matrix that is used to illustrate the temporal positions of the findings. We also explained the implementation of the generation of the Harris Matrix. These components are interactive, so archaeologists and bioarchaeologists can set the illustrated results in relation. Finally, we applied data from a real excavation site to TARDIS to point out the usefulness of the application.

While this application has already many benefits for the domain experts, there are also some improvements and extensions that can be addressed in the future:

The 2D representation of the excavation site shall consider the definition of other shapes than rectangular and circular ones. Therefore, the plotting of new shapes has to be supported, like polygon shapes. That is important to describe more complex structures

[35]The morphological distinction of the bones of sheep and goat is not always possible in the zooarchaeological context, zooarchaeologists often use the term 'small ruminants' instead of 'sheep or goats'

of linear excavation sites, e.g. for motorway routes or pipelines, but also misshaped areas or sections on a smaller excavation site.

The Kernel Density Estimation is integrated to the application as one example for a statistical analysis. In future, other statistical methods shall be added to the 2D representation as well to offer a wide selection of distribution information. The 3D representation could also be extended with statistical analysis methods that also consider the third dimension.

The filter options that are currently realized are the most important options in the archaeo-related work. However, the filter options can be extended to also consider other information from the excavations, possibly also supporting a custom filtering with user-defined attributes.

The application should be extended by a feature to display two or more illustrations that show data of different filter settings. This would help the archaeologists to visually compare two data compositions side by side.

Finally, the real position of the layers in the areas shall be plotted more detailed in the 3D representation. Currently the layers are displayed as a rough estimation. The level of detail could be increased especially if layers are cut or are arranged side by side.

Acknowledgement

We thank *Grabungsfirma ReVe, Büro für Archäologie*[36], Bamberg, Germany, in particular Barbara Wührer, and Dr. Christian Behrer of *Büro für Denkmalpflege*[37], Regensburg, Germany, for providing the stratigraphic data and archaeological information of the excavation Marienhof-Haltepunkt, Munich, Germany, used for the use case in this chapter.

[36] http://www.reve-archaeologie.de
[37] http://www.denkmalpflege.biz

Chapter 6

Conclusion and Discussion

Attribution

This chapter does not use any material from previous publications.

In this thesis, we proposed an e-Science architecture to collect, share, retrieve, and analyze scientific data as can be seen in Figure 6.1.

First, in Chapter 2, we introduced the framework XBOOK which enables the creation of graphical database applications to collect and share scientific data. We summarized the historical development of the zooarchaeological database OSSOBOOK to the framework XBOOK and gave an overview of the features of XBOOK. Then, in Chapter 3, we described the synchronization, a vital part of XBOOK, in more detail. The synchronization enables scientists to enter data everywhere, even if no Internet connection is available. The data can be synchronized later, to collaborate, share, and back-up the entered information.

While working on a database for zooarchaeological data, it became clear that the demand for such an infrastructure is not only required in archaeo-related disciplines and sub-disciplines, but also in other scientific areas. Of course, several databases (mostly spreadsheet) were already used, but they were mostly designed for the personal usage or for a very limited group. Therefore, this recorded data could only be shared with difficulty since each data scheme is individual and often is not compatible with other data schemes. A possibility to aggregate data from different data sources did not exist or at least not in a way that it was usable. After we created several databases for the archaeo-related contexts, the scientists demanded to be able to combine these different data sources in a search query.

A approach for the demand is presented in Chapter 4, a new architecture for the retrieval of heterogeneous data from anonymous, distributed data sources. This architecture enables data owners to share their data while remaining full control over their data. In Chapter 4.7, we describe REMIS CLOUD that extends the REMIS architecture to

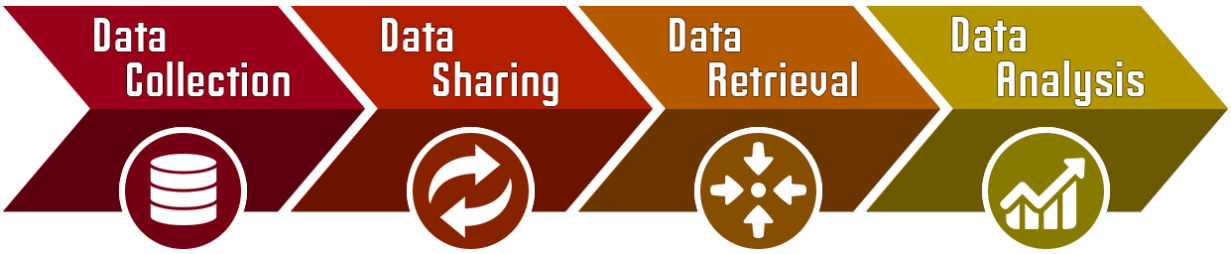


Figure 6.1: *A visualization of the workflow of scientific work as described in this thesis.*

allow the retrieval of data from different data sources of diverse areas without complex queries.

While the origin of the REMIS architecture was for the archaeo-related domains, it became clear that other sciences could also benefit from the overall concept of the infrastructure while special utility functions can be added if having a specific domain in mind.

We hope to have contributed to a solution for the problems mentioned in this thesis by creating XBOOK and REMIS, so that data owners are now able to keep storing their data in their own databases and can share the data with other scientists. They can use the data for analyses with a larger data record. This helps in creating analyses which are more comparable and can be statistically more meaningful.

Finally, in Chapter 5, we first presented an embeddable ANALYSIS TOOL which enables the execution and creation of analyses inside a base application. This allows scientists to work inside a familiar working environment and therefore encourage them to try new types of analyses which might lead to new knowledge. In the second part of the chapter, we introduced TARDIS, a standalone analysis application for the analysis of archaeological and zooarchaeological data with their spatial and temporal information with the help of a Harris Matrix.

With the development of the ANALYSIS TOOL, we also want to satisfy the demand for the possibility to run statistical analyses directly inside the application. The first approach for these statistical analyses with plug-ins [Loh11] encountered the problem that the plug-ins had to be updated as soon as the database scheme was changed. This was required because the plug-ins were designed to use individual implementations for the necessary SQL queries for their analyses. These queries had to be updated as well as soon as the database scheme was changed.

For every statistical analysis, a new plug-in had to be created as well. Thereby, the more analysis methods were implemented, the more maintenance had to be done when a new database version was published. Today, the creation of individual plug-ins is still possible with the ANALYSIS TOOL, but since the database scheme is abstracted from the plug-ins and the plug-ins are connected with an API, they do not have to be updated

anymore once they have been implemented.

With the help of the ANALYSIS TOOL, scientists can now analyze their data, without first having to export the data and then to re-import it inside the application in which the analyses would be carried out otherwise.

This allows quick analyses, for example of the distribution of animals of an excavation in a zooarchaeological context or the average weight of patients in a clinical study.

Nevertheless, this all is only the first step. Each area is a useful addition on its own, but they have to be linked to use the full potential of the combined infrastructure. For example, the server of XBOOK could directly be a Connector Application for the configured REMIS architecture to provide the data which is entered to the system. This would greatly help offering a simpler distribution and wide availability for the REMIS. However, the databases of XBOOK would require some changes to allow project owners to configure whether they want to make their data available and to which degree. Since the user base in XBOOK databases might be too extensive and diverse, the server administrator cannot decide which data shall be made available for the REMIS. The users should then be able to define in the project settings if they want to make the project available. Additionally, individual data sets could also be handled individually. Then, these right settings would be regarded by the REMIS to decide if a data set is available for data requests inside the system.

Another important connection is the linkage of the ANALYSIS TOOL to the REMIS. It should be possible to use data from other sources in the ANALYSIS TOOL without first having to download it or importing it into the base application. The simplest solution would be to create a new type of Retriever which is able to read spreadsheet or CSV files. This would also allow to use old data that was not saved inside a database, or exports of databases from other database application solution which are not connected to the REMIS to be used for analyses. The REMIS would then be searched for suitable data sets in a dedicated screen. The results could then be saved as a spreadsheet file and with the new Retriever be used in an analysis. Alternatively, the ANALYSIS TOOL could be extended with a new Worker that allows to search for the desired data in REMIS. The Minimal Search Parameter would be entered with the help of a property inside the Worker. If the same analysis is re-executed later, the same data is retrieved from the REMIS again, provided that the data in the connected data sources has not changed.

Especially for data retrieved over the REMIS CLOUD, the import of external data or direct integration of the system in the ANALYSIS TOOL would be a necessary addition before the infrastructure could be used in practice. Since the data scheme of the retrieved data certainly is not compatible, the possibility to first import the data into the base application is not possible at all. Therefore, the dynamic retrieval of domain-extrinsic data enables considering interdisciplinary data in analyses without having to store and handle the retrieved data. This is time-saving and enables complex data analyses.

With the integration of the REMIS in the ANALYSIS TOOL, it is possible to create new and improved analyses which were not – or at least only with great effort – possible. For example, scientists might be interested in considering data from different sub-domains. These could have an attribute in common (like location information or identifiers) which also would be part of the Minimal Search Parameter. Previously, the scientists had to get in contact with the scientists responsible for the individual sub-domain to receive the data manually. This is time-consuming and cumbersome. With the help of the REMIS, the same analyses – subject to the condition that the data sources of the sub-domains are connected to the REMIS– can be carried out in minutes, instead of days or weeks. Also, only one scientist is required which saves a great amount time.

TARDIS could also benefit from a connection to other systems. So, TARDIS could be converted into a specialized Worker in the ANALYSIS TOOL which expects a predetermined scheme of data. But in addition to the graphical display provided by TARDIS, the calculated results could be prepared as a data output in the Worker which then could be used in further analyses. This way TARDIS could also serve as a graphical filter possibility (e.g. by selecting layers in the interactive Harris Matrix or the site plan) which result can be considered in analyses of the ANALYSIS TOOL.

The data that is used in TARDIS could be prepared with help of the ANALYSIS TOOL. This would allow a more detailed filtering of values which is not possible in TARDIS directly. The attributes of the objects that are correlated to the different layers in TARDIS could also be changed, e.g. instead of using the name of the object, the material or other attributes could be selected.

Most of the concepts covered in this thesis are currently only a basic implementation that already provide a great benefit for the scientific work on their own.

1. **Data Collection:** A dynamic framework for database applications for the collection of scientific data.

2. **Data Sharing:** A synchronization method for collaborating, sharing, and backing up data.

3. **Data Retrieval:** An information system to retrieve heterogeneous, but related data from distributed data sources.

4. **Data Analysis:** An embeddable tool for analyses to offer different analysis options.

But especially the joint reflection and the interaction of all components discussed in this thesis holds a large research potential for both, the scientific end users and the computer sciences. If the single components work as one, built upon each other, and are perfectly matched, then not only the single scientist can profit in the end, but the whole scientific community.

References

[ABF08] Paulo Sérgio Almeida, Carlos Baquero, and Victor Fonte. Interval tree clocks. A Logical Clock for Dynamic Systems. In *International Conference On Principles Of Distributed Systems*, pages 259–274. Springer, 2008.

[ASB+08] Steve Androulakis, Jason Schmidberger, Mark A. Bate, Ross DeGori, Anthony Beitz, Cyrus Keong, Bob Cameron, Sheena McGowan, Corrine J. Porter, Andrew Harrison, Jane Hunter, Jennifer L. Martin, Bostjan Kobe, Renwick C. J. Dobson, Michael W. Parker, James C. Whisstock, Joan Gray, Andrew Treloar, David Groenewegen, Neil Dickson, and Ashley M. Buckle. Federated repositories of x-ray diffraction images. *Acta Crystallographica Section D: Biological Crystallography*, 64(7):810–814, 2008.

[BC12] Reed S. Beaman and Nico Cellinese. Mass digitization of scientific collections: New opportunities to transform the use of biological specimens and underwrite biodiversity science. *ZooKeys*, (209):7, 2012.

[blf16] *Vorgaben zur Dokumentation archäologischer Ausgrabungen in Bayern. August 2016*. Bayerisches Landesamt für Denkmalpflege, Munich, Germany, 2016. Available from: `http://www.blfd.bayern.de/medien/dokuvorgaben_august_2016.pdf`.

[BSO] BSON. `http://bsonspec.org/`. [Online; accessed 23-January-2018].

[Dan10] Svetlana Danti. Cluster Analysis of Features of Animal Bones and Similarity Search on Multi Instance Objects of the Archaeozoological Data Pool. Diploma thesis, Ludwig-Maximilians-Universität München, Munich, Germany, 2010.

[Fid88] Colin J. Fidge. Timestamps in Message-Passing Systems that Preserve Partial Ordering. In *Proceedings of the 11th Australian Computer Science Conference*, volume 10, pages 56–66, 1988.

[Fou] The Apache Software Foundation. HDFS Architecture. http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html. Online, retrieved 18 May 2018.

[Gö18] Reiner Göldner. Resümee zum Workshop 'Digitale Grabungsdokumentation – objektiv und nachhaltig', 2018. Available from: http://bit.ly/2Cjk07N.

[GAA+08] Daniel Gardner, Huda Akil, Giorgio A. Ascoli, Douglas M. Bowden, William Bug, Duncan E. Donohue, David H. Goldberg, Bernice Grafstein, Jeffrey S. Grethe, Amarnath Gupta, Maryam Halavi, David N. Kennedy, Luis Marenco, Maryann E. Martone, Perry L. Miller, Hans-Michael Müller, Adrian Robert, Gordon M. Shepherd, Paul W. Sternberg, David C. Van Essen, and Robert W. Williams. The neuroscience information framework: A data and knowledge environment for neuroscience. *Neuroinformatics*, 6:149–160, 2008.

[GHK+15] Raymond Gallardo, Scott Hommel, Sowmya Kannan, Joni Gordon, and Sharon Biocca Zakhour. *The Java® Tutorial. A Short Course on the Basis. 6th Edition*, pages 124–125. Addison-Wesley, 2015.

[Goe] Brian Goetz. Java theory and practice: Thread pools and work queues. https://www.ibm.com/developerworks/java/library/j-jtp0730/index.html. [Online; accessed 06-December-2017].

[Gol14] Joachim Goll. *Architektur- und Entwurfsmuster der Softwaretechnik*. Springer, 2014.

[Gooa] Google Developers. FlatBuffers. https://google.github.io/flatbuffers/. [Online; accessed 23-January-2018].

[Goob] Google Developers. Protocol Buffers. https://developers.google.com/protocol-buffers/. [Online; accessed 23-January-2018].

[GS02] Rajat P. Garg and Ilya Sharapov. *Techniques for Optimizing Applications: High Performance Computing*. Prentice Hall Professional Technical Reference, 2002.

[Har75] Edward C. Harris. Stratigraphic Analyses and the Computer. *Computer Applications in Archaeology*, 3:33–40, 1975.

[Har89] Edward C. Harris. *Principles of Archaeological Stratigraphy*. Academic Press, London and San Diego, 1989.

[Her] Irmela Herzog. Stratify. http://www.stratify.org/. Computer Software.

[Her11] Irmela Herzog. Possibilities for Analysing Stratigraphic Data. In *CD of the Workshop "Archäologie und Computer" held in Vienna, Austria, 2011*, 2011.

[Her14] Friederike Herzog. Zweiter Jahreszyklus gestartet: Kuckucke mit Satellitensendern. *Der Falke. Journal für Vogelbeobachter*, 61:23–24, 2014.

[HHO+16] Rick Herrick, William Horton, Timothy Olsen, Michael McKay, Kevin A. Archie, and Daniel S. Marcus. Xnat central: Open sourcing imaging research data. *NeuroImage*, 124:1093–1096, 2016.

[HIB91] Edward C. Harris, Marley R. Brown III, and Gregory J. Brown. *Practices of Archaeological Stratigraphy*. Academic Press, London and San Diego, 1991.

[HM85] Dennis Heimbigner and Dennis McLeod. A federated architecture for information management. *ACM Transactions on Information Systems*, 3(3):253–278, 1985.

[HMP+] Christoph Hundack, Petra Mutzel, Igor Pouchkarev, Barbara Reitgruber, Barbara Schuhmacher, and Stefan Thome. ArchEd. A program for drawing Harris Matrices. `https://www.ac.tuwien.ac.at/files/archive/ArchEd/`.

[Inm05] William H. Inmon. *Building the Data Warehouse, 4th*. John Wiley & Sons, 2005.

[Jon16] Alex Jones. Top 10 Data Analysis Tools for Business. `http://www.kdnuggets.com/2014/06/top-10-data-analysis-tools-business.html`, 2016. Online, retrieved 3 May 2016.

[JSOa] RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format. `https://tools.ietf.org/html/rfc8259`. [Online; accessed 23-January-2018].

[JSOb] The JSON Data Interchange Syntax: Standard ECMA-404, 2nd Edition / December 2017. `http://www.ecma-international.org/publications/standards/Ecma-404.htm`. [Online; accessed 23-January-2018].

[Kal11] Daniel Kaltenthaler. Design and Implementation of a Graphical User Interface for the Archaeozoological Database OssoBook. Project thesis, Ludwig-Maximilians-Universität München, Munich, Germany, 2011.

[Kal12] Daniel Kaltenthaler. Visual Cluster Analysis of the Archaeological Database OssoBook with special focus on Data Integrity and Consistency. Diploma thesis, Ludwig-Maximilians-Universität München, Munich, Germany, 2012.

[KH14] Steven Krauwer and Erhard Hinrichs. The CLARIN research infrastructure: resources and tools for e-humanities scholars. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, pages 1525–1531. European Language Resources Association (ELRA), 2014.

[KKO+09] Hans-Peter Kriegel, Peer Kröger, Henriette Obermaier, Joris Peters, Matthias Renz, and Christiaan van der Meijden. OSSOBOOK: database and knowledgemanagement techniques for archaeozoology. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009*, pages 2091–2092, 2009.

[KL18] Daniel Kaltenthaler and Johannes-Y. Lohrer. The Historic Development of the Zooarchaeological Database OssoBook and the xBook Framework for Scientific Databases. *ArXiv e-prints: 1801.08052*, January 2018.

[KLK+15] Daniel Kaltenthaler, Johannes-Y. Lohrer, Peer Kröger, Christiaan van der Meijden, and Henriette Obermaier. Synchronized Data Management and Its Integration into a Graphical User Interface for Archaeological Related Disciplines. In *Design, User Experience, and Usability: Users and Interactions - 4th International Conference, DUXU 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015, Proceedings, Part II*, pages 317–329, 2015.

[KLK17] Daniel Kaltenthaler, Johannes-Y. Lohrer, and Peer Kröger. Supporting Domain Experts Understanding Their Data: A Visual Framework for Assembling High-Level Analysis Processes. In *11th International Conference on Interfaces and Human Computer Interaction 2017, Lisbon, Portugal, 2017*, pages 217–221, 2017.

[KLK+18a] Daniel Kaltenthaler, Johannes-Y. Lohrer, Peer Kröger, Christiaan van der Meijden, Eduardo Granado, Jana Lamprecht, Florian Nücke, Henriette Obermaier, Barbara Stopp, Isabelle Baly, Cécile Callou, Lionel Gourichon, Joris Peters, Nadja Pöllath, and Jörg Schiebler. OssoBook v5.6.2. Software, Munich, Germany; Basel, Switzerland, 2018.

[KLK+18b] Daniel Kaltenthaler, Johannes-Y. Lohrer, Peer Kröger, Christiaan van der Meijden, Michaela Harbeck, and Andrea Grigat. AnthroDepot (Dev version). Software, Munich, Germany, 2018.

[KLK+18c] Daniel Kaltenthaler, Johannes-Y. Lohrer, Peer Kröger, Christiaan van der Meijden, and Ciarán Harrington. InBook (Dev version). Software, Munich, Germany, 2018.

[KLK+18d] Daniel Kaltenthaler, Johannes-Y. Lohrer, Peer Kröger, Christiaan van der Meijden, Ciarán Harrington, Erich Claßen, Rupert Gebhard, Sonja Marzinzik, and Heiner Schwarzberg. ArchaeoBook v5.6.2. Software, Munich, Germany, 2018.

[KLK+18e] Daniel Kaltenthaler, Johannes-Y. Lohrer, Peer Kröger, Christiaan van der Meijden, Silke Jantos, Agnes Rahm, Ina Sassen, Tilman Wanke, Roland Wanninger, Jochen Haberstroh, and Sebastian Sommer. ExcaBook v5.6.2. Software, Munich, Germany, 2018.

[KLK+18f] Daniel Kaltenthaler, Johannes-Y. Lohrer, Peer Kröger, Christiaan van der Meijden, and Henriette Obermaier. AnthroDepot v5.6.2. Software, Munich, Germany, 2018.

[KLK+18g] Daniel Kaltenthaler, Johannes-Y. Lohrer, Peer Kröger, Christiaan van der Meijden, Tatiana Sizova, Anja Mösch, Michaela Harbeck, Andrea Grigat, and Anita Toncala. AnthroBook. Software, Munich, Germany, 2018.

[KLKO17] Daniel Kaltenthaler, Johannes-Y. Lohrer, Peer Kröger, and Henriette Obermaier. A Framework for Supporting the Workflow for Archaeo-related Sciences: Managing, Synchronizing and Analyzing Data. In *Datenbanksysteme für Business, Technologie und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs ,,Datenbanken und Informationssysteme" (DBIS), 6.-10. März 2017, Stuttgart, Germany, Workshopband*, pages 89–98, 2017.

[KLP+17] Daniel Kaltenthaler, Johannes-Y. Lohrer, Ptolemaios Paxinos, Daniel Hämmerle, Henriette Obermaier, and Peer Kröger. TaRDIS, a Visual Analytics System for Spatial and Temporal Data in Archaeo-related Disciplines. In *13th IEEE International Conference on e-Science, eScience 2017, Auckland, New Zealand, October 24-27, 2017*, pages 345–353, 2017.

[KLRK17] Daniel Kaltenthaler, Johannes-Y. Lohrer, Florian Richter, and Peer Kröger. ReMIS Cloud: A Distributed Information Management System for Interdisciplinary Knowledge Linkage. In *8th International Conference on Internet Technologies & Society 2017, Sydney, NSW, Australia, 2017*, pages 107–114, 2017.

[KLRK18] Daniel Kaltenthaler, Johannes-Y. Lohrer, Florian Richter, and Peer Kröger. Interdisciplinary Knowledge Cohesion through Distributed Information Management Systems. *Journal of Information, Communication and Ethics in Society*, (to appear) 2018.

[KNI06] KNIME.com. KNIME Analytics Platform. http://www.knime.org/knime, 2006. Online, retrieved 6 May 2016.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[Lam08] Jana Lamprecht. Conception and Implementation of a Intermittently Synchronized Database System for Palaeoanatomic applications. Diploma thesis, Ludwig-Maximilians-Universität München, Munich, Germany, 2008.

[Lev00] Alon Y. Levy. Logic-based techniques in data integration. In *Logic-Based Artificial Intelligence*, pages 575–595. Springer, 2000.

[Lib] The GNU C Library. Sockets. http://www.gnu.org/savannah-checkouts/gnu/libc/manual/html_node/Sockets.html. [Online; accessed 23-January-2018].

[LKK+14] Johannes-Y. Lohrer, Daniel Kaltenthaler, Peer Kröger, Christiaan van der Meijden, and Henriette Obermaier. A Generic Framework for Synchronized Distributed Data Management in Archaeological Related Disciplines. In *10th IEEE International Conference on e-Science, eScience 2014, São Paulo, Brazil, October 20-24, 2014*, pages 5–12, 2014.

[LKK16a] Johannes-Y. Lohrer, Daniel Kaltenthaler, and Peer Kröger. Leveraging Data Analysis for Domain Experts: An Embeddable Framework for Basic Data Science Tasks. In *7th International Conference on Internet Technologies & Society 2016, Melbourne, VIC, Australia, 2016*, pages 51–58, 2016.

[LKK+16b] Johannes-Y. Lohrer, Daniel Kaltenthaler, Peer Kröger, Christiaan van der Meijden, and Henriette Obermaier. A generic framework for synchronized distributed data management in archaeological related disciplines. *Future Generation Computer Systems*, 56:558–570, 2016.

[LKK+17] Johannes-Y. Lohrer, Daniel Kaltenthaler, Peer Kröger, Henriette Obermaier, and Christiaan van der Meijden. Reverse Mediated Information System: Web-based Retrieval of Distributed, Anonymous Information. In *16th International Conference on WWW/Internet 2017, Vilamoura, Portugal, 2017*, pages 63–70, 2017.

[LKR+18] Johannes-Y. Lohrer, Daniel Kaltenthaler, Florian Richter, Tatiana Sizova, Peer Kröger, and Christiaan van der Meijden. Retrieval of Heterogeneous Data from Dynamic and Anonymous Sources. In *8th IEEE International Conference*

Confluence 2018 on Cloud Computing, Data Science and Engineering, Noida, Uttar Pradesh, India, pages 592–597, 2018.

[Loh11] Johannes-Y. Lohrer. Design and Implementation of a Dynamic Database for Archaeozoological Applications. Project thesis, Ludwig-Maximilians-Universität München, Munich, Germany, 2011.

[Loh12] Johannes-Y. Lohrer. Density Based Cluster Analysis of the Archaeological Database OssoBook in Condideration of Aspects of Data Quality. Diploma thesis, Ludwig-Maximilians-Universität München, Munich, Germany, 2012.

[Mat] MathWorks. MATLAB. `https://www.mathworks.com`. Computer Software.

[Mat89] Friedemann Mattern. Virtual Time and Global States of Distributed Systems. *Parallel and Distributed Algorithms*, 1(23):215–226, 1989.

[McA] Colt McAnlis. JSON Compression: Transpose & Binary. `http://mainroach.blogspot.de/2013/08/json-compression-transpose-binary.html`. [Online; accessed 23-January-2018].

[Mic08] Microsoft. Microsoft SQL Server: Replication Features and Tasks. `http://technet.microsoft.com/en-us/library/bb677158.aspx`, 2008. Online, retrieved 21 January 2015.

[MWSM10] Luis Marenco, Rixin Wang, Gordon M. Shepherd, and Perry L. Miller. The nif disco framework: Facilitating automated integration of neuroscience content on the web. *Neuroinformatics*, 8(2):101–112, 2010.

[MyS13] MySQL. MySQL 5.7 Reference Manual: Replication. `http://dev.mysql.com/doc/refman/5.7/en/replication.html`, 2013. Online, retrieved 21 June 2015.

[Neu12] Tanja Neumayer. Design and Implementation of Analysis Methods for Archaeozoological Data. Bachelor thesis, Ludwig-Maximilians-Universität München, Munich, Germany, 2012.

[Nov] Novalys. SQL Anywhere. `http://www.sqlanywhere.info`. Online, retrieved 14 February 2018.

[Ora13a] Oracle. Oracle Documentation Library: Replication Manual. `http://docs.oracle.com/cd/F49540_01/DOC/server.815/a67791/pref.htm`, 2013. Online, retrieved 21 January 2015.

[Ora13b] Oracle. Oracle Streams: Part 1 Oracle Streams Concepts. http://docs.oracle.com/cd/B28359_01/server.111/b28321/pt_concepts.htm#i996787, 2013. Online, retrieved 21 January 2015.

[Ora13c] Oracle. Oracle Streams Recplication Administator's Guide: Understanding oracle Streams Replication. http://docs.oracle.com/cd/B28359_01/server.111/b28322/gen_rep.htm#STREP011, 2013. Online, retrieved 21 January 2015.

[Per] PervaSync. PervaSync. http://www.pervasync.com. Online, retrieved 14 February 2018.

[Rap06] RapidMiner. RapidMiner. http://www.rapidminer.com, 2006.

[Sch98] Jörg Schibler. OSSOBOOK, a database system for archaeozoology. In *Man and the Animal World: Studies in Archaeozoology, Archaeology, Anthropology and Palaeolinguistics in Memoriam Sándor Bökönyi*, pages 491–510, 1998.

[Ser] Imagination Computer Services. Harris Matrix Composer. http://www.harrismatrixcomposer.com. Computer Software.

[SL90] Amit P. Sheth and James A. Larsen. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surverys*, 22(3):183–236, 1990.

[Sof03] Tableau Software. Tableau Public. http://public.tableau.com, 2003.

[Sta14] Bayerische Staatskanzlei. BAYERN DIGITAL II, Investitionsprogramm für die digitale Zukunft Bayerns, 2014. Available from: http://www.bayern.de/wp-content//uploads/2014/09/17-05-30-masterplan-bayern-digital_massnahmen_anlage-mrv_final.pdf.

[Ste46] S. S. Stevens. On the Theory of Scales of Measurement. *Science*, 103(2684):677–680, 1946.

[Swe02] Latanya Sweeney. Achieving k-Anonymity Privacy Protection Using Generalization and Suppression. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):571–588, 2002.

[Tsu10] Yuliya Tsukanava. Development and Appliance of Data Mining Methods on the Palaeoanatomic Data Collection. Diploma thesis, Ludwig-Maximilians-Universität München, Munich, Germany, 2010.

[vdDK14] Angela von den Driesch and Mostefa Kokabi. *Densdrochronological report*. Bayerisches Landesamt für Denkmalpflege, Munich, Germany, 2014.

[WAC+16] Lei Wang, Kathryn I. Alpert, Vince D. Calhoun, Derin J. Cobia, David B. Keator, Margaret D. King, Alexandr Kogan, Drew Landis, Marcelo Tallis, Matthew D. Turner, Steven G. Potkin, Jessica A. Turner, and Jose Luis Ambite. SchizConnect: Mediating Neuroimaging Databases on Schizophrenia and Related Disorders for Large-Scale Integration. *NeuroImage*, 124:1155–1167, 2016.

[Wes] Patrick Wessa. Kernel Density Estimation (v1.0.12). In Free Statistics Software (v1.1.23-r7), `http://www.wessa.net/rwasp_density.wasp`. Office for Research Development and Education.

[Wes14] Timm Weski. *Densdrochronological report*. Bayerisches Landesamt für Denkmalpflege, Munich, Germany, 2014.

[Wie92] Gio Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, 1992.

[Wie94] Gio Wiederhold. Interoperation, Mediation, and Ontologies. In *Proc. Int. Symposium on 5th Generation Computer Systems (FGCS'94), Workshop on Heterogeneous Cooperative Knowledge-Bases. Tokyo, Japan, 1994*, pages 33–48, 1994.

[Wie13] Gio Wiederhold. Mediators, Concepts and Practice. In Tansel Özyer, Keivan Kianmehr, Mehmet Tan, and Jia Zeng, editors, *Information Reuse and Integration In Academia And Industry*, pages 1–27. Springer, Vienna, 2013.

[XFJ] JFJSON. `https://github.com/mainroach/compression/tree/master/xfjson`. [Online; accessed 23-January-2018].

Own Publications

Main Publications

[LKK+14] Johannes-Y. Lohrer, Daniel Kaltenthaler, Peer Kröger, Christiaan van der Meijden, and Henriette Obermaier. A Generic Framework for Synchronized Distributed Data Management in Archaeological Related Disciplines. In *10th IEEE International Conference on e-Science, eScience 2014, São Paulo, Brazil, October 20-24, 2014*, pages 5–12, 2014.

[LKK+16b] Johannes-Y. Lohrer, Daniel Kaltenthaler, Peer Kröger, Christiaan van der Meijden, and Henriette Obermaier. A generic framework for synchronized distributed data management in archaeological related disciplines. *Future Generation Computer Systems*, 56:558–570, 2016.

[LKK16a] Johannes-Y. Lohrer, Daniel Kaltenthaler, and Peer Kröger. Leveraging Data Analysis for Domain Experts: An Embeddable Framework for Basic Data Science Tasks. In *7th International Conference on Internet Technologies & Society 2016, Melbourne, VIC, Australia, 2016*, pages 51–58, 2016.

[KLP+17] Daniel Kaltenthaler, Johannes-Y. Lohrer, Ptolemaios Paxinos, Daniel Hämmerle, Henriette Obermaier, and Peer Kröger. TaRDIS, a Visual Analytics System for Spatial and Temporal Data in Archaeo-related Disciplines. In *13th IEEE International Conference on e-Science, eScience 2017, Auckland, New Zealand, October 24-27, 2017*, pages 345–353, 2017.

[LKK+17] Johannes-Y. Lohrer, Daniel Kaltenthaler, Peer Kröger, Henriette Obermaier, and Christiaan van der Meijden. Reverse Mediated Information System: Web-based Retrieval of Distributed, Anonymous Information. In *16th International Conference on WWW/Internet 2017, Vilamoura, Portugal, 2017*, pages 63–70, 2017.

[LKR+18] Johannes-Y. Lohrer, Daniel Kaltenthaler, Florian Richter, Tatiana Sizova, Peer Kröger, and Christiaan van der Meijden. Retrieval of Heterogeneous Data from Dynamic and Anonymous Sources. In *8th IEEE International Conference Confluence 2018 on Cloud Computing, Data Science and Engineering, Noida, Uttar Pradesh, India*, pages 592–597, 2018.

[KL18] Daniel Kaltenthaler and Johannes-Y. Lohrer. The Historic Development of the Zooarchaeological Database OssoBook and the xBook Framework for Scientific Databases. *ArXiv e-prints: 1801.08052*, January 2018.

Further Publications

[Loh11] Johannes-Y. Lohrer. Design and Implementation of a Dynamic Database for Archaeozoological Applications. Project thesis, Ludwig-Maximilians-Universität München, Munich, Germany, 2011.

[Loh12] Johannes-Y. Lohrer. Density Based Cluster Analysis of the Archaeological Database OssoBook in Condideration of Aspects of Data Quality. Diploma thesis, Ludwig-Maximilians-Universität München, Munich, Germany, 2012.

[KLK+15] Daniel Kaltenthaler, Johannes-Y. Lohrer, Peer Kröger, Christiaan van der Meijden, and Henriette Obermaier. Synchronized Data Management and Its Integration into a Graphical User Interface for Archaeological Related Disciplines. In *Design, User Experience, and Usability: Users and Interactions - 4th International Conference, DUXU 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015, Proceedings, Part II*, pages 317–329, 2015.

[KLKO17] Daniel Kaltenthaler, Johannes-Y. Lohrer, Peer Kröger, and Henriette Obermaier. A Framework for Supporting the Workflow for Archaeo-related Sciences: Managing, Synchronizing and Analyzing Data. In *Datenbanksysteme für Business, Technologie und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs ,,Datenbanken und Informationssysteme" (DBIS), 6.-10. März 2017, Stuttgart, Germany, Workshopband*, pages 89–98, 2017.

[KLK17] Daniel Kaltenthaler, Johannes-Y. Lohrer, and Peer Kröger. Supporting Domain Experts Understanding Their Data: A Visual Framework for Assembling High-Level Analysis Processes. In *11th International Conference on Interfaces and Human Computer Interaction 2017, Lisbon, Portugal, 2017*, pages 217–221, 2017.

[KLRK17] Daniel Kaltenthaler, Johannes-Y. Lohrer, Florian Richter, and Peer Kröger. ReMIS Cloud: A Distributed Information Management System for Interdisciplinary Knowledge Linkage. In *8th International Conference on Internet Technologies & Society 2017, Sydney, NSW, Australia, 2017*, pages 107–114, 2017.

[KLRK18] Daniel Kaltenthaler, Johannes-Y. Lohrer, Florian Richter, and Peer Kröger. Interdisciplinary Knowledge Cohesion through Distributed Information Management Systems. *Journal of Information, Communication and Ethics in Society*, (to appear) 2018.

List of Figures

1.1 A visualization of the workflow of scientific work. 2

2.1 Data collection is usually the first step in any scientific workflow. 8
2.2 The input mask in OssoBook 1.0. [Loh12, Kal12] 10
2.3 The analysis of the age of long bones in OssoBook 1.0. [Loh12, Kal12] . 11
2.4 The input mask in OssoBook 3.4. [Loh12, Kal12] 12
2.5 The input mask in OssoBook 4.1. [Loh12, Kal12] 14
2.6 The input mask in OssoBook 4.1.14. [Loh12, Kal12] 16
2.7 The OssoBook Updater allowed the user to update the OssoBook application. [Loh12, Kal12] 23
2.8 The first xBook Launcher 1.0 with the selection of four different Books based on the xBook framework. 24
2.9 The further developed version xBook Launcher 4.3. 26
2.10 The origin draft for the architecture for the OssoBook synchronization. [KKO+09] 28
2.11 The synchronization panel in OssoBook 5.2.4. [LKK+16b] 29
2.12 The input mask of OssoBook (top) and ArchaeoBook (bottom). Both applications are based on the xBook framework that provides a basic graphical user interface and functions, but allows customization like e.g. individual input fields. [LKK+16b] 31

3.1 Sharing is usually the second step in any scientific workflow. 36
3.2 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 backed-up and shared via the server. 38
3.3 The table “inputunit” of the database of OssoBook [KLK+18a] as an example for the primary keys *ID*, *DatabaseID*, *ProjectID* and *ProjectDatabaseID*. These primary keys are necessary in every data table of xBook. . . . 45
3.4 Three important system columns in the input tables: “Status”, “MessageID” and “Deleted”. The entries with the message ID “–1” are conflicted. 46

3.5 The Code Table "animal" in OssoBook that defines the available values for species. Adding the column "language" allows using terms in different languages: 0 for general terms, 1 for German, 2 for English, etc. 48
3.6 UML visualization of the data types handling the data in xBook. 50
3.7 Simplified visualization of the synchronization, displaying the commit of changed entries to the server and new data from the server. 51
3.8 The synchronization panel in OssoBook 5.2.4. [LKK+16b] 53
3.9 Top: The Conflict Management Screen that displays all conflicted entries of the loaded project. Bottom: The Solve Conflict Screen allows the user to use the local or server values for a specific conflicted entry. 55

4.1 Retrieving data is usually the third step in any scientific workflow. 60
4.2 The basic flow of archaeological data: The data from the excavation is partly passed from the offices to the specialized collections and specialists, who perform individual analyses on the findings and save the results in their databases. The results of these analyses are not accessible from the offices. In sum, neither the offices nor the specialized collections have all information about their findings. 61
4.3 Sketch of the well-known Mediator-based architecture. A central administrator is required to connect the data sources and to mediate the requests from the user. The administrator has to know each data source to be able to connect them. 63
4.4 Sketch of the Reverse-Mediated Information System. The data owners can register their databases to the system on their own. The necessary mediation setup is executed by a wizard dialog. The architecture forwards the user request to the data sources where the request is mediated. 63
4.5 The digital data exchange is hindered. Institutes and specialized collections and specialists each have no digital access to the detailed or individual data of other databases. 65
4.6 The three layers of the architecture: Data Layer, Server Layer, and User Layer. 68
4.7 Sequence diagram of the initialization process. 69
4.8 Sequence diagram of the registration process. 70
4.9 Sequence diagram of the process to fetch data from the single data sources. 72
4.10 Screenshot of the wizard dialog where administrators can map the specified parameters of the Minimal Search Parameter to the actual data of their database. 76

4.11 Screenshot of the wizard dialog where administrators can determine the columns from the connected data source which should be communicated and transferred to the user. 77
4.12 Screenshot of the wizard dialog where administrators can define foreign keys to define related columns in separated tables, e.g. for the use of IDs and value tables. 78
4.13 Screenshot of the search mask of REMIS. 82
4.14 Screenshot of the (shortened) retrieved result of REMIS. 83
4.15 An abstract sketch of the REMIS CLOUD architecture. 85
4.16 A simplified sketch of the REMIS CLOUD architecture for archeo-related sciences. 88
4.17 A simplified sketch of the REMIS CLOUD architecture for the eLearning example. 89
4.18 Screenshot of the REMIS CLOUD Prototype displaying the result described in Chapter 4.7.4 89

5.1 Analyzing data is usually the final step in any scientific workflow. 94
5.2 Schematic representation of a Property. 99
5.3 Schematic representation of the IController. 103
5.4 Schematic representation of the Worker. 104
5.5 From left to right, the screenshots of the Retriever, Combiner, Filter, and Sorter, as they are represented in the graphical user interface of the tool. . 107
5.6 Examples for different Diagrams 108
5.7 Composition of the analysis for animal distribution in Munich, described in the first example of Chapter 5.1.7. The analysis framework is embedded to the zooarchaeological database OSSOBOOK [KLK+18a]. 109
5.8 Possible result of the first analysis. 111
5.9 Composition of the analysis for the most common animal, dependent on the average temperature, described in the second example of Chapter 5.1.7. 112
5.10 Result of the second analysis. 113
5.11 “The Harris Matrix system recognizes only three relationships between units of archaeological stratification: (A) The units have no direct stratigraphic connection. (B) they are in superposition; and (C) the units are correlated as parts of an once-whole deposit or [layer] interface.” [Har89] 118

5.12 Screenshot of the TARDIS application. The 2D representation (top center) shows the areas and sections and displays heat map colors for the absolute number of findings and the result of a Kernel Density Estimation in the background. The 3D representation (bottom center) shows the distribution of findings in the layers. The generated Harris Matrix is displayed on the right. Settings and filter options can be entered on the left. 120
5.13 Illustration of the algorithms described in Chapter 5.2.3.3 to create a Harris Matrix. 124
5.14 Planum (left) and profile (right) drawing of shaft 5 of the excavation Marienhof-Haltepunkt in Munich, Germany. Drawing: ReVe, Büro für Archäologie, Bamberg, Germany. The layers 360, 745, 746, 778, 801, and 997, that are mentioned in Chapter 5.2.4, are highlighted. The mentioned layer 393 is not visible in the drawing. 128
5.15 The 3D representation and Harris Matrix of data from shaft 5 of the excavation Marienhof-Haltepunkt in Munich, Germany. The shown Harris Matrix (A) is the temporal structure of the drawing sheet in Figure 5.14 with the unfiltered distribution of findings. The other illustrations show the 3D representation of the shaft and the distribution of filtered findings of (B) frogs, (C) cattle, and (D) dogs/cats. 129

6.1 A visualization of the workflow of scientific work as described in this thesis. 134

List of Tables

4.1 Extract of the result for the example query “Luther” in the Category “Historic Events” 88

5.1 Animals 110
5.2 Average Temperatures 111

List of Algorithms

1 The (shortened) example structure of the *book.xml* file. 25

2 Trigger to update the status column to the current time in entry tables. . . . 45
3 Trigger that updates the status column to the current time for Code Tables. 47
4 Trigger that also updates the status column in the version table. 47

5 Checking connectedness and loops 125
6 Check for redundant links 126
7 Sort for height of nodes 126
8 Sort nodes to minimize crossings 126