Authenticus: Architecture and Mechanisms to Support a National Repository of Scientific Publications

Fábio André Domingues
Mestrado em Ciência de Computadores
Departamendo de Ciência de Computadores
2015

Orientador
Doutor Fernando M. A. Silva, Professor Catedrático, FCUP

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

Fábio André Domingues

Authenticus: Architecture and Mechanisms to Support a National Repository of Scientific Publications

Dissertação submetida à Faculdade de Ciências da Universidade do Porto para obtenção do grau de Mestre em Ciência de Computadores

Orientador: Professor Dr. Fernando M. A. Silva

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
2015

Abstract

Research evaluation is a complex task and the use of advanced bibliometric indicators to compare within and across scientific disciplines can be an helpful instrument in assisting evaluators. However, the quality of the indicators is rather crucial, specially when we are dealing with the evaluation of individual researchers. Having a curated repository of standard scientific publications metadata is fundamental to enable credible and reliable bibliometric studies.

Authenticus is a project developed at the University of Porto and CRACS/INESC TEC, co-funded by FCT, that aims to build a national repository of publications metadata authored by researchers of Portuguese instituitions. The system automatically uploads publications from multiple indexing databases, automatically associates publication authors with known researchers and institutions, provides specialized interfaces to researchers and instituions to confirm or dismiss proposed associations, allows interoperability with other CRIS systems, provides synchronization with ORCID, both for import and export, among many other functionalities.

This dissertation describes the architecture of Authenticus, focusing in more detail on the contributions made by the author. These included the design and implementation of an *Action Center* and corresponding framework. The action center is a core component as it controls and monitors the execution of actions, logs events and errors for future recovery, and monitors data transactions. Other functionalities developed included a query builder to simplify development and to increase system's security, tools to automatically import publications from multiple indexing sources, duplicates detection, a task scheduler, interoperability services with other CRIS systems, and synchronizes bi-directionally with ORCID.

Resumo

A avaliação do desempenho de investigação é uma tarefa complexa, sendo o uso de indicadores bibliométricos especializados que possibilitem uma comparação da produção numa ou entre áreas disciplinares, uma ajuda e um instrumento importante para os avaliadores. Contudo, a qualidade dos indicadores é crucial, em especial quando lidamos com a avaliação de investigadores individuais. Possuir uma base de dados com os meta-dados de publicações científicas validadas constitui por isso um passo fundamental para se poderem realizar estudos bibliométricos mais credíveis e confiáveis.

O Authenticus resulta de um projeto desenvolvido na Universidade do Porto e CRAC-S/INESC TEC, co-financiado pela FCT, com o objetivo de construir um repositório nacional de meta-dados de publicações científicas com autoria de investigadores de instituições portuguesas. O sistema é alimentado automaticamente a partir de múltiplas bases de dados de indexação, associa automaticamente autores de publicações a investigadores e publicações a instituições, fornece interfaces especializadas a investigadores e instituições para confirmarem ou anularem associações, possibilita interoperabilidade de dados com outros sistemas CRIS, e possibilita a sincronização bidirecional com o sistema ORCID.

Esta dissertação descreve a arquitetura do Authenticus, focando-se em mais detalhes nas componentes que constituem a contribuição do autor. Estas incluem o desenho e implementação do *action-center* e correspondente *framework* de desenvolvimento. O *action-center* é um componente nuclear no sistema responsável por controlar e monitorizar a execução de ações, efetuar *logging* de eventos e erros para recuperação futura, e monitorizar a execução de transações. Outras funcionalidades desenvolvidas incluem um *query builder* para simplificar o desenvolvimento e aumentar segurança, utilitários para carregar automaticamente publicações de múltiplas bases de indexação, deteção de duplicados, um gestor de tarefas, serviços de interoperabilidade com outros sistemas CRIS e sincronização com o ORCID.

Aos meus pais.

Acknowledgements

R&D for a junior student is always a challenge and I am very grateful to Professor Fernando Silva in believing in me when provided a research grant at an early stage, when I was still as bachelor student. It allowed me to start contributing to the development of the Authenticus. That lead to the work described in this dissertation and I am very thankful for his advice, guidance, patience, and support throughout.

I would also like to specially thank Sylwia Bugla, my co-developer in Authenticus project, for her friendship, kind cooperation, support, help with the many figures in this dissertation, and also its revision. It is a pleasure to work with her.

I would also like to thank and mention the support of Authenticus by the University of Porto and of FCT/FCCN.

I am also thankful to the colleagues that share room 1.71 for the good company and for sharing good experiences.

Special thanks are also due to all my friends in Porto and from RUCA (and to Mercedes chat) that have been part of my family over the last years, without whom this endeavour would not have been the same. Thanks to Abreu, Andreia, Bilhó, Carolina, Catarina, Chicote, Daniela, Diogo, Hélder, Hélio, Inês, Joana, Leandro, Lost, Marcelo, Miguel, Otto, Pangaio, Queiroz, Renata, Ricardo, Romeu, Rui, Susana, Tiago, Vasco and Zé, for helping in keeping my level of sanity closest to zero! I hope I am not forgettin anyone.

Last but certainly not the least, I would like to thank my sister and my parents for their constant support and love, and for always being there for me.

Index

List of Tables

List of Figures

Chapter 1

Introduction

Policy makers and funding government agencies are naturally interested in promoting more systematic and less costly ways to evaluate research performance of institutions, groups or individuals. Although research evaluation is a complex issue, advanced bibliometric indicators to compare within and across scientific disciplines are being introduced. Many countries are collecting scientific publication data from their Universities, thus building national bibliographic databases, and use them to steer research management. Thus, creating a normalized and validated bibliographic database of scientific publications is a fundamental step to enable more credible and reliable bibliometric studies being conducted. Whatever sources are used, the problem of validation of the data is far from solved. Dealing with the heterogeneity, non-uniformity and lack of consistency of the raw data resources that feed these databases is a major issue and still object of much research and development worldwide.

Authenticus is a software platform that automatically associates publication authors to known researchers and corresponding publication records to Portuguese institutions, aiming at building a national repository of scientific publications. Authenticus has been designed and implemented to provide researchers and institutions with a set of functionalities and tools to fulfil such goal, namely it provides:

- a multicriteria identification algorithm to automatically associate publication authors to known researchers and publications to Portuguese institutions;
- specialised interfaces for researchers and institutions to access data and indicators, and allow them to confirm their publications, or dismiss wrong associations in a simple but effective way;

- automatic upload capabilities of publications indexed by well known comercial sources for which the institutions hold a license, such as ISI-Web-of-Knowledge from Thomson-Reuters, Scopus from Elsevier, but also from non-comercial sources such as DBLP (specialised in Computer Science) and Scholar from Google;

- automatic detection of duplicated publications metadata originated from multiple sources, and hability to merge duplicates into just one Authenticus record while keeping the original identifiers;

- dynamic updates of citation information from multiple sources;

- an integrated view of the journals and venues where the publications occurs.

- multiple export formats so that researchers or institutions can synchronise data with other systems;

- synchronise with ORCID, both for Import and for Export of the researchers' list of publications;

- initial bibliometric indicators about researchers and institutions.

The development of Authenticus started in 2010 with the development of the identification algorithm [1]. It initially was planned just for the University of Porto, thus for a smaller scale, and just with one main developer. Its interface was Web based and developed using the CakePHP framework [3]. In 2013, the project lead by Professor Fernando Silva from FCUP and CRACS/INESC TEC, received the support from FCT and its scope became national. While on one hand this was an opportunity, it brought more responsibility to the leading team, since the requirements for both efficiency and scalability became more demanding, as one needed to be prepared to efficiently support hundreds or even thousands of users simultaneously. The development team increased, the design of Authenticus was completely revised, and it was re-implemented with a new design architecture.

The work of this dissertation focus precisely on the new architecture of Authenticus, namely on the contributions made by the author to the project as outlined in the contributions below.

1.1 About Authenticus

Authenticus is a Web application already publicly available to reasearchers and institutions at address `www.authenticus.pt`. Authenticus uses a role based access to control the display of interfaces and access to functionalities as will be detailed in the next chapters. Non-authenticated users can query about publications, researchers and institutions, however the information that is displayed is limited when compared to users that are authenticated in Authenticus and are associated with an institution with b-On access. Figure 1.1 illustrates a view that any user on the Web will see when he searches for a specific researcher in Authenticus and sees his list of validated publications. The title links shown in each publication lead only to minimal public information about the publications.

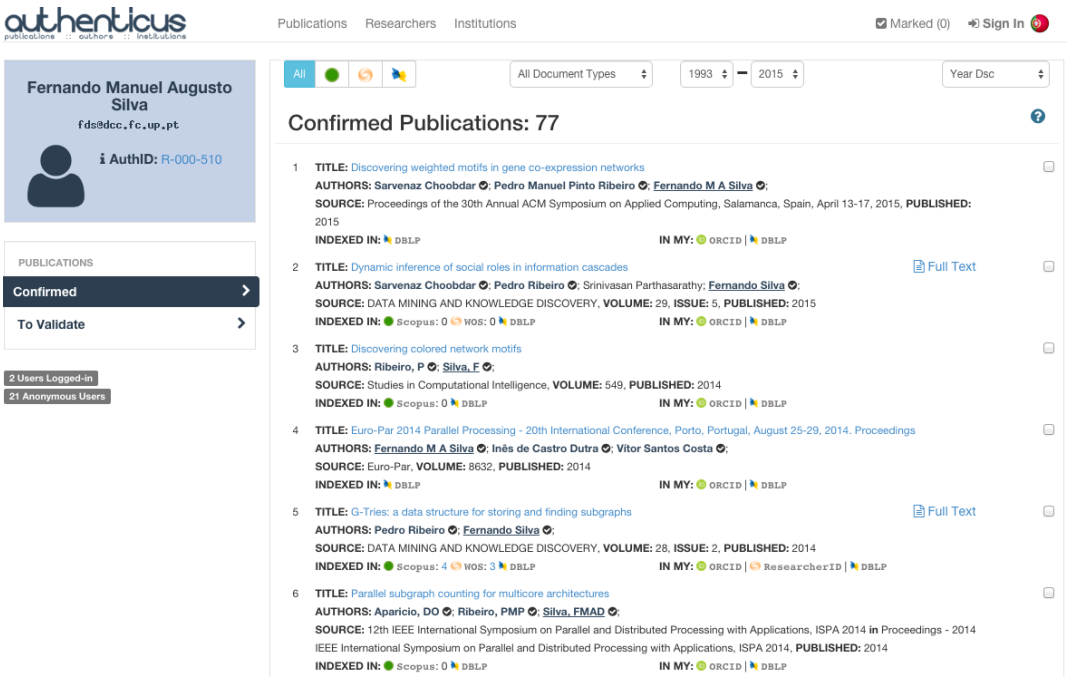


Figure 1.1: Public view of a validated list of publications.

Currently, Authenticus already has more than 1750 users that haved logged in at least once in the system. Users may authenticate in Authenticus using the federated authentication used in their institution, if it complies with RCTSaai, or else request an account with authenticus, using an institutional email, and login with that account. For users that are connected with institutions that are under the umbrella of b-On licensing, they have a broader and more detailed information view.

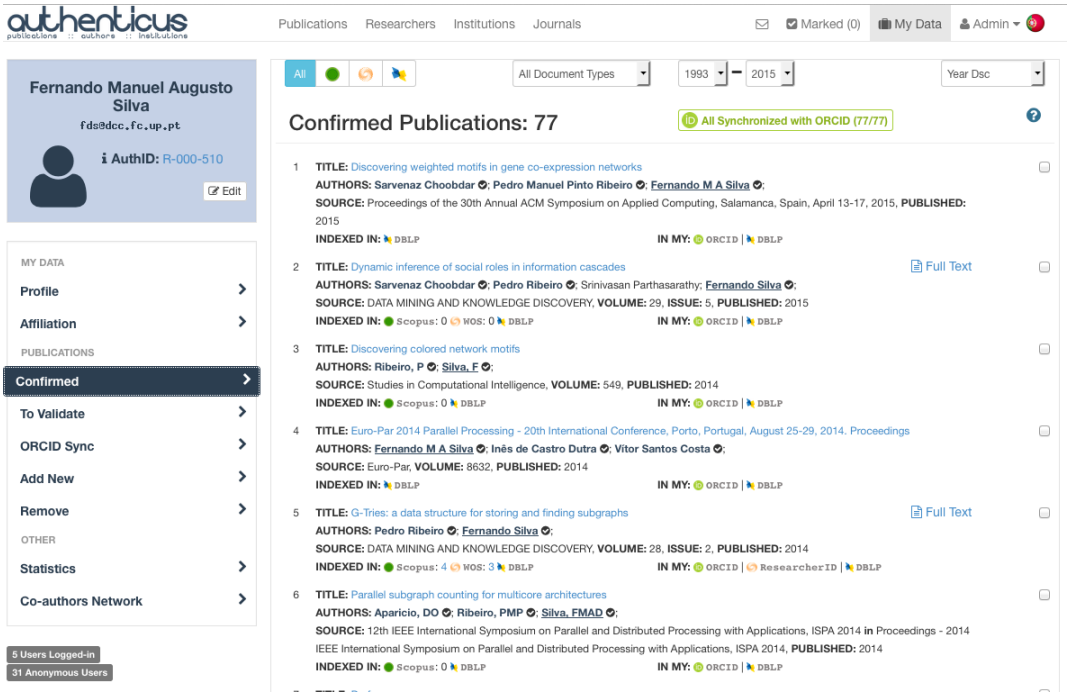


Figure 1.2: Authenticated view of a validated list of publications.

Figure 1.2 illustrates, for the same researcher shown in Figure 1.1, the same page as before. The title links in each publication now lead to further details about the publications as can be seen in Figure 1.3.

Yet another facet of Authenticus, not usually seen by normal users, is the dashboard interface that is only available to users with an administration role and is used to handle many of the backoffice operations that are necessary to plan or execute as it will be made clearer in the chapters that follow. Figure 1.4 illustrates the dashboard and many of the entries that it contains to backend functionalities.

In Chapter 2 we give a more detailed overview of Authenticus, including a description of its architecture, databases, and framework tools used in its development.

1.2 Contributions

Although Authenticus has had a few contributors, with a colleague and I being the main contributors, I briefly outline the functionalities that I have developed, or for which I was the main contributor:

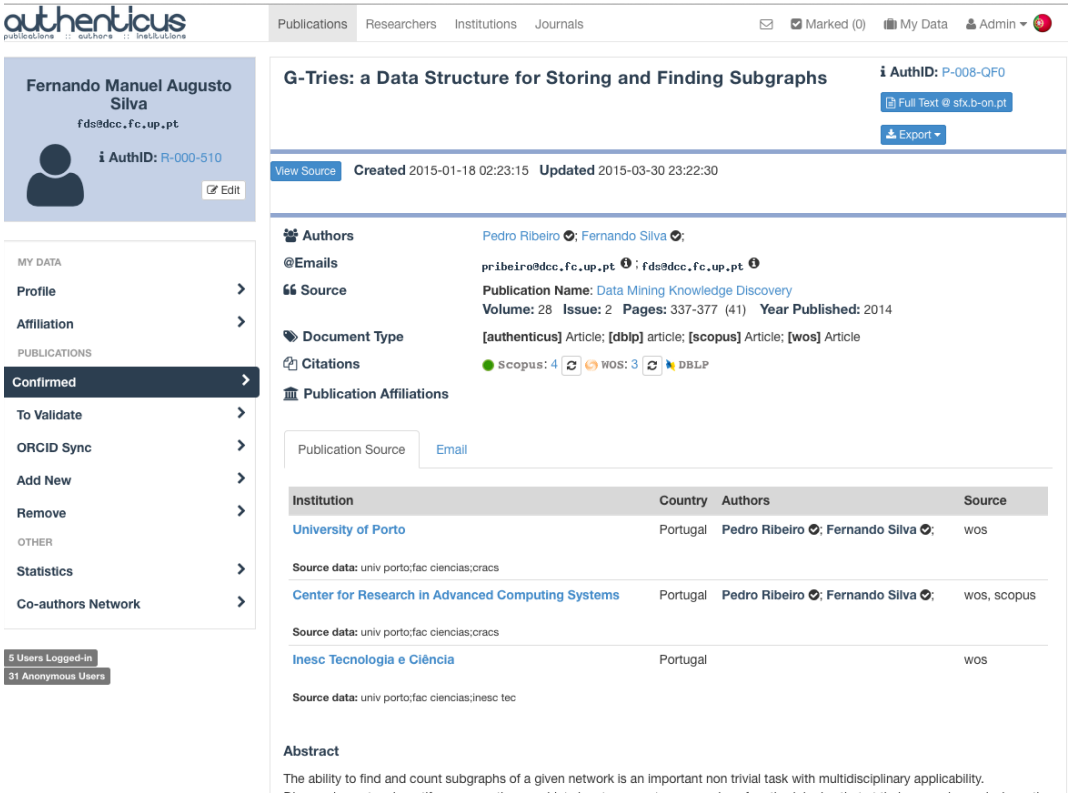


Figure 1.3: A detailed view of a given publication.

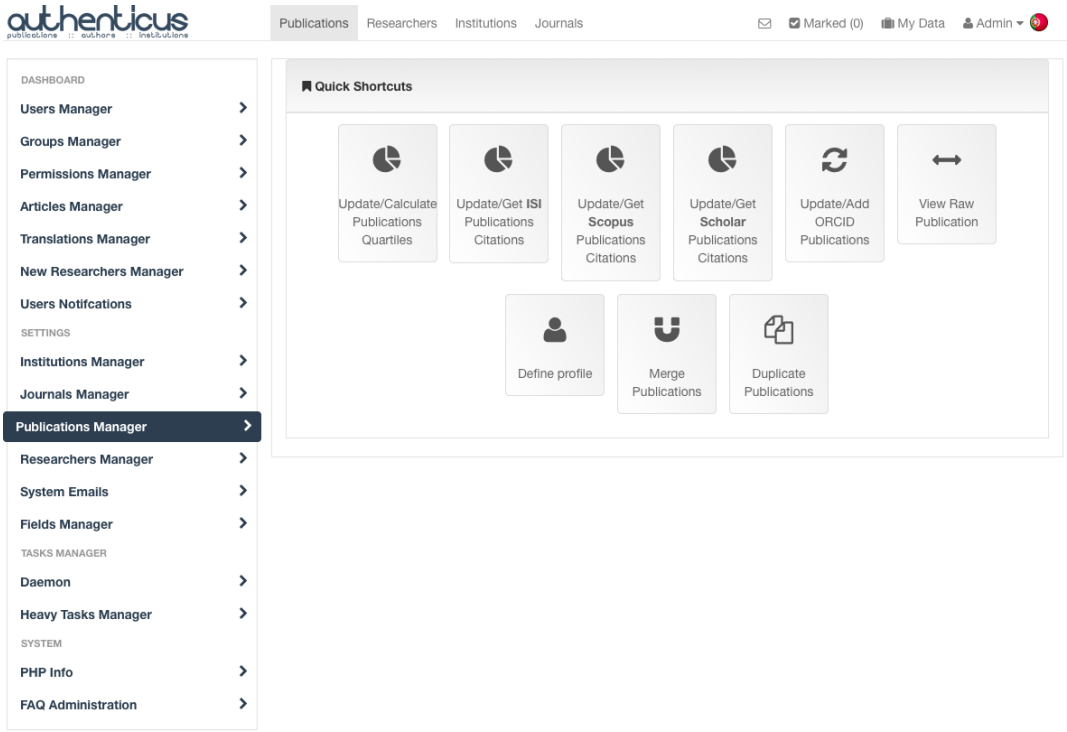


Figure 1.4: A view of the dashboard accessible to system administrators.

- The action-center is a core component of Authenticus that abstracts database access for Authenticus actions, controls and monitors actions execution, ensures logging of events and errors for future recovery purposes, and transparently ensures the consistent use of database transactions.

- A flexible SQL query builder was developed to simplify query construction, to prevent SQL injections by using prepared statements, and allows the use of caching on specific recurrent queries (Memcached).

- Tools to automatically import publications from multiple sources were developed so that Authenticus is preemptively updated without the researchers having to interfere. Currently, Authenticus uploads every week publications from ISI Web-of-Knowledge, Scopus, DBLP and ORCID.

- A tool to detect publication duplicates. When dealing with publications metadata originated from multiple sources, it is natural to expect that the same publication may be indexed by more than one source. We try to identify such cases and merge those publications in a new Authenticus publication record that is a maximal representation of all the data of the various sources. Considering that multiple

sources represent the same information differently, or may have errors in their data, or simply cases in which some fields are the same, but the evidence is not enough to make an automatic merge decision, the researchers are invited to confirm the merge or to dismiss it.

- Authenticus is currently able to synchronise, in both directions, all publications of a researchers with ORCID. This is a tremendous benefit for researchers considering that the interface provided by Authenticus is much simpler and complete because it handles all indexing sources at once.

- Authenticus provides a Publications Retrieval Web Service that enables external allowed entities to retrieve the publications of a given set of researchers.

- A task scheduler was developed to handle heavy tasks that can be broken down into smaller ones for better execution monitoring and control, as well as to handle recurrent tasks, e.g. weekly updates from ISI or Scopus. A daemon was implemented and is awaken whenever there are tasks to be executed. Tasks may have priorities assigned to defined urgency and order of execution, and may be placed in one of multiple queues to allow parallel execution whenever possible.

- A backoffice was developed for Authenticus on which multiple control and monitoring functionalities have been included. It is only accessible to the administrator user, not to the common users.

1.3 Outline of the Dissertation

In this chapter we have introduced Authenticus, its main goals and functionalities, and highlighted the contributions of this work. In Chapter 2 we will overview the architecture of Authenticus and its interface, and discuss other systems that exist with similar aggregation goals.

Chapter 3 describes in detail the action center with its multiple facets and functionalities, namely the query builder, processing of loggs and errors, and handling of transactions. The task scheduler is also described as it basically is the executor of the actions run by the action center.

Chapter 4 focus on the management of publications in Authenticus, namely on the upload of publications metadata from multiple sources, on detecting duplicates and

merging publications, ORCID synchronisations, and on interoperability services supported in Authenticus.

Chapter 5 summarises some conclusions of the work done and leaves some thoughts on future work that could be included in Authenticus.

Chapter 2

Authenticus: a System Overview

This chapter provides a high-level overview of the Authenticus system, focusing on its logical system architecture to simplify the understanding of the following chapters and the contributions of this work. We start by giving a general context and present its logical architecture, present the structure of the system through its components and their interactions. We also describe the technologies used in Authenticus and possible constraints. Then, we overview the Authenticus database, by presenting the main entities and relationships involved. We also discuss and justify the reasons to choose a specific database system and the benefits it brings. Then we describe in general terms the Authenticus Author Identification Algorithm, an important component of Authenticus that contributes to the automatic assignment of publications to researchers and institutions. We conclude this chapter with an overview of the features and functionalities of the main "business" logic modules, specially those dealing with the "Researchers Module" or 'User/Group Module".

2.1 Basic System Architecture

Authenticus is a software system that operates as a standard client-server web application and as a web service to support machine-to-machine interaction over a network. It is designed to become a certified database of scientific publications authored by researchers from Portuguese institutions. The system is accessible through different profile views, one that is public and others that require authentication, each of them having different types of role base access permissions to data and functionalities. General public has access to basic information and only very few features of the system. The information

that an anonymous user can find in Authenticus is, in principle, also accessible in other public places over the Web, or its public view has been granted permission. Authenticated users may be of three types of entities: institutional user, individual user (researcher or academic staff), or admin (reserved for developers only). An institutional user has additional permissions compared to an individual user as he can manage data about his institution including creating new researchers associated with that institution and validating publications of researchers of his institution. Permissions of the authenticated users are additionally controlled with the special b-On license, where users belonging to institutions with the license have access to extended functionalities and data of Authenticus. Section 2.1.3 gives a better overview of b-On license and of Authenticus in b-On environment.

2.1.1 Logical System Architecture

The logical view shows the architecturally significant components of the application and the major relationships and dependencies between them. Figure 2.1 shows a diagram illustrating a three layered architecture for Authenticus, a client layer, a server layer and a data layer.

Client layer. Authenticus is a multi-client application in which clients can communicate in different ways with the system. It can be directly accessed through a browser, using the Authenticus interface (local clients), or through the invocation of web services made by external systems (external clients). Clients communicate directly with the server, namely with the controller, which in the MVC architecture is responsible for responding to user input, validating it and performing business operations. The communication of the local clients with the application is controlled with session attributes, which allow to identify unique users across the requests and store persistent data for specific users. External clients treat Authenticus as an "application server" and make use of its functionality by communicating with the business layer through a web service interface. The communication between client and server in this scenario is stateless, meaning that each request has an independent transaction that is unrelated to any previous request so that the communication consists of independent pairs of request and response. In order for Authenticus external clients to use the Authenticus web service it requires authentication embedded in the request parameters. Further details on the Authenticus Publications Retrieval Web Service are given in Section 4.4.3.

Server layer. Authenticus server layer components implement the core functionality of

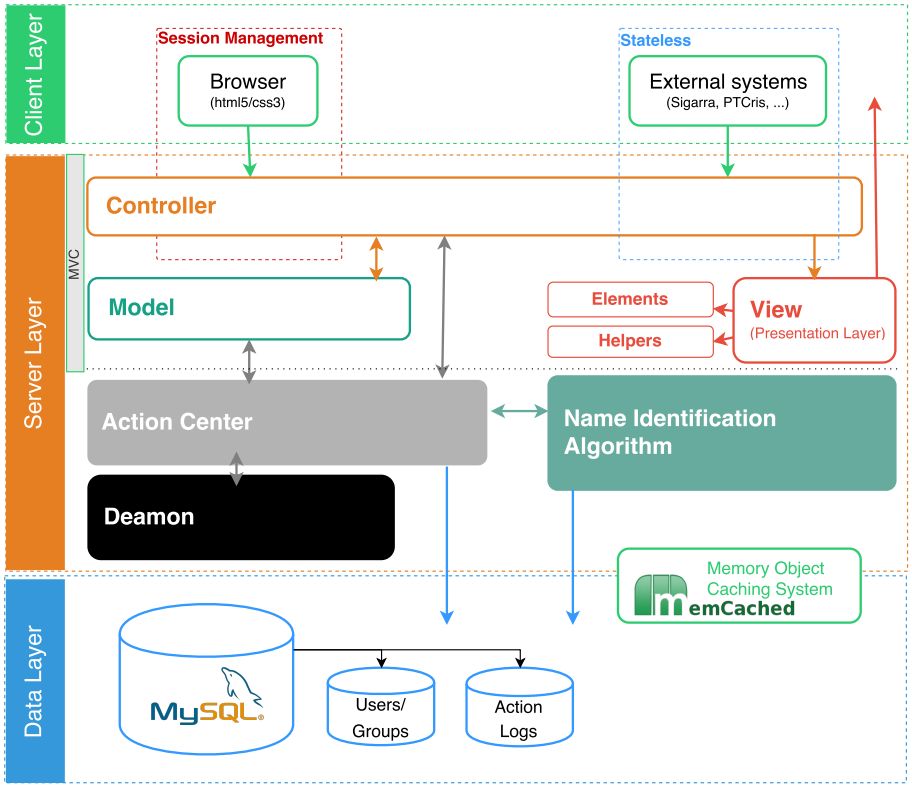


Figure 2.1: Authenticus Logical System Architecture.

the system, and encapsulate the relevant business logic. The main components are kept in the model-view-controller architecture, where the model captures the behavior of the application and directly manages the data, logic and rules of the application. A view is any output data/information representation presented to the users. The controller accepts input from users, and based on it decides which operations will be performed on the models and propagates resulting information to the view. This base structure of Authenticus was built with a free, open-source and rapid development framework, CakePHP[1]. On the latest phases of application development, Authenticus was extended with other components to support complex features and behaviors of the system. The most relevant components are: the Action Center (Chapter 3) to control and monitor actions execution of Authenticus, Daemon (Section 3.3) responsible for prioritizing an execution and scheduling of some complex actions, and Name Identification Algorithm (Section 2.3). The following sections give a better overview of the core components of Authenticus.

Data layer. Authenticus data layer components provide access to the database that is hosted within the boundaries of the system. In the application there are two components

[1]CakePHP - http://cakephp.org

which communicate directly with the data layer: Action Center over MySQL PHP connector and Name Identification Algorithm over JDBC connector. Between data and server layer there is a distributed memory object caching system to speed up the system by alleviating database loads. An overview of the database is given in Section 2.2, where we present the main tables.

2.1.2 Authenticus Technological Details

Authenticus resorts to a number of technological resources, both at the physical infrastructure (support hardware) as well as the software needed to deploy the system. Table 2.1 details the various setups used, namely the hardware configuration that supports Authenticus server, the software setup for Authenticus, the database server used and the Web interface software used.

2.1.3 Architectural Constraints

In order for Authenticus to operate in an academic environment and deal with external publications metadata, it has to face several constraints. Those constraints affect our architecture decisions as they impose limitations on to whom we can expose our data and how we can do it. The major constraints of Authenticus concerns publications metadata, which is usually copyrighted by the owners/authors of the publication but also by publishing companies and other commercial bibliographic database. This problem could be overcome with a b-On (Online Knowledge Library) license, which "makes unlimited and permanent access available, within the research and higher education institutions, to full texts from over 16,750 scientific international publications from 16 publishers, through subscriptions negotiated on a national basis with these publishers."[2] In Authenticus, users with known affiliation that are part of the b-On community, have granted an extended access to Authenticus resources, which allows them to view and manipulate various metadata not visible for public and non b-On users.

[2]B-On: www.b-on.pt/en/what-is-b-on

Hardware Setup	
Processor	AMD Opteron(tm) Processor 6344 - 12 cores, 2.6 Ghz
HDD	2TB - setting in RAID 1 and controller in hardware with 1GB cache
RAM	128GB DDR3 1600Mhz ECC
Software Setup	
Operating System	Linux Ubuntu server 14.04.3 LTS
PHP	5.5.9-1ubuntu4.11
Python	2.7.6
Web Server	Apache/2.4.7
Shibboleth	shibboleth 2.5.2
Caching System	memcached 1.4.14
Version Control	SVN
Java	1.8.0_60
Other libraries	php5-mysql, php5-curl, php5-memcached, python-mechanize, php5-xsl, texlive, gnumeric
Database Server Setup	
Version	mysql Ver 14.14 Distrib 5.6.19, for debian-linux-gnu (x86_64) using EditLine wrapper
Data Size	28GB
Web Inteface software	
Framework	CakePHP 2.6
Template	Bootstrap 3
Javascript Libraries	jQuery v@1.10, jQuery UI, Highmaps 1.0.4, Font Awesome and others.

Table 2.1: Authenticus technical details.

2.2 Database

Authenticus database is a relational database, running on a MySQL [10] server, a popular open source relational database management system (RDBMS). To handle the administration of the database we use a tool phpMyAdmin[3] that provides a traditional web interface. There are several reasons why we decided to use MySQL database:

- It is a free, open source and considered a proven technology in the industry.
- It is scalable, we can extend it as the application usability grows.
- It is a well-established relational DBMS with support for foreign keys. Foreign keys have constraints assigned that define how the DBMS should perform when the link between two tables change. This is very useful in Authenticus due to the high number of tables and relations between them. For example, when a publication is removed from the table, MySQL automatically removes all associated information as authors, keywords, institutions, etc. It is much simpler and efficient to just set the foreign key constraints and let the database do the job.
- It supports indexing, which improves performance on "SELECT" operations and supports full-text indexing and searching, essential for fuzzy searches. In Authenticus we need full-text indexing to perform fast and complex searches in text fields, as publication title. It is also very useful for approximate searches, where using "MATCH AGAINST" function we can specify in a "WHERE" clause a search term, and the function returns rows automatically sorted with highest relevance first. In Authenticus we use this function in searching for a publication by title term.
- It features ACID [7] support, distributed transactions and server enforced referential integrity checks on transactions.
- It can also reach top speed and provide stable and consistent performance for the size Authenticus database and complexity, which currently accounts to 150 tables and more than 85 million records.
- It supports all Unicode 5.0 and 6.0 characters. Unicode is a standard for consistent encoding of characters of the different languages. In Authenticus publications are written in many language and authors come from every part of the world.

[3] phpMyAdmin - www.phpmyadmin.net

The fact that MySQL supports Unicode characters, allows to store in the same database column text with different languages and alphabets without doing any transformation. Another advantage of using Unicode is that we can do a "non-binary" search, for example in German *ß* is equal to *ss* and MySQL respects that, thus searching for *ß* returns all records with *ss* and vice versa.

Authenticus database has over 150 tables, more than 85 million records and it size exceeds 28GB. The data it contains could be grouped into several clusters, such as data about publications, researchers or data about institutions. There is also a great set of auxiliary data which is static, thus there are no methods nor constraints that modifies it. An example of such a data could be: set of countries, set of scientific fields or a list and mapping of publication document types.

Other DBMS systems exist that could also be used to support Authenticus, namely PostgreSQL[4], an open source object-relational database system, and Oracle, a popular comercial database system. However, for this dissertation, the choice of DBMS was not an option.

Figure 2.2 shows a diagram that illustrates three main entities, namely Publications, Researchers and Institutions that are the main information objects that are represented and characterized in Authenticus. For each of these entities the figure shows a partial set of tables that are needed to fully characterize the main entity and the relationships existing among the entities. A fourth entity is represented to group tables that represent auxiliary information.

2.3 Name Identification Algorithm

Authenticus aims to build a certified database of scientific publications authored by researchers from Portuguese institutions. Thus, it requires the ability to identify an author name in a publication and correctly associate it with a researcher in the database. Author identification in publications is full of uncertainty, author names are not unique, variations of names may be caused by abbreviations, permutations, accents, hyphens, typos, cultural naming conventions, etc, thus making it difficult to identify homonym author names. Author identification is a special case of identity uncertainty resulting from the fact that objects may not be labeled with unique identifiers.

[4]PostgreSQL 9.5 (latest version) is available at `http://www.postgresql.org`

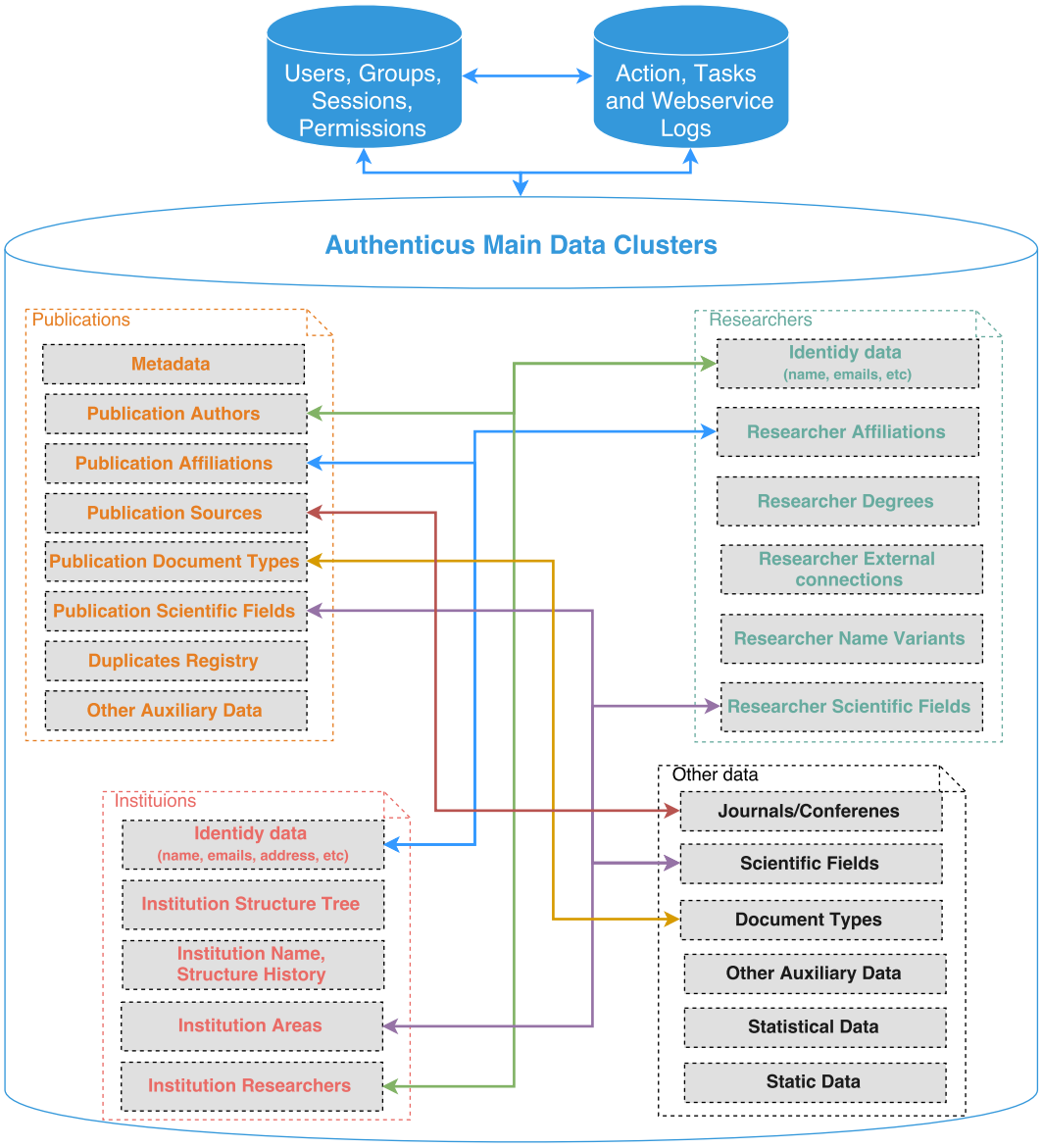


Figure 2.2: Authenticus main data clusters.

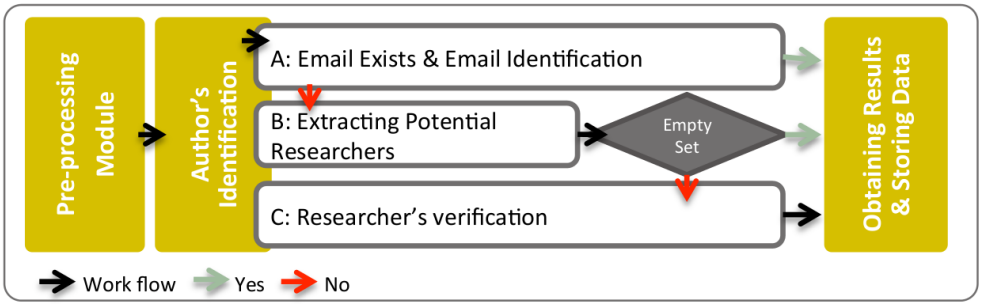


Figure 2.3: The author identification algorithm flow of execution.

Authenticus approach for author identification is optimized for contextually reach datasets and was proposed in [1]. From each publication record, the algorithm uses author names, emails, institutional addresses, journals, subject categories, and keywords. In our algorithm, authors can be fully identified by their email or by their names. Name identification relies on string matching methods and on attribute verification methods to filter and validate the set of successful matches between the name of the publication author and full names of researchers. The goal is to reduce this set of matches to just one match and the verification rules based on name matching rules, institutional addresses, publication journals, subject areas, publication name and co-author analysis are decisive to correctly and uniquely associate an author with a researcher name. If the validation is not sufficient to fully identify the author, partial information on the identification is kept for future runs.

The process of identifying an author in a publication record is structured in three main modules that are executed in succession: data acquisition and pre-processing, author identification, results analysis and data storage. Figure 2.3 illustrates the main flow of the algorithm.

The data acquisition and pre-processing module processes, parses and standardizes some fields of the publication meta data and gathers existing contextual information about the record that is required in the next steps for author identification.

The main module of the algorithm, author identification, analyses each author of the publication to determine its correct identity. In the first step, the algorithm tries to establish a correspondence between authors and emails in the publication record. To connect emails and authors it uses the information about the researchers stored in the database. A successful match, fully identifies an author given the uniqueness of email addresses.

If email identification is not successful, the algorithm resorts to identification based on

the analysis of author names and other attributes of the publication. We first run a name matching procedure that uses exact (direct `LIKE` SQL queries) and approximate string matching algorithms (accomplished with Java text search engine library Apache Lucene[5]) to produce an initial set of potential researchers that are candidates to become the author being analyzed. This set is first simplified by filtering out names that do not obey our name construction rules. If the set of researchers becomes empty, it means that the author being analyzed is not known in the researchers database and, thus, is declared as a new researcher and no further verification is needed. If the set of candidates is not empty, then the algorithm proceeds with the identification by applying a number of verification procedures and calculating a similarity value between each candidate and the author being analyzed. The similarity value is calculated by weighing the scores obtained by a candidate on each attribute being verified. The weights used for each attribute depend on their relevance for disambiguation. For example, co-authorships are weighed more then publication journal names. A candidate is kept in the set only if its similarity score is above a pre-defined minimum threshold. In the end, if the set of potential candidates is reduced to just one candidate, or if it has more then one candidate, but only one has a similarity score above a fully identified threshold, then that candidate is assigned as the author of the publication. The logic behind the verification procedures is to assert a relationship between candidates and one author based on past evidence that relate their publication attributes.

The last module of the algorithm analyses the results and saves all relevant data gathered about the identified authors of the analyzed publication. It includes newly discovered emails of authors, institutional addresses of authors, new co-author relations, journal and subject category of the current publication if any. The newly stored data is a very important resource as it provides information that can help to identify authors in other publications, thus its correctness is a very important issue.

2.4 Authenticus Business Logic Modules

Authenticus includes many modules in its business logic layer. Every module has a model–view–controller (MVC) architecture, meaning that there are three interconnected parts: model, view and controller. The model part deals with the application data and communicates with database. It is the "heart" of each of the modules, where the whole module business logic is defined. In Authenticus, this layer was extended

[5]Apache Lucene - https://lucene.apache.org/core/

with an Action Center (work done within the scope of this dissertation and detailed in Chapter 3), which, among others, defines rules and transactions necessary to run some database procedures. The controller part determines behaviors of the processes. The view part generates an output presentation to the final user of the application. The whole Authenticus user web interface consists of different views of different modules. Figure 2.4 shows the most important modules of Authenticus. Some of the modules are interconnected, they complement each other or extend complex functionalities. There are other independent or auxiliary modules to support complex Authenticus features.

To manage features and functionalities of all Authenticus modules we developed an application administration layer. It constitutes the Authenticus back-office, only accessible to a few users, where it is possible to control processes, execute special actions and where are located all the tools needed to keep Authenticus running. In Authenticus, the back-office administrator can:

- manage user, groups and permissions;
- manage Authenticus multilingual translations;
- manage institutions (add/edit/remove, change structure, merge/split, update statistics);
- manage publications (update publications scores like citations, quartiles, view information from source, merge publications, manage detected publication duplicates, and define what metadata to show);
- manage journals (add/edit/remove, merge, update information);
- manage researchers (add/edit/remove, merge/split, update statistics, recalculate counters like keywords and co-authors);
- manage Authenticus task scheduler, Daemon (start/stop/kill, schedule tasks, view progress, view errors).

2.4.1 Publications Module

Authenticus publication module is the largest and one of the most important component of Authenticus. This module holds the data to produce statistics and views for researchers and institutions.

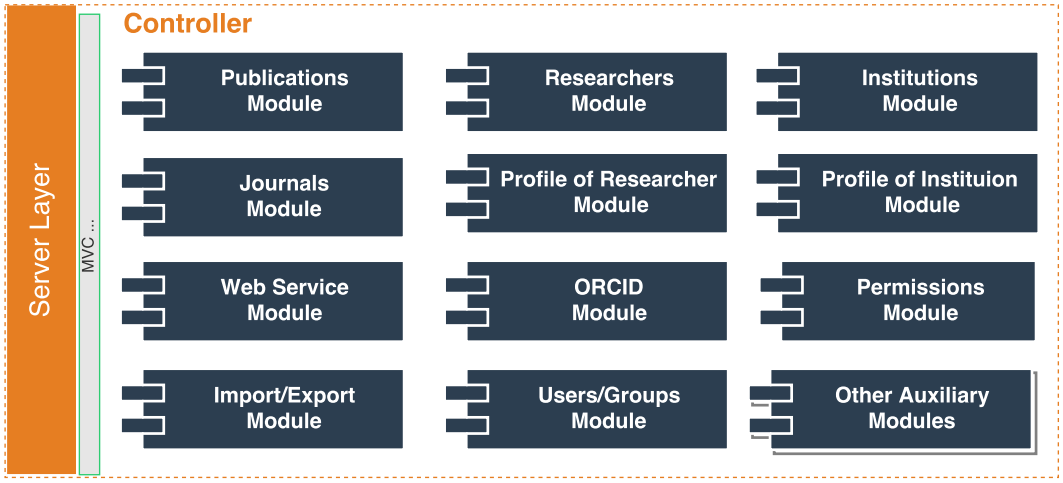


Figure 2.4: Main Modules of Authenticus Business Logic.

The publication module is extended with additional modules to handle complex tasks related with publication management. Those modules are:

- Import Module - to import publications into Authenticus;
- Export Module - to export publications from Authenticus in various formats;
- Web Service Module - to respond to specific service requests;
- ORCID Module - to interoperate publication metadata with ORCID, both import and export.

In this dissertation, we dedicated an entire Chapter 4 to fully explain the features and methods of the publication module.

2.4.2 Researcher Module

A researcher is an individual scientist usually affiliated with some academic institution, who pursues scientific study and produces scientific works. The researcher module is the second most developed part of Authenticus. To handle its complex structure and functionalities it is extended with an extra module, called *Profile of Researcher Module*. Both of the Authenticus researcher modules deal with researcher data, display of researcher profile, publications and various statistics.

There are public and private interfaces for researchers, where private is an extended version of the public interface. The public interface provides basic information about

a researcher that can be publicly displayed to guest users. This information includes a name and email of the researcher together with a list of confirmed publications and list of publications to validate at Authenticus.

Authenticated users of Authenticus have access to extended information about the researcher. This information is grouped into several sections. The first section, "Profile" includes basic data about the researcher and its work, namely variants of author names extracted from publications associated to the researcher, email addresses, external connections with other researcher profiles, such as ORCID, ResearcherID, DBPL or personal web page. It also contains an overview of the research accomplished, namely lists of co-authors, venues, scientific fields, subject categories, and a cloud of keywords, all extracted from publications associated with the researcher. The second section "Affiliation", shows the history of affiliations over the years of the researcher, both with higher education institutions and research units. Some of those affiliations were extracted from the researcher's publications, others were provided to Authenticus by external entities or by the researcher himself. The next two sections are dedicated to the researcher scientific publications. They include a list of publications already confirmed by the researcher, a list of publications to validate by the researcher and a list of potential duplicates to be clarified. The statistics section of the researcher page shows some general, descriptive, productivity and quality statistics of a researcher, such as: an histogram illustrating the number of publications over the years, top 10 publications by impact factor, top 10 publications by number of citations, co-authorship distribution by country, distribution of publications by type, etc. The last section is called "Co-Author Network" and it visualizes the collaboration graph of each researcher.

Whenever an authenticated user is associated with a researcher, he/she has access to the full profile, information and functionalities of its own researcher. Full access means that besides being able to visualize all the data, as each of the authenticated users of Authenticus can, the user can also perform actions in order to verify some data and to improve its quality within Authenticus. These include:

- validate publications that were automatically associated by the name identification algorithm as belonging to the researcher;
- remove publications from the confirmed or potential list of publications;
- add publications from Web of Science, Scopus, DBLP or from ResearcherID profile. To add a new publication, the researcher must provide either some publication identifier (e.g. WOS or eid-Scopus), the DOI, or DBLP-id;

- import and associate new publication from Web of Science, Scopus and DBLP;
- manage list of its duplicated publications;
- synchronize Authenticus researcher profile with ORCID profile, import of publications from ORCID and export of confirmed list of publication to ORCID;
- export of publications to Bibtex, EndNote, CSV, RIS and ISI;
- change profile public information;
- clean profile parameters in case they were wrongly associated with the researcher, thus removing wrongly associated publication;
- update researcher publication citations;
- print all or selected publications to PDF with or without citations and to HTML.

All the interfaces of the researcher module are simple and straightforward. Users should, in principle, intuitively navigate and interpret visible information.

2.4.3 Institution Module

The "'Institution Module" and its extension "Profile of Institution Module" handle displays and behaviors of data and functionalities related with institutions of higher education and research units.

There are public and private interfaces for institutions, where private is an extended version of the public interface. The public interface provides basic information about institution that can be publicly displayed to guest users. It includes basic characterization data, localization and contact data, institution type and, in case of a research unit, research classification and coordinator. Public intuition interfaces also show hierarchical organizational structure and structure of associations per year and a list of publications associated with this institution. In future versions of Authenticus, the institutional interfaces will also show lists of researchers associated with the institution, and some statistics about the productivity of the institution.

Authenticus provides an interface for institutions in which authenticated users with the right permissions can manage the institution profile and data related to it, namely update the basic institutional data, edit institution structure per year, manage list of researcher per year, associate publications to its researchers, etc. This is a module still

under development. It will allow for users to generate productivity reports for groups of researchers or institutions.

2.4.4 Users/Groups Management

Authenticus is equipped with back-office modules to simplify management of users, groups and permissions of those in connection with all other modules of the application. The user module allows one to manage user accounts, assign user to groups, assign a user to researcher. It is connected with a group module, where one user can belong to many groups. A powerful component of Authenticus is the Access Control Lists (ACL) component, a way to manage application permissions in a fine-grained, yet easily maintainable and manageable way. ACL in Authenticus is managed by the permissions module, where we define permissions and possible actions for each group of users.

This closes our ovierview of Authenticus and the next chapters will be dedicated to the design and implementation details on the author's main contributions.

Chapter 3

Action Center

The Action Center is a core component of Authenticus. It abstracts the database for Authenticus actions, controls and monitors the execution of actions, ensures logging of events and errors for future recovery purposes, and transparently ensures the consistent use of database transactions. Its development was justified to overcome limitations of the CakePHP framework, but more importantly to ensure database isolation and scalability of the application. Moreover, we wanted to improve the development process in terms of code maintainability and error detection.

In this chapter, we detail the implementation of the framework that constitutes the Action Center. It further includes a flexible and complete SQL Query Builder that simplifies query construction and also prevents SQL injections by using prepared statements, and to enable the use of caching strategies on specific recurrent queries. Figure 3.1 represents diagrammatically the internal structure of the Action Center. There are basically three components, one that implements helper classes and methods, another that implements the action classes and methods, and a third component that implements the Query Builder classes and methods. The following sections describe in detail the implementation of the Query Builder and Action components.

3.1 Query Builder

In order to define a common software development framework that enables developers to easily handle complex and large SQL queries in a systematic way, the Action Center includes a query builder that provides to developers a fluent interface to incrementally

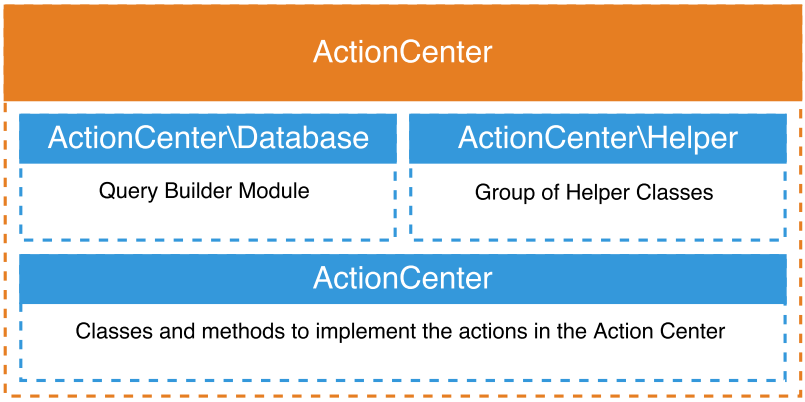


Figure 3.1: Action Center internal structure.

construct and execute SQL queries. It has been designed and implemented as a PHP API to interact, in our case, with MySQL databases, although it can be extended to handle other database management systems (DBMS).

3.1.1 Advantages and Limitations of CakePHP Models

Initial development of Authenticus used the CakePHP[1] framework fully, which quickly showed strong limitations, both in terms of development and efficiency, namely in building large and more complex SQL queries using the CakePHP Models API. In Cake for each table in the database, we need to create a model class and in each of these classes we have to define all the relations to other models (one to one, one to many, many to one and many to many) that although it is a laborious task to perform on a large database as is the case of Authenticus, it benefits the developers by automatically retrieving all the data related to an entity. For example, we only need to make a function call to get all the information related to a researcher: `$this->Researcher->find('first', ['conditions' => ['Researcher.id' => 1]]);`, and it returns all the researcher's personal information, all the publications, all the information related to the publications, etc.

The automated retrieval of Cake, as just described, does not always translates into a positive gain, it can also be a drawback. All the information related to an entity provided at once is often too much information, not needed in the same page. For example, in some cases we need the researcher's name and the email, in others we need the researcher's name and the list of publications, but we still do not need the

[1]CakePHP - http://cakephp.org/

full description of the publication, and by default Cake would grab it all. To bypass this problem Cake provided us two options: unbind unwanted relations before every find procedure [2], which is impracticable due the frequent changes in the database structure, or use the containable model behaviour [4]. This model behavior allowed as to specify the relations that we are interested in. For example, we could get only the researcher name and the oldest 10 publications as seen in code listing 3.1.

```
$this->Researcher->find('first', [
  'conditions' => ['Researcher.id' => 1],
  'fields' => ['name'],
  'contain' => [
    'Publication' => [
      'limit' => 10,
      'order' => 'Publication.year ASC'
    ]
  ]
]);
```

Listing 3.1: Example of Select in CakePHP model system.

This is great but since the containable behavior only modifies existing relationships it will not allow us to restrict results by distant associations. For example, in order to get the list of researchers with publications in computer science, we need to make custom join queries such us the code block 3.2, which is very verbose and it is hard to build dynamically since it uses a cascade of arrays.

```
$options['joins'] = array(
  array('table' => 'publications_authors',
    'alias' => 'PA',
    'type' => 'INNER',
    'conditions' => array(
      'PA.researcher_id = Researcher.id',
    )
  ),
  array('table' => 'publication_subject_categories',
    'alias' => 'PSC',
    'type' => 'INNER',
    'conditions' => array(
      'PSC.publication_id = PA.publication_id',
    )
  )
```

```
16 );
17 $options['conditions'] => [
18   'PSC.subject_category' => 'Computer Science'
19 ];
20 $options['fields'] => ['DISTINCT Researcher.id', 'Researcher.name'
     ];
21 $rows = $this->Researcher->find('all', $options);
```

Listing 3.2: Example of Join in CakePHP model system.

To be able to use models, we need to previously define what we would need in the definition of the controller. If in some action of the controller we need one other model we just have to load it dynamically using `$this->loadModel('Publication')` and then can immediately use it normally `$this->Publication->read(null, '1')`. But if we are outside of the controller, for example on a generic function to create a publication, we need to load and use the models needed as follows:

```
1 App::uses('Publication', 'Model');
2 $PublicationModel = new Publication();
3 $PublicationModel->read(null, '1');
```

Listing 3.3: Load CakePHP model.

Since to create a publication we need to read or write in 21 distinct tables, we need to repeat the previous code 21 times, and pay extra attention to the name of the variable (`$PublicationModel`) that does not have to be equal to the table name. It would be much simpler just to call the table name and submit the query.

Even though CakePHP allows one to submit queries with prepared statements, the CakePHP model system does not use prepared statements. However, these are important to dramatically improve security against SQL injection. In a standard SQL query, we mix both query info and data, for example `SELECT * FROM table WHERE column = 'value'`. In a prepared statement the process is divided in two: first we send the query to the server `SELECT * FROM table WHERE column = ?` and next we send only the data (through an API call), thus we do not need to sanitize it which is a benefit.

To overcome these drawbacks raised by the use of the models, we decided to put aside the CakePHP model system on the most complex and important parts of Authenticus and construct all the queries by hand. Since CakePHP 2.3 did not have a tool to build SQL queries without an associate model and we did not find any PHP tool that would allow as to use prepared statements, we started developing our own Query Builder.

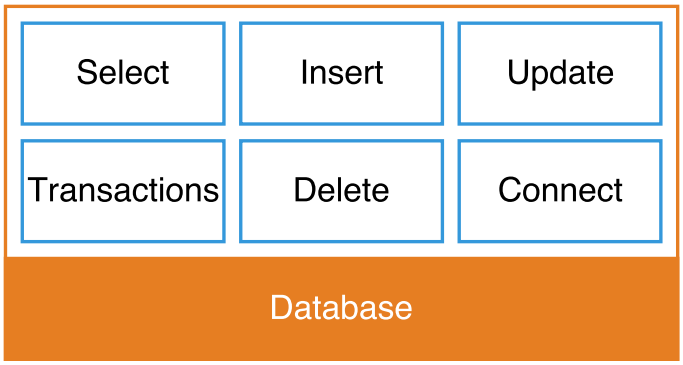


Figure 3.2: Query Builder Database high level representation.

3.1.2 Design

Authenticus Query Builder was designed to facilitate the development but without neglecting the performance, extensibility and still maintaining a low learning curve for new collaborators. The query builder is a collection of classes that, for easy access, are wrapped inside a main class called `Database` (Figure 3.2), which manage the database connection and exposes to the developer a friendly API to interact the database.

In Authenticus we have a security policy that each database should have its own user. This means that we cannot make one query that crosses several databases, neither share the same server connection for the existing databases. We need to create and manage one connection to the server for each database, as is the case in Authenticus for its databases associated with logging and data that, naturally, have different users. This is also what CakePHP does. Hence, to simplify, we use the CakePHP database management system to communicate with the database, which is basically an extension of the PHP MySQL PDO [2]. Using this CakePHP feature, we only need to configure the database connection once, on the CakePHP configuration file, and we also get on each page the log (Figure 3.3) of the executed queries, useful for debugging. But if needed, we can easily remove the dependency of the CakePHP and also allow execution of cross database queries, this without big changes in the code.

To initiate and start using the Query Builder for one database, we instantiate the `Database` class with the name given to that database in the CakePHP configuration file:

[2] "PHP MySQL PDO is a driver that implements the PHP Data Objects (PDO) interface to enable access from PHP to MySQL 3.x, 4.x and 5.x databases." - http://php.net/manual/en/ref.pdo-mysql.php

(default) 34 queries took 35 ms					
Nr	Query	Error	Affected	Num. rows	Took (ms)
1	SELECT COUNT(*) AS `count` FROM `mails` WHERE `user_id` IS NULL AND `processed`=? AND (`type`=? OR `type`=?) , params[0, SendAdmin, Reply]		1	1	0
2	SELECT COUNT(*) AS `count` FROM `researchers`		1	1	32
3	SELECT `researchers`.`name`,`researchers`.`id`,`researchers`.`emails` FROM `researchers` LIMIT 15		15	15	0

Figure 3.3: CakePHP SQL Log example.

```
1 $DB = new ActionCenter\Database\Database("database_name");
```

The returned object will provide to the developer a friendly fluent interface [6] to perform the different operations on the database, as data manipulation (select, insert, update, delete), manage transactions, manage the connection, perform clean queries, get the results, and others.

A fluent interface is an implementation of an object oriented API using method chaining, in which each method returns the context that should be used in subsequent calls. This improves code readability, and if the developer uses an IDE with auto-complete it is much easier to write the code as in each call the IDE shows all available operations for the current context. For example:

```
1 $DB->delete("table")
2   ->addCondition(condition)
3   ->execute();
```

Next we detail the implementation of the Query Builder, its functionalities and show some use case examples.

3.1.3 Implementation

As mentioned in the previous section the Query Builder is wrapped inside the class `Database`. This class is the entry point to the API and it is also used to manage the database connection. When the class is instantiated, it stores the CakePHP connection manager for the given database name and that connection is the one that will be used

in all the operations that Query Builder implements. We can divide the `Database`'s methods in two groups, one that represents simple operations that are immediately executed in the database, and another group that return the context fluent interface to some complex operation. In the Figure 3.4, we show a UML class diagram of the Query Builder with all the classes.

The first group includes the low level operations related with the database:

- **Manage database connection** - connect, reconnect, disconnect to the server;
- **Execute SQL queries and fetch the results** - execute query, fetch next row, fetch all rows, get last inserted id and get the number of affected rows;
- **Transaction management** - begin, commit, rollback transaction;
- **Nested transaction management** - begin, commit, rollback nested transaction;

MySQL does not support pure nested transactions, the statement `START TRANSACTION` does an implicit commit[3]. Thus, if we already are inside one transaction and attempt to start another, MySQL commits the first and then starts the second. But when we are inside a transaction MySQL allows to set savepoints which we could rollback in the future, and with that we could implement nested transactions. In Query Builder at the begining of a nested transaction method we send a query to the server to set a savepoint with a specified name: `SAVEPOINT $name`, on the commit of a nested transaction we release the savepoint name: `RELEASE SAVEPOINT $name` and on the rollback of a nested transaction we do the rollback to the given savepoint name: `ROLLBACK TO $name`.

The second group of methods of `Database` class returns the context fluent interface, namely, an instance of the class that implements functionalities, which are listed bellow.

- **Raw string** - the returned object holds a string and tells to the system that it should not be parsed nor sanitized in the query building process;
- **Select** - build, execute and retrieve a `SELECT` query on the specified table;
- **Insert** - build and execute `INSERT` query on the specified table;
- **Update** - build and execute `UPDATE` query on the specified table;

[3]MySQL Statements That Cause an Implicit Commit - https://dev.mysql.com/doc/refman/5.6/en/implicit-commit.html

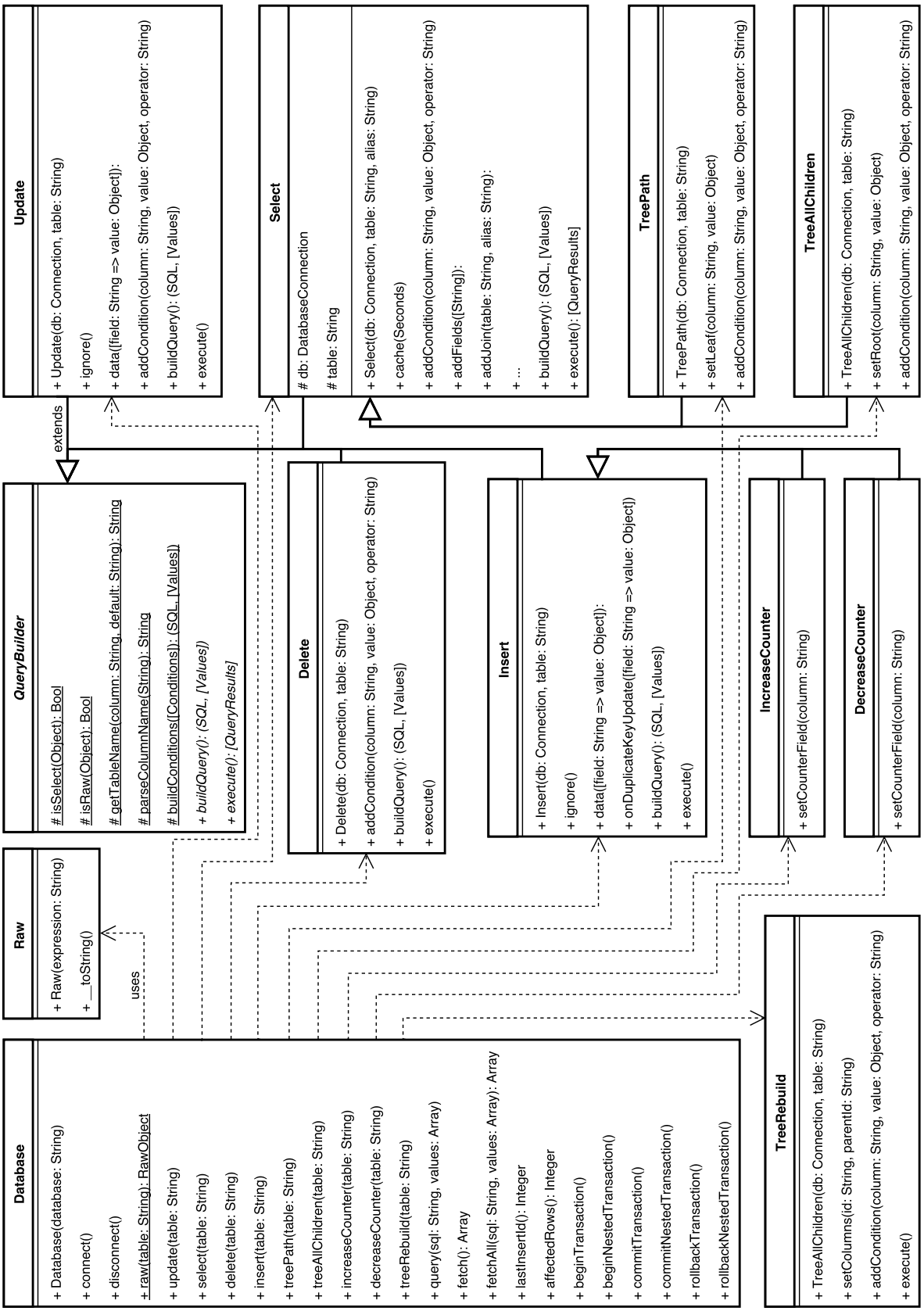


Figure 3.4: UML Diagram of the Query Builder Class.

- **Delete** - build and execute `DELETE` query on the specified table;
- **Increase Counter** - build and execute a query to create or increment a field by one, it uses the `INSERT` statement;
- **Decrease Counter** - it is similar to the previous method, but its effect is to decrement the field by one;
- **Tree Path** - is used in a table with a tree structure and will build and execute a query to retrieve the path from a node to the root of the tree;
- **Tree All Children** - is similar to the previous, but in this it will return all descendants, up to the leafs, starting from a given node;
- **Tree Rebuild** - is used in a table with a pair of column (id, parent id) and will transform that table to a tree structure;

As we can see in the UML class diagram in Figure 3.4 all the classes that provide an interface to build queries (`Update`, `Delete`, `Select`, `Insert`, `TreePath`, `TreeAllChildren`, `IncreaseCounter` and `DecreaseCounter`) extend the `QueryBuilder` class. This class defines that each of the actions have to implement two methods: `buildQuery()` and `execute()`. The `buildQuery()` produces a SQL Query and a list of values, because in the prepared statements we do not mix values with the query. The `execute()` executes the query produced by `buildQuery()` and returns the results. The `QueryBuilder` also provides useful methods to all query builders such as:

- `isSelect(Object)` - to check if some object is a `Select` query builder object. It is used to created nested select queries explained later in this section;
- `isRaw(Object)` - to check if the given object should not be parsed nor sanitized. This allows to insert a piece of handwritten SQL, like `sum(col)/count(col) as field` as a column name in a `SELECT` statement;
- `parseColumnName(String)` - given a string it parses the content and creates the SQL field, for example:

```
1 parseColumnName("*") -> "*"
2 parseColumnName("count(column)") -> "count(`column`)"
3 parseColumnName("table.column") -> "`table`.`column`"
```

- `buildConditions(...)` - it constructs the SQL string, and the list of values, for a given structured object of conditions.

Select

The `Select` class provides a fluent interface to build `SELECT` SQL queries. The constructor takes three arguments: a Query Builder `Database` object, the table name and an optional table alias. Instead of manually creating a new instance of the class we can call the `select` method of the correspondent `Database` object as shown in Listing 3.4.

```
1 public function select($table, $alias = null) {
2   return new Select($this, $table, $alias);
3 }
```

Listing 3.4: Authenticus QueryBuilder Database->select(...).

To build a query we start to get a `Select` object and specify the fields that we want to retrieve (line 2):

```
1 $query = $DB->select("researchers")
2               ->addFields("id", "name");
```

The `addFields` can be called as many times as needed. By default, when no `addFields` is used, all fields of a given table are returned, using SQL operator `*`.

During the development of Authenticus we notice that almost all query conditions are built mostly using conjunctions (`AND`). Thus to facilitate development, we decided that order the conditions in any query in the conjunctive normal form, that is, in our case, `WHERE c1 AND c2`, `WHERE c1 AND (c2a OR c2b)` or `WHERE (c1a OR c1b)`. Since the order of the conditions is not relevant, this ensures that new conditions can be added whenever necessary. For example:

```
1 $query->addCondition("country", "Portugal")
2       ->addFields("email")
3       ->addCondition("email", "%up.pt", "LIKE");
```

In line 1 we filter researchers from "Portugal", in line 2 we do something else, in this case select another field, and in line 3 we place another condition. We used the `buildQuery()` method (from `Database`) to preview the query, which at this point is:

```
1 # print_r($query->buildQuery());
2 array(
3   "SELECT `id`,`name`,`email` FROM `researchers` WHERE `country`=?
      AND `email` LIKE ?;",
4   array(
```

```
5       "Portugal",
6       "%up.pt"
7     )
8   )
```

If we need to do a disjunction we call `startOr()` that returns an object that allows us to insert `OR` conditions and also starts `AND` conditions:

```
1  $query->startOr()
2          ->addCondition("email", "%fc.up.pt", "LIKE")
3          ->addCondition("email", "%fe.up.pt", "LIKE")
4          ->startAnd()
5            ->addCondition("create_date", 2013, ">")
6            ->addCondition("email", "%fep.up.pt", "LIKE")
7          ->endAnd()
8          ->endOr();
```

The condition above produces following SQL query:

```
1  # print_r($query->buildQuery());
2  array(
3    "SELECT `id`,`name`,`email` FROM `researchers` WHERE `country`=?
       AND `email` LIKE ? AND (`email` LIKE ? OR `email` LIKE ? OR (`
       create_date` > ? AND `email` LIKE ?));",
4    array(
5      "Portugal",
6      "%up.pt",
7      "%fc.up.pt",
8      "%fe.up.pt",
9      2013
10     "%fep.up.pt"
11   )
12 )
```

The Query Builder allows one to use all MySQL condition operators, starting with `=` and `MATCH AGAINST`, among others. The argument of a condition can hold a value, a column, an array of values, and also a `Select` object that produces a `IN (Select ...)` condition.

Our Query Builder also supports table joins. In order to join tables, we call `addJoin(table)`, which provides the same methods to build conditions as seen in the `Select` method,

extended with one method that can be used to set the index that MySQL should use. For example, Listing 3.2 showed how one could get the list of researchers with publications in Computer Science in CakePHP, and now, using the Authenticus Query Builder, the code in Listing 3.5 shows how to accomplish the same.

```
1  $query = $DB->select("researchers")
2    ->distinct()
3    ->addFields("researchers.id", "researchers.name")
4    ->addJoin("publication_authors")
5      ->addJoinCondition("researchers.id", "publication_authors.
           researcher_id")
6    ->addJoin("publication_subject_categories")
7      ->addJoinCondition("publication_authors.publication_id", "
           publication_subject_categories.publication_id")
8    ->addCondition("publication_subject_categories.subject_category",
         "Computer Science")
9    ->formattingListResults(["researchers.id"=>"researchers.name"]);
10 $rows = $query->execute()
```

Listing 3.5: Authenticus QueryBuilder select with join example.

In line 10, we execute the query and it returns the results.

```
1  # print_r($query->buildQuery());
2  array(
3    'SELECT DISTINCT `researchers`.`id`,`researchers`.`name`
4    FROM `researchers` JOIN `publication_authors` ON (
5      `researchers`.`id`=`publication_authors`.`researcher_id`
6      )
7    JOIN `publication_subject_categories` ON (
8      `publication_authors`.`publication_id`=`
           publication_subject_categories`.`publication_id`
9      )
10   WHERE `publication_subject_categories`.`subject_category`=?',
11   array(
12     'Computer Science'
13   )
14 )
```

Listing 3.6: Authenticus QueryBuilder select with join buildQuery() output.

By default the results are returned in an array structure

`[line# => [table => [field => value]]]`. However, to simplify development we provide two extra methods, `formattingListResults` and `formattingResults`, which transparently format the results within the array to a desirable structure. Both methods can receive two types of arguments: (1) a column name if we only want one list with values of a column, or (2) an array such us `["key_column"=>["col1", "col2", ...]]`. The first method assumes that the key column is unique, the other does not. The result of the previous example 3.5 is as shown in Listing 3.7.

```
1  array(
2    135 => 'adriano da silva carvalho',
3    8018 => 'jorge manuel miranda dias',
4    16422 => 'rui paulo pinto da rocha',
5    7975 => 'jorge manuel dos santos ribeiro fernandes',
6    ...
7  )
```

Listing 3.7: Return data example using `formattingListResults()`.

For the same example, if we use the method `formattingResults()` the output is as shown in Listing 3.8.

```
1  array(
2    135 => array(
3      0 => 'adriano da silva carvalho'
4    ),
5    8018 =>  array(
6      0 => 'jorge manuel miranda dias'
7    ),
8    16422 =>  array(
9      0 => 'rui paulo pinto da rocha'
10   ),
11   7975 =>  array(
12     0 => 'jorge manuel dos santos ribeiro fernandes'
13   ),
14   ...
15 )
```

Listing 3.8: Return data example using `formattingResults()`.

Query Builder allows also to use `distinct()`, `groupBy()`, `orderBy...()`, `having()`, `union()`, and other methods to produce the desirable `Select` query.

As an extention of the `Select` method exists a caching mechanism, which allows to catch a query for a certain time in memory. It is implemented with a `cache(seconds)` method, which stores the query results in a Memcached[4] server. Memcached is specified in the Authenticus configuration files.

It is also possible to do a nested select where instead of selecting a table name, we select another `Select` object:

```
1  $query = $DB->select($DB->select("publication_authors")
2                              ->addFields("publication_id", "count(*)")
3                              ->groupBy("publication_id"), "a")
4              ->addCondition("count", 100, "<=")
5              ->orderByAsc("count");
6
7  # print_r($query->buildQuery());
8  array(
9    "SELECT * FROM (SELECT `publication_id`, count(*) as count FROM `
       publication_authors` GROUP BY `publication_id`) as a WHERE `
       count`<=? ORDER BY `count` ASC",
10   array(
11     100
12   )
13 )
```

Insert

To instantiate the `Insert` query we need to provide a database name and a table name to be affected, and as in the `Select` case, we can easily get an instance of it by calling the `Database` method `insert($table)`. Beyond the obligatory `buildQuery()` and `execute()` methods, the Insert Query Builder provides three extra methods that allow one to properly build `INSERT` SQL queries. The extra methods are: `data($data)`, `ignore()` and `onDuplicateKeyUpdate(data)`.

To insert a row in a table we only need to set the row data using `data($data)` in which data is an array in the form `["column"=>$value, ...]`. The value could be a string, a number, a boolean or a `Raw` string.

```
1  $query = $DB->insert("researchers")
```

[4] http://memcached.org

```
2                 ->data([
3                     "name" => "Tiago",
4                     "email" => "tiago@example.com"
5                   ])
6                 ->execute();
```

The previous example produces the following SQL query:

```
1 # print_r($query->buildQuery());
2 array(
3   "INSERT INTO `researchers` (`name`, `email`) VALUES (?, ?)",
4   array(
5     "Tiago",
6     "tiago@example.com"
7   )
8 )
```

In the QueryBuilder, the `ignore()` method is used to insert the `IGNORE` SQL keyword in the query. Similarly, to update something whenever a duplicate key error occur, we use the `onDuplicateKeyUpdate($data)` method with the fields that we want to update, for example, to insert or increment a column. The next listings illustrate the code to generate an SQL query using an insert and a `onDuplicateKeyUpdate($data)` method, and then the preview of the SQL query that resulted from it. The example increments, or creates if they do not exist, the co-author counters among two researchers by one unit.

```
1 $query = $DB->insert("researcher_co_researchers")
2                 ->data([
3                     "researcher_id_1" => 10,
4                     "researcher_id_2" => 20,
5                     "count" => 1
6                   ])
7                 ->onDuplicateKeyUpdate([
8                     "count" => $DB->raw("`count`+VALUES(`count`)")
9                   ]);
```

Listing 3.9: Authenticus Query Builder Insert On Duplicate Key

```
1 # print_r($query->buildQuery());
2 array(
```

```
3   "INSERT INTO `researcher_co_researchers` (`researcher_id_1`, `
        researcher_id_1`, `count`) VALUES (?, ?, ?) ON DUPLICATE KEY
        UPDATE `count`=`count`+VALUES(`count`)",
4   array(
5     10,
6     20,
7     1
8   )
9 )
```

Update

The Update method of a Query Builder is called the same way as the Insert method and it provides additional methods such as: (1) `ignore()`, (2) `values(values)` to set the fields and values to be updated (similarly as in `Insert`), and (3) the same methods to build the conditions as described for the `Select` method.

```
1 $query = $DB->update("researcher_co_researchers")
2             ->values([
3                 "count" => 0
4               ])
5             ->addCondition("researcher_id_1", 10);
```

```
1 # print_r($query->buildQuery());
2 array(
3   "UPDATE `researcher_co_researchers` SET `count`=? WHERE `
        researcher_id_1`=?",
4   array(
5     0,
6     10
7   )
8 )
```

Delete

The Delete Query Builder is called the same way as all the previous methods, and has the same methods to build the conditions as described for the `Select` method.

```
$query = $DB->delete("researcher_co_researchers")
             ->addCondition("researcher_id_1", 10);
```

```
# print_r($query->buildQuery());
array(
  "DELETE FROM `researcher_co_researchers` WHERE `researcher_id_1
     `=?",
  array(
    10
  )
)
```

Increase and Decrease Counter

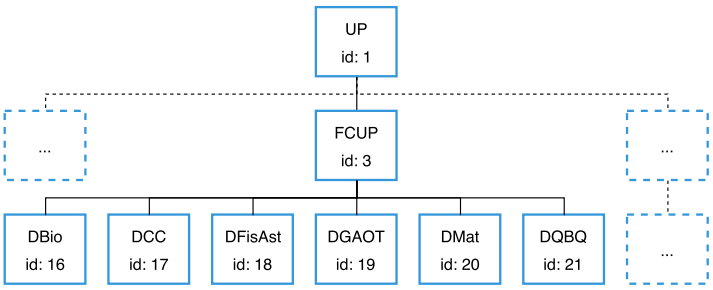
The Increase and Decrease Counter Query Builders are an extension to the class `Insert` with a `setCounterField($column)` method that in case a duplicate key error occurs it has the effect of increasing or decreasing the counter field by the given value. This is highly useful in Authenticus for updating indicators related to a researcher, for example the counting of keywords or co-authors. The example in Listing 3.10 will produce the same query as the `Insert` example in Listing 3.9.

```
$query = $DB->increaseCounter("researcher_co_researchers")
             ->data([
                  "researcher_id_1" => 10,
                  "researcher_id_2" => 20,
                  "count" => 1
                ])
             ->setCounterField("count");
```

Listing 3.10: Authenticus Query Builder Increase example

Tree Path

In Authenticus, institutions (higher education and research units) are organized in a tree structure where each institution can have one and only one parent institution. That information is recorded in one table with two columns: `institution_id` and `institution_parent_id` as illustrated in figure 3.5.



(a) Tree representation

ID	Parent ID	Name
1	NULL	UP
3	1	FCUP
16	3	DBio
17	3	DCC
18	3	DFisAst
...	...	...

(b) Table representation

Figure 3.5: Institution structure example

With this kind of structure we can easily get the parent or the children of some institution, as it only takes one query. But if we want to get the path from one node to the root of the tree, for example the path of DCC is UP→FCUP→DCC, we need to do it recursively. First we get the parent of DCC, which is FCUP, then the parent of FCUP, which is UP, then the parent of UP which is `NULL` and stops the recursion. It took three queries, one for each node. The same thing happens if we want the list of all descendants, from the node up to the leafs, which in the case of FCUP requires to execute seven queries. As these operations are often performed, for example every time that we print the name of some institution we also show the path to the root, we use a more structured table, described in [14] that allows us to get both the path and the descendants with only one query.

The `TreePath` is an extension of the `Select`. It provides a method that given a node in the tree, it returns the path to the root of the tree. Since it extends the class `Select`, it inherits methods that correspond to SQL constructs as is the case of a join or where operations.

```
1 $query = $DB->treePath('institutions_tree')
2               ->setLeaf('id', 61)
3               ->addCondition("year", 2012)
4               ->addJoin("institutions")
5               ->addJoinCondition('institutions_tree.id', '
                    institutions.id')
6               ->addFields('institutions.name', 'institutions_tree.id'
                    , 'institutions_tree.parent_id');
7
8 # print_r($query->buildQuery());
9 array(
```

```
10    "SELECT `institutions`.`name`,`institutions_tree`.`id`,`
          institutions_tree`.`parent_id` FROM `institutions_tree` JOIN `
          institutions` ON (`institutions_tree`.`id`=`institutions`.`id
          `) WHERE `lft` <= (SELECT `lft` FROM `institutions_tree` WHERE
           `id`=? AND `year`=?) AND `rght` >= (SELECT `rght` FROM `
          institutions_tree` WHERE `id`=? AND `year`=?) AND `year`=?
          ORDER BY `lft` ASC",
11    array(
12      61,
13      2012,
14      61,
15      2012,
16      2012
17    )
18  )
```

Listing 3.11: Get path of the institution 61 for the year 2012, and the name of the institutions.

Tree All Children

`Tree All Children` Query Builder is similar to the method `TreePath`, but instead of returning the path to the root it returns all descendants, up to the leafs, starting from a given node. For example, if we give the "University of Porto" as the node value, it will give us all the faculties (direct descendants) and all sub-unities within the faculties (indirect descendants).

```
1  $query = $DB->treeAllChildren('institutions_tree')
2              ->setRoot('id', $instituionID)
3              ->addCondition("year", 2012)
4              ->addCondition("source", 61)
5              ->addJoin("institutions")
6              ->addJoinCondition('institutions_tree.id', '
                   institutions.id')
7              ->addFields('institutions.name', 'institutions_tree.id'
                   , 'institutions_tree.parent_id');
```

Listing 3.12: Get the descendants of the institution 61 for the year 2012, and the name of the institutions.

Tree Rebuild

This method is used to transform table with a pair of columns (id, parent id) to a tree structure. It does not extend `QueryBuilder`, nor build a query, it is more like an helper, which gets all pairs, builds a tree and updates a tree table.

```
1 $query = $DB->treeRebuild("institutions_tree")
2             ->setColumns("id", "parent_id")
3             ->addCondition("year", 2012)
4             ->execute();
```

Listing 3.13: Rebuild the institutions tree for a given year.

An alternative solution to our Query Builder is the Laravel query builder[5] [13]. There are many similarities among the builders although our work was developed independently in the beginning of 2013, and without knowing of the existence of Laravel.

3.2 Action

An *action* is an operation that runs on a semi-controlled environment of the Action Center, which ensures that each action executed is atomic and properly logged. Action logging is a necessary part of every application. In Authenticus action logging consists of the details about the executed actions, such as: name, parameters, invoking user and date and information about the execution result, namely the time spend and error or warnings. The second feature of an *action*, atomicity, is a guarantee of action isolation from concurrent processes in which the operations involved have a succeed-or-fail definition, that is they either successfully change the state of the system, or have no apparent effect.

To achieve these proprieties, we created an object oriented scheme where every action extends the same main abstract class, which is responsible for controling the execution of the operation's code. The main abstract class `Action` defines two protected abstract methods that every action must implement, the `executeAction()` and `executeUndoAction()`, where we put the code to do the operation and the inverse of it. The approach of having in one action two opposite operations is a great advantage for developers as almost every Authenticus operation that changes the database may have to be undone. It also prevents the proliferation of pairs of similar actions like

[5]Laravel - `http://laravel.com`.

`AssociatePublication()` and `DisassociatePublication()`. We implement only one `AssociatePublication` action that includes two operations `executeAction()` and `executeUndoAction()` which in the end have the same effect, but in a more disciplined way.

The `Action` class does not overload nor extend any PHP 5 class. Each action in the Action Center has its own input needs, thus extension of a PHP class is impossible, as an extended class cannot have a constructor with a different number of arguments of the base class. To solve this limitation and to keep the code readable, we decided to put the `Action` constructor protected and force every action to provide a public static method that instantiates the action class. By convention that method is called `init(...)` and will take as many arguments as needed.

3.2.1 Loggings

One goal of the Action Center is to register every action that is executed. This allows to detect and debug errors, and in the future, perform some statistical analysis using the logs records. To make the logs reliable and reusable, the following data fields are logged with every action being executed:

- unique ID of the parent action;
- action name;
- parameters - arguments used in the `init(...)`;
- whether the action is a *do* execution or an *undo* execution;
- ID of the user;
- whether it is executed by the daemon or by the web application;
- time and date of creation;
- developer defined warnings;
- PHP errors;
- time spent on exection.

Most of the fields to be logged are easy to collect, except the parameters and the errors. For unit of the creation time we use seconds instead of a higher precision value. This is a result of the time differences in the internal clock of each core, which often are not synchronized and could produce erroneous logs. This situation was reported in the beta version of the logging mechanism, when one child action was recorded as created before the parent action.

Action Parameters

Every *action* in an Action Center may have different number of arguments. To register the action arguments we use `func_get_args()` function on `init(...)`, however, it only returns an array of values and not the argument names that are very useful for debugging. To overpass this limitation, instead of the typical variable assignment seen on constructors:

```
1 public static function init($x, $y) {
2     $this->x = $x;
3     $this->y = $y;
4 }
```

we use an auxiliary function `addParameter($name, $value)`:

```
1 public static function init($x, $y) {
2     $this->addParameter("x", $x);
3     $this->addParameter("y", $y);
4 }
```

that besides the variable assignment also registers the pair name-value in the log. This approach may have some minor performance penalty and also use more memory, but it promotes code readability, traceability and ease of development.

Errors

Tracing and debugging errors of actions' execution is another challenge we had to deal with. Our experience shows that many of the reported errors originate in the standard PHP 5 libraries. PHP is not fully an object oriented language, only PHP 5 has a full object model[6]. Standard PHP error handling is done in text into the standard

[6]http://php.net/manual/en/oop5.intro.php

output, but what is worst, not all types of errors have the same level of severity and so the behavior of PHP is different. A PHP script on error occurrence could terminate immediately or continue as if nothing had happened. This is very dangerous especially if we want to ensure atomicity of the actions. An error may occur that changes the expected output of the action, and since no one noticed the error, the action is seen as a success.

In the Action Center we handle all errors and warnings and then throw an exception that could be catched with a `try catch()` statement. Every time that one action is executed, we set an error handler function with `set_error_handler()` and that handler throws an exception with the name of the type of error/warning and its description: name, file and line.

However, there are some errors that are fatal and can not be handled, for example in case of exceeding the maximum execution time or maximum memory allowed. When one fatal error occurs, the context of the execution is lost and the script terminates immediately. PHP has a function that allows one to register a function that is called when the script terminates, `register_shutdown_function()`. The registered function verifies if the last action has terminated, and if not, then it calls `error_get_last()` to check if some error has occurred. If the type of the error is `E_ERROR` we log it. We had to change CakePHP error handler to avoid it from dying preemptively and prevent the registered function to be executed.

Since some errors could be related to the database connection, we decided not to log directly into the database, instead the log is done to a file in two steps: first it is created a JSON entry with the name `insert` and an action unique ID (`uniqid(null, true)`), and when the action terminates, or an error occurs, we create another entry with the name `update` with the same action ID and the result of the action (that is, the error and execution time). This file later is loaded and processed into the database to simplify its analysis.

The following illustrates an example of a few log lines:

```
1 {"log_operation":"insert","action_id":"561da2c64c1395.50908899","
    data":{"parent_action_id":"561da2c64b8595.19371729","name":"
    ActionCenter\\ValidatedPublicationsWebserviceLog","is_undo":0,"
    parameters":"{\"author_id\":\"5079043\"}","user_id":"1","
    user_group_id":"1","started_from_background":0,"created_date":"
    2015-10-14 01:33:10"}}
2 {"log_operation":"update","action_id":"561da2c64c1395.50908899","
```

```
data":{"duration":0.00092792510986328}}
```

As an example of what these logs allow, other than serving to detect errors, we can analyse the logs to derive statistical information that results from the users activity in Authenticus, which corresponds to the execution of actions. The following figures, Figure 3.6, 3.7, 3.8 show, respectively, the evolution of the number of actions executed by Authenticus users (not including the administrators) in the period of 1st of October 2015 and 6th of January 2016, the average number of actions executed per user, and also the number of users that executed actions. The figures show, for example Figure 3.8 , an increase in the number of users performing actions in Authenticus, which an inside observation to the logs show that correspond to publication validation actions.

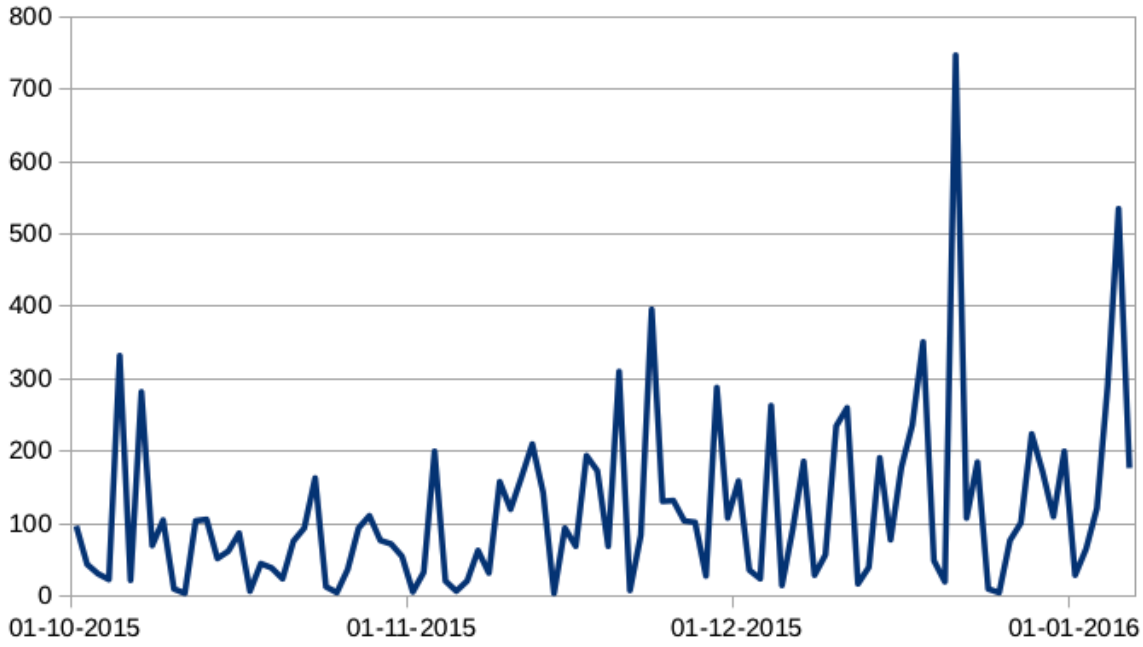


Figure 3.6: Number of actions executed by Authenticus users.

3.2.2 Atomicity

From a developer's point of view an action is a black box that does some task, and in case of failure it throws an exception that can be caught and no changes are committed. The error handler of the action center must be active to automatically throw exceptions for all kinds of errors and warnings, especially the non fatal ones. It happens often that one action with a status successfully completed, throws a warning which was not visible due to the settings of PHP error handling. The action was executed, its behavior was

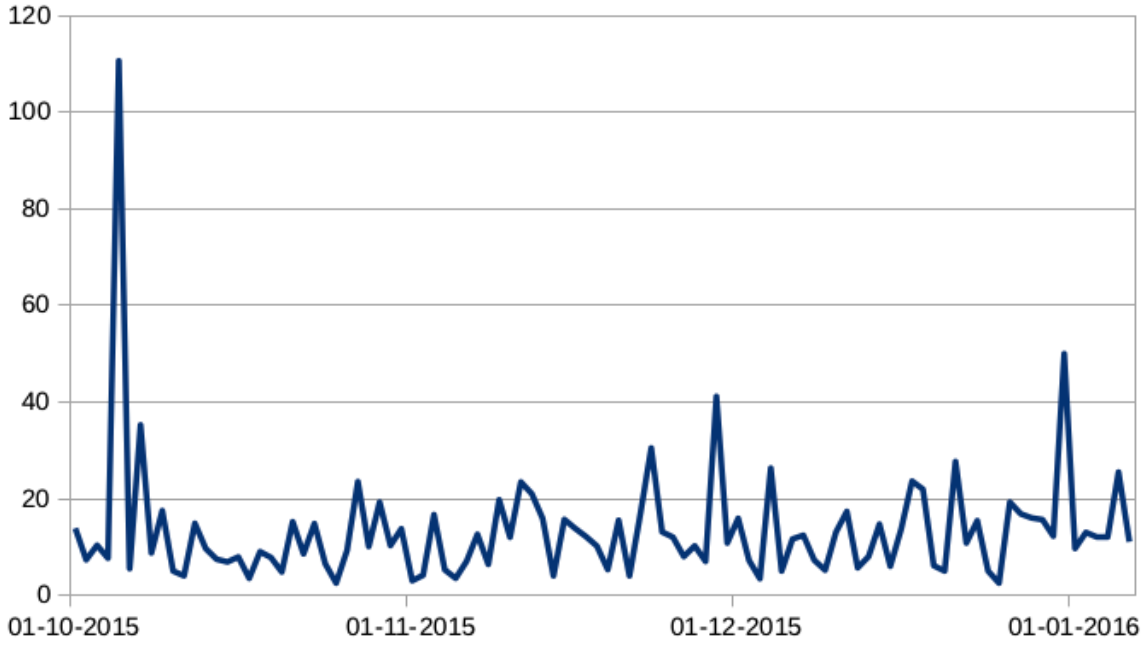


Figure 3.7: Average number of actions executed per Authenticus user.

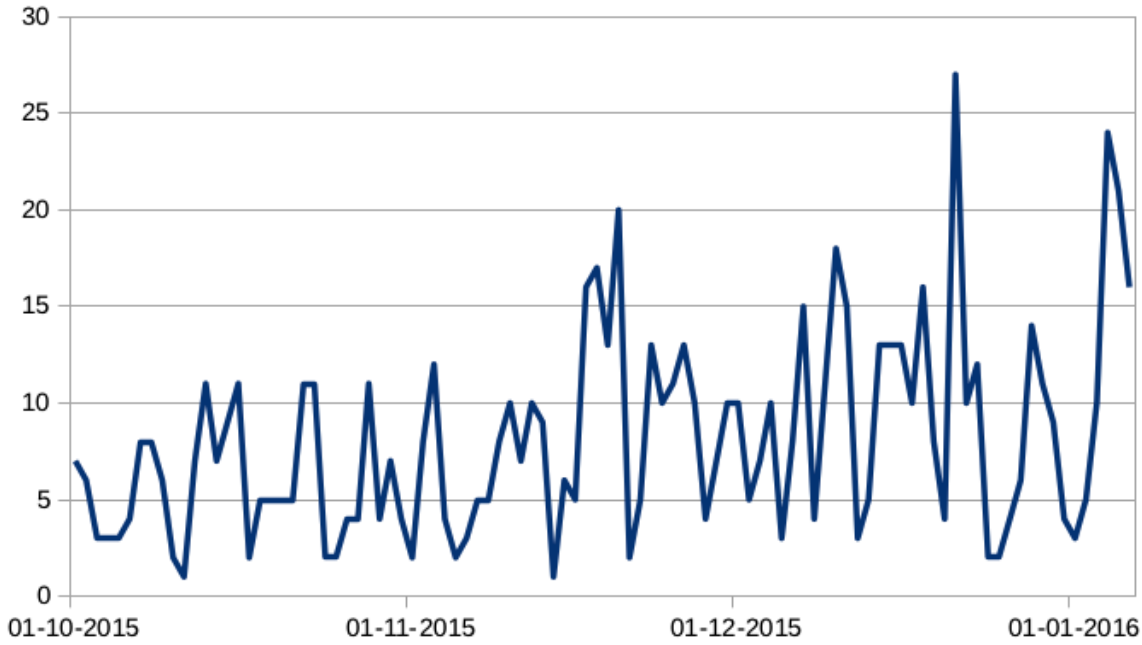


Figure 3.8: Number of Authenticus users that executed actions.

not the expected and no one could easily and immediately notice the warning. For example, in PHP the following code would be fully executed:

```
1 $empty_dictionary = [];
2 $user_id = $empty_dictionary["user_id"];
3 do_something_with($user_id);
```

as the dictionary does not have the key `user_id` PHP emits a warning, assigns `null` to the variable `$user_id`, and nevertheless the function in the last line is executed. With the error handler active in the action center, an exception is thrown when line 2 is executed and the developer can immediately notice the bug.

When an error occurs it is important that all data changes done by the action being executed are reverted. Since all the data that we use and modify in Authenticus is stored in a MySQL database, action center can use database transactions to undo the changes. Before one action is started we initiate a database transaction, and if the action succeeds we commit, otherwise we do a rollback.

3.2.3 Cascade of Actions

Another important feature of *actions* is called cascade of execution, which allow one action to execute another action. To keep the expected proprieties, namely logging and atomicity, in theory we just need to execute the same procedures as in the parent action. However, this is much more complicated to be accomplished because MySQL does not support nested transactions.

The following is an example of a cascade action that fails due to the nested transactions executions.

1. **Action 1** executed, start a transaction;
2. **Action 2** executed, start a transaction, but in MySQL the `START TRANSACTION` statement does an implicit commit, so all the changes made between 1 and 2 are saved;
3. **Action 2** succeeds, commits the transaction open in 2; at this point all changes made are saved;
4. **Action 1** continues the execution, does some queries, since the transaction has already committed at this point the queries are executed outside of any transaction

and so they are automatically committed;

5. **Action 1** fails, rollback transaction, but MySQL does not have any transaction so it will ignore the query;

Our first attempt to solve this problem only used start and commit/rollback transaction in the outer Action. This works only if there are no errors in the inner actions. An error in an inner action will trigger the `try catch()` which makes the outer action to deal with the error and succeed thus preventing the rollback of any inner action. The problem is that the modification done by action 2 until it failed is not reverted, and it should because it failed.

1. **Action 1** executed, start a transaction;
2. **Action 2** executed, we are not the head (or outer action), continue;
3. **Action 2** fail, throw exception;
4. **Action 1** catch the exception and continues the execution;
5. **Action 1** succeed, commit transaction;

The MySQL supports the `SAVEPOINT` statement, which allows one to create a checkpoint at some point in the transaction and name it. This allows us to rollback to that saved point without loosing what was done before. With `SAVEPOINT savepoint_id` and `ROOLBACK TO savepoint_id` we implemented nested transactions using the following strategy:

1. **Action 1** executed, start the transaction;
2. **Action 2** executed, `SAVEPOINT action_2_id`;
3. **Action 2** fail, `ROLLBACK TO action_2_id`, throw exception;
4. **Action 1** catch the exception and continues the execution;
5. **Action 1** succeed, commit transaction;

3.2.4 Action Life Cycle

The last thing worth to mention while characterizing *actions* is their action life cycle, namely what happens in an action during its execution. To simplify the understanding of an action life cycle, we illustrate a concrete example of a class implementing an action, in this case `RaisesAllSalaries` action. It includes the three basic operations, the `init()` method to initialize the action with specific parameters, returning an action instance ready to be executed. It also includes methods that implement the action and its reverse, namely `executeAction()` will for each employee create an action and execute it to raise his salary, and `executeUndoAction()` does the opposite.

```
1  class RaisesAllSalaries extends Action {
2      public static function init($amount) {
3          $instance = new self();
4          $instance->addParameter("amount", $amount);
5          return $instance;
6      }
7      protected function executeAction() {
8          foreach(employees() as $employee)
9              RaisesSalary::init($employee, $this->amount)->run();
10     }
11     protected function executeUndoAction() {
12         foreach(employees() as $employee)
13             RaisesSalary::init($employee, $this->amount)->undoRun()
                   ;
14     }
15 }
```

The execution of an action always starts with a call to the `init(...)` method that will set all the parameters with `addParameter($name, $value)`, and return an instance of the action. Then, one must call `run()` or `undoRun()` of the class `Action` to execute the code in `executeAction()` or `executeUndoAction()`.

```
1 ActionCenter\RaisesAllSalaries::init(20)->run();
2 ActionCenter\RaisesAllSalaries::init(20)->undoRun();
```

The `run()` and `undoRun()` are public methods implemented in the `Action` abstract class that manages the execution of the called action. The implementation of these methods follows the following structure:

```
beforeRun();
// execute and save the returned $value and the $exception
// if it exists;
executeAction(); or executeUndoAction();
afterRun();
return $value;
```

In this case, the `beforeRun()` is responsible to proper logging, error handler and start transactions:

```
beforeRun() {
  start log;
  if is head of the actions:
     start error handler;
     start transaction;
  else:
     start a nested transaction;
}
```

The `afterRun()` is responsible for detecting termination status, committing the transaction or handling the rollback as necessary, and closes the log.

```
afterRun() {
  if is head of the actions:
     if has exception:
        rollback transaction;
     else:
        commit transaction;
  else:
     if has exception:
        rollback nested transaction;

  if is head of the actions:
     disable error handler;

  update log;
  if has exception:
     throw exception;
}
```

3.3 Daemon

With the growth of the Authenticus interface functionalities we observed the need to decompose and prioritise the execution of some complex actions that are more demanding in terms of CPU and memory, easily exceeding the PHP limits that per default are 30 seconds and 16MB. Our solution was to develop a task scheduler, that we call daemon, to execute the actions in background and implement a daemon that is awaken whenever new tasks are added to the task queues stored in the database to deal with concurrency. A task is basically an action to be executed.

The execution life cycle of the daemon is quite simple:

1. get action from table;
2. mark the action as executed;
3. execute the action;
4. if there is new action go to 1 otherwise sleep;

The class `Action` includes two methods `runBackground()` and `undoRunBackground()` that instead of running immediately the actions, inserts them as new tasks in the database. The action center is responsible for sending a wake up signal to the daemon if there are tasks to be executed. For each task that is added, the following data is recorded:

- action ID - used later to match the daemon table with the logs table;
- parent action ID;
- action name;
- if the action was a *do* execution or an *undo* execution;
- user ID;

To execute an action the daemon loads the named action class and calls the action method `initFromParametersBackground($parameters, $id, $parent_id)` to obtain an instance of the action, where `$parameters` is a JSON string with a name-value dictionary, and than calls the method `run()` or `undoRun()`.

Queues

An easy way to do parallel work in background is by having multiple queues in the database and open multiple concurrent daemons, one per queue. To put an action in one particular queue we use the `Action`'s method `setBackgroundQueue($queue_name)`.

Priorities

Some actions may have to be executed immediately while others can be delayed. We allow priorities for actions to determine how fast they should be handled for execution. A new data field `priority` is used to record the priority of an action. The actions with lower priority number will be the first to be executed. To set the priority, we use the `Action`'s method `setBackgroundPriority($value)`.

Scheduling

Often, we need to schedule tasks for future execution or even to be periodically executed, for example to update the import of publication from Scopus, or consolidating the log-file into the database, etc. For this, we record a new data field with the task that is the `schedule_date`. The daemon will only get actions that have a `schedule_date` smaller than the present time. To schedule an action we call `runBackground($delay)` or `undoRunBackground($delay)` in which `delay` is the time in seconds added to the current time to define the `schedule_date`.

To handle periodic tasks, we implemented the `Cron` action that has as arguments the action name, action parameters, if it is a do or undo operation, and the periodicity. Before it runs the action with the given parameters, the action schedules itself with the same parameters for execution in `time()+periodicity`, where periodicity can be `minute`, `hour`, `week`, `month` or `year`.

Implementation Details

To implement the daemon we used CakePHP shell[7] because we can use the same configurations to access the databases and directories used by the Web application.

[7]Improve this Doc Shells, Tasks & Console Tools - `http://book.cakephp.org/2.0/pt/console-and-shells.html`

Furthermore, it automatically parses shell arguments. The daemon has five possible arguments:

- start queue - starts the daemon in the specified queue; we first check if there is already one daemon in that queue by inspecting a file lock named `queue.daemon.pid`. This file also stores the process ID of the daemon. The following pseudo-code gives the bookkeeping details needed for the daemon execution life cycle, namely to get tasks from the queue and execute them, or else to enter sleep mode.

```
1  if cannot lock the file:
2      exit with error, daemon already running;
3  fork();
4  if parent:
5      write the child process ID to the lock file and exit;
6
7  create a new session ID;
8  close standard output, input and error channels;
9  redirect standard output and error to queue.daemon.log;
10 tell to kernel to ignore SIGTSTP, SIGTTOU, SIGTTIN and SIGHUP
       signals;
11 set process user id and process group id to the www-data, so
       apache user can send signals to daemon without root;
12 set SIGTERM handler;
13 block default SIGALRM processing;
14
15 while not SIGTERM:
16     reconnect to databases;
17     if getNextTask():
18         runTask() in a child process and wait until it ends;
19     else:
20         sleep(seconds) until next task:
21             schedule a kernel alarm signal to X seconds;
22             block process until a SIGALARM;
23             clean schedule alarm signals;
24     process pending kernel signals;
```

We need to reconnect to the database on every action executed because when the child PHP process exits it closes all active MySQL connections. The `runTask()` is responsible to update the status of the execution on the daemon table. To put the action as DONE we have to do some trick to register a function that is

executed immediately before the process terminates, we register a shutdown that when executed will register another shutdown function and is the second that updates the table.

- wakeUp queue - wakeup daemon in the specified queue; the corresponding pseudo-code to implement the wakeup daemon is as follows:

```
1 if can lock the file:
2     error daemon is not running;
3 get process ID from file;
4 send SIGALRM to process ID;
```

- stop queue - stop deamon in the specified queue;

```
1 if can lock the file:
2     error daemon is not running;
3 get process ID from file;
4 send SIGTERM to process ID;
```

- kill queue - kill deamon in the specified queue;

```
1 if can lock the file:
2     error daemon is not running;
3 get process ID from file;
4 kill all process tree:
5     pkill –KILL –P pid
6 send SIGKILL to process ID;
```

- isRunning - return 0 if running, 1 if not;

```
1 if can lock the file:
2     exit(1);
3 exit(0);
```

This concludes our description of the Action Center. We described its key components and their implementation, namely the Query Builder, the Actions, the Logging, and the Task Scheduler. The Action Center is a core component of Authenticus as it controls and monitors the execution of the actions that are performed within the system, while ensuring logging, error control, data consistency, etc.

Chapter 4

Authenticus Publication Management

Currently, the largest and one of the most important components of Authenticus is the publications module. This is so because we rely on that module's data to produce all the statistics and some views for the researchers and institutions. Consequently, we need to be very careful and efficient when dealing with publication data. To accomplish our goals, Authenticus splits the entire process of handling publications data in three independent steps: import the metadata into Authenticus from multiple indexing repositories; pre-process the original metadata, normalize it, and convert it to an Authenticus publication record, check if the publication is duplicated (or redundant) and create or update the publication record; identify and associate the metadata to existing researchers and institutions.

To simplify the understanding of this rather complex module, we divide its description into several sections within this chapter in the attempt to answer the following questions:

- What is an Authenticus Publication Record and how is it represented in Authenticus?
- How do we import a new publication into Authenticus?
- What processes do we run to store a new Authenticus Publication Record in the database?
- How do we pre-process publication data and associate it to researchers?
- How do we preview a publication and what are publication profiles and interfaces?

- What can we do with the publication record and how do we interoperate publication metadata?

4.1 Publication Module

The publications module plays a crucial role, and its functionality is determinant for other modules of Authenticus. Example of modules that rely the publications module are the module of researchers, the module of institutions and the module of statistics and reports. Possibility to review the metadata sources, updating the metadata from sources, and duplicates detection are only a few of the issues we had to deal while designing this module. In the following sections, we focus on defining the Authenticus publication record and its data model representation.

Authenticus Publication Record

Authenticus publication record (APR) is a complex data structure with many attributes that fully characterise a publication in Authenticus. Every publication that is imported into Authenticus is transformed into an APR. Publication records are stored in the database and Figure 4.1 illustrates a partial view of the tables, and the dependences between them, that compose an Authenticus publication record. This is a high level representation, we do not represent primary nor foreign key constraints, however it gives the reader a good understanding of the complexity of a publication structure.

The main table in the Authenticus publication record database structure is the table `Publication`. It has over 50 columns, holds the publication primary key, used all over the Authenticus application and includes details such as title, year published, pages, volume number etc. Depending on the source of the publication metadata, the publication record can be contextually rich or contain just a minimal description. When the publication metadata is imported from ISI Web of Science or SCOPUS it is usually very complete, thus we can extract maximum amount of information and the APR is very rich. The main `Publication` table is connected with other tables in the database. Some of them are obligatory, such as `Authors`, `Document Type`, or `External IDS`, others are auxiliary and in many cases those relations exists only when the APR is contextually reach. For example, in the case of publication records imported from DBLP, we do not retrieve any information about scientific fields or citations counts because such information is non existant in their metadata.

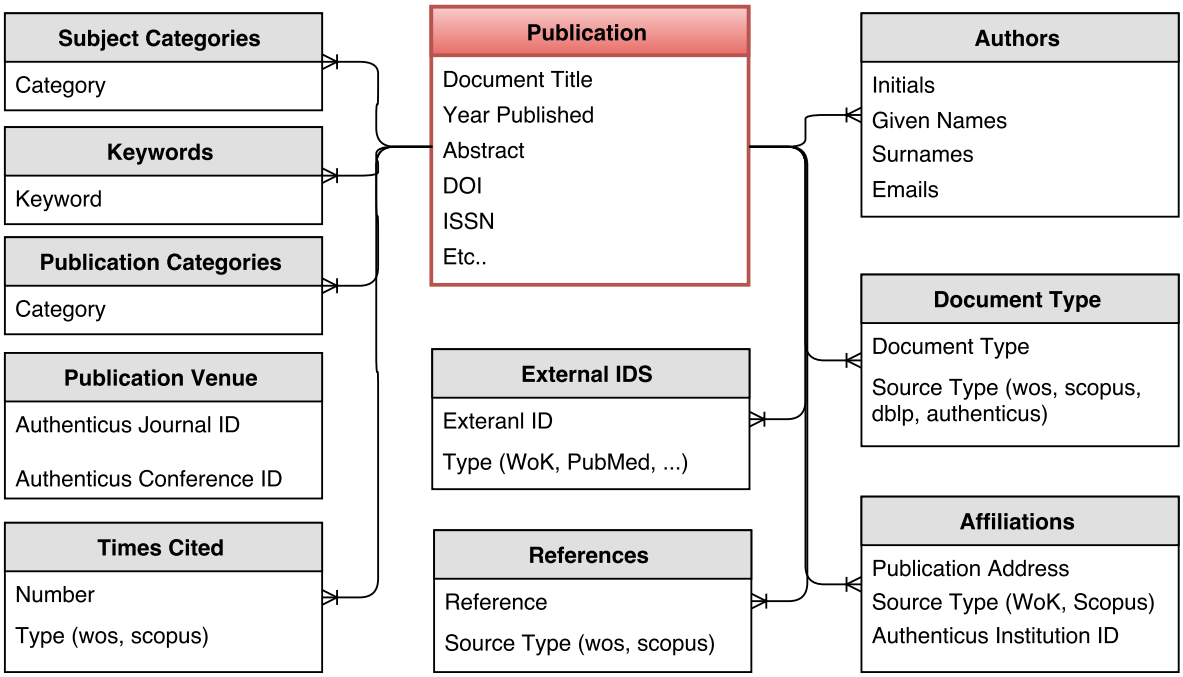


Figure 4.1: Partial view of the tables representing a publication record.

Each publication has a set of authors that are represented in the `Authors` table. For each author in the publication, a record is inserted in table with the publication primary key, as foreign key, the normalized author name, and an index indicating the order of the author in the publication. This is the representation of the publication metadata, and thus the author as represented in this table has has nothing to do with the representation of a researcher in the database. The identification algorithm will make the corresponding associations as evidence is built.

The entire publication module of Authenticus has a database structure with over 30 tables, connected each other with weak or strong constrains. Most of the tables are part of the APR structure, but there are many other auxiliary tables to support other functionalities of the publication module that are not. Examples are tables to keep duplicated publication records, tables with raw data imported from external sources, or tables/views used for statistical purposes.

Auxiliary Publication Modules

The Publication module is extended with additional modules to handle related complex publication management tasks. Those modules are:

- **Publication Profile Module**. This module is described in the Section 4.3.2 of this chapter. It serves to define and manage publication profiles required to control publication visualizations. In Authenticus there are different types of user groups, each of them with a different set permissions to the publication data. Within this module we are able to define to which publication data fields each user profile (or group) has access to.

- **Marked List Module**. Marked List is a custom list of publications manageable by users of Authenticus. Every user of Authenticus, while browsing Authenticus publications interfaces, can select one publication or a set of publications and add them to the temporary list *Marked List*. The link to the Marked List is a static element of the base Authenticus layout, thus accessible from every interface of the application. On the main Marked List interface, users can preview the list of publications they created, remove some records from the list or completely clean the list. Authenticated users of Authenticus have access to additional Marked List functionalities, such as pretty-print the list in HTML format, or pretty-print the list in PDF, or export the list into various supported formats, such as: BibTeX, EndNote, CSV, RIS or ISI.

- **Export Module**. This module is described in the Section 4.4.2 of this chapter. It is a set of methods to handle the export of an APR into various formats, such as: BibTeX, EndNote, CSV RIS or ISI. This module also serves to prepare print lists of publication in HTML and PDF format.

- **Publication Merge Module**. This module handles the merge of redundant Authenticus publication records. The complex structure of an APR makes the process of merging two publications a tricky task. This module has several interfaces, which facilitates the merge process. There are two ways of finding duplicates in Authenticus: (1) manual, detection made by users of Authenticus; (2) automatic, found by the Duplicates Detection Algorithm. In case of the manual merge, the user selects two publications, can preview those records and send a request to Authenticus editors to merge the two publications. Duplicates found automatically are listed in the researchers to validate lists. Users can access the list and for each pair of duplicate publications, they can decide if those are really duplicates, select which record to keep and which one to dismiss or decide about the preferred metadata source (in case there are more than one). The Duplicates Detection Algorithm is described in Sub-section 4.2.3 of this chapter.

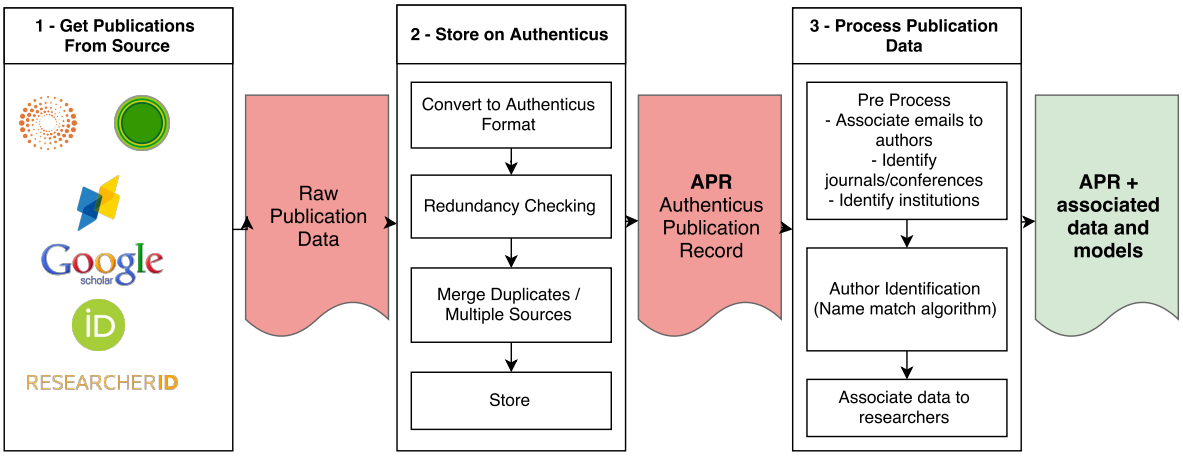


Figure 4.2: Authenticus publication creating process.

4.2 Creating new Authenticus Publications

To accomplish our goal of building a reliable and national repository of scientific publications, we split the entire process of handling publications data in three independent steps:

Step 1: importing publications metadata into Authenticus from multiple source repositories;

Step 2: conversion of original metadata to the Authenticus publication record; this includes redundancy verification based on which a new record is created or an existing one is updated;

Step 3: identification and association of the publication data with other modules of the system.

Figure 4.2 visualizes the whole process of creating a new publication in Authenticus. After each step of the process the new publication record is more complete and structured. The final product is a complete Authenticus publication record associated with other Authenticus entities, where authors are identified with researchers, venues are identified with journals and/or conferences, keywords and scientific fields are extracted and publication affiliations are connected with real institutions.

The following sections detail the three steps involved in creating a publication record.

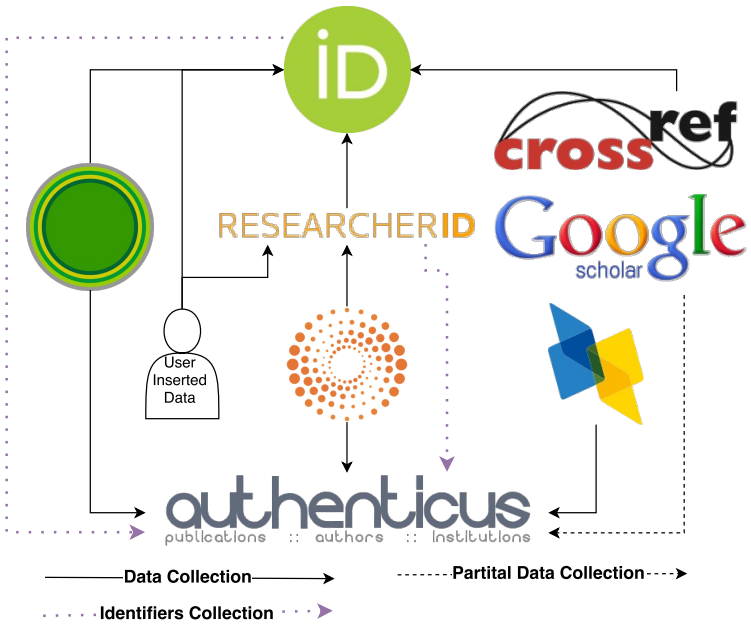


Figure 4.3: Authenticus metadata sources.

4.2.1 Importing Publications Metadata

Authenticus, as a publications metadata aggregator, can currently import publications metadata from three external sources: ISI Web of Science (WOS); Scopus; Google Scholar[1], and three other external aggregators: ORCID; DBLP; ResearcherID. Each of the metadata providers uses different APIs, thus the information retrieved from the other systems has a different structure, different data characterization and classification. Figure 4.3 shows the Authenticus metadata sources and the flow of the data/identifiers between the various publication databases and publications aggregators.

To be able to import metadata from various sources, we split such process into two steps:

Step 1: get publications from source: a group of generic helper classes which deal with the different systems APIs and retrieve metadata,

Step 2: store raw publication metadata: a group of actions that processe the retrieved metadata and stores the raw data in Authenticus database for further processing.

[1]Authenticus has aggregated citations from Google Scholar in the past, however due to Google's policy in blocking servers that make too many requests to their servers, we have suspended for now, this service.

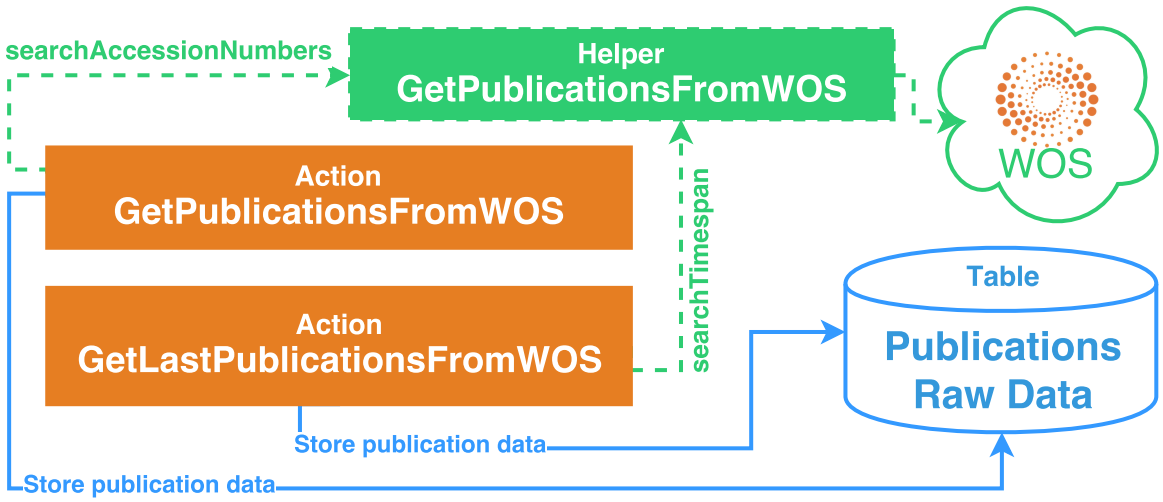


Figure 4.4: Import publications from WOS.

All the helpers and actions that deal with importing the metadata, have a similar name and arguments structure. The final output of importing publications metadata into Authenticus is a raw publication data stored in the Authenticus database. The publication raw data table contains data such as: metadata source, metadata source identifier or a creation and update date.

The following sections give technical details about importing publications metadata from different sources.

WOS - ISI Web of Science

To import publications from ISI Web of Science, Authenticus makes use of two APIs provided by Thomson Reuters [2]:

1. **Article Match Retrieval Service**[3] "allows for a real-time lookup of bibliographic metadata such as Digital Object Identifier, author, source title, etc., against the Web of Science™ Core Collection database using the institution's subscription entitlements. If a match is found, the service will return Times Cited information as well as links to view the full record, related records page, or citing articles page in Web of Science Core Collection."[12]

[2]Thomson Reuters: www.thomsonreuters.com
[3]Article Match Retrieval Service: http://wokinfo.com/directlinks/amrfaq/

2. **Web of Science Web Services**[4] available for institutions subscribed to Web of Science Core Collection. "Web Services allows for a real-time lookup of Web of Science Core Collection source record information against Web of Science Core Collection. (...) This service provides access to formatted, current information for enhancing the institution's repository, automatic, real time querying of multiple records and deliverability of real-time Web of Science Core Collection information(...)". [12]

Figure 4.4 illustrates the workflow of the import from ISI Web of Science. The helper used to get publications from ISI Web of Science has the name `GetPublicationsFromWOS`, which communicates with the APIs and makes searches using WOS queries. The helper consists of two extensions of the search methods: (1) `searchAccessionNumbers` that searches publications sources using WOS queries but only with the WOS Numbers (internal identification of publications at WOS), and (2) `searchTimespan` which searches publications using WOS query and a timestamp to limit the search results. All the methods return a dictionary of fields. There are two actions to process and store the raw publications data obtained from WOS: (1) `GetPublicationsFromWOS`, which calls the `searchAccessionNumbers` helper method, and (2) `GetLastPublicationsFromWOS` which calls `searchTimespan` method.

Scopus

Autenticus uses Elsevier Scopus APIs[5] and Elsevier ScienceDirect APIs[6]. The APIs give access to: "journals and books published by Elsevier on ScienceDirect full-text platform;" [5] and "citation data and abstracts from virtually all relevant scholarly journals, as indexed by Scopus, Elsevier's citation database." [5]. The full API access is free of charge and is only granted to clients that run within the networks of organizations with Sciencedirect and/or Scopus subscriptions.

Figure 4.5 illustrates the workflow of the Scopus publication metadata import procedure. The flow is very similar and the Scopus helper and actions hold the same structure as in the case of import from WOS. The main difference are classes names and the names of the methods to search in Scopus for publications using the internal Scopus publication identifier (EID) - `searchEIDs`. All the methods also return a dictionary of fields.

[4]Web of Science Web Services: http://wokinfo.com/products_tools/products/related/webservices/
[5]Elsevier Scopus APIs - http://dev.elsevier.com
[6]Elsevier ScienceDirect APIs - http://dev.elsevier.com/sd_apis.html

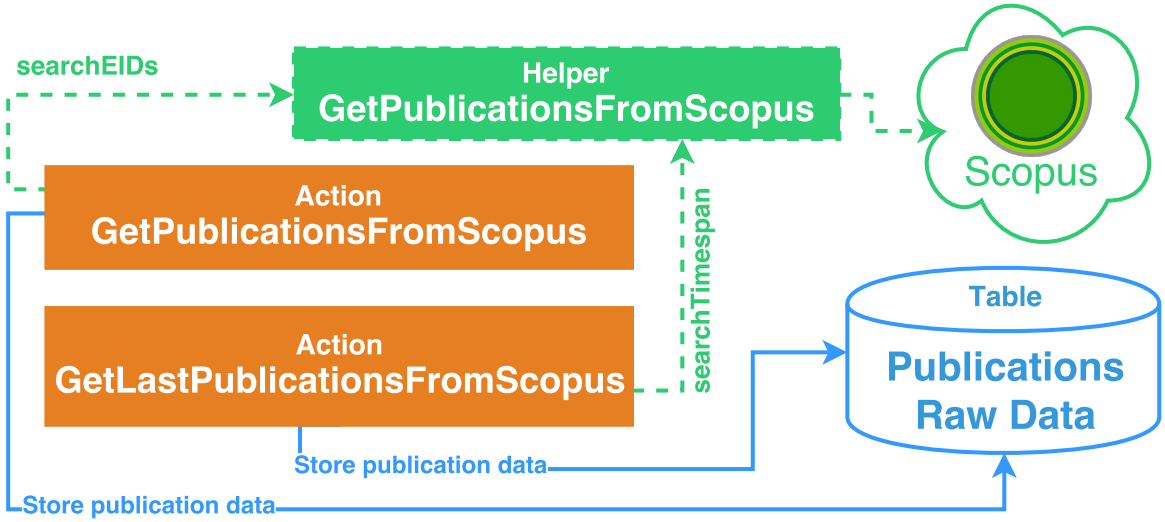


Figure 4.5: Import publications from Scopus.

DBLP

To import publications from DBLP [7] Authenticus use an XML-based API. The DBLP API is described in Michael Lay article - "DBLP — Some Lessons Learned" [8]. DBLP provides free access to high quality bibliographic metadata for the benefit of the international computer science research community. All of DBLP's data has been released under the Open Data Commons ODC-BY 1.0 license [8].

The DBLP API differs from the WOS and Scopus APIs, as it does not allow for specialised searches, for example, search for all publications of authors with affiliation Portugal. To obtain metadata from DBLP we need to know a priori the id of the researcher at DBLP, which can only be provided by the corresponding researcher. Once Authenticus researcher provides its DBPL-ID, we store it and take care so that all publications of that researcher are automatically updated, thus no researcher intervention is requierd. Publication list obtained from DBLP can be quite accurate, specially if the researcher has made correction requests whenever necessary. Nevertheless, the metadata of publications are not as contextually rich as in the case of metadata from WOS and Scopus. The only "problem" with DBLP is that the publications aggregated at DBLP are mostly limited to the area of Computer Science.

Figure 4.6 shows the workflow of obtaining publications from DBLP. The process takes

[7]DBLP - http://dblp.uni-trier.de/
[8]Open Data Commons ODC-BY 1.0 - http://opendatacommons.org/licenses/by/summary/

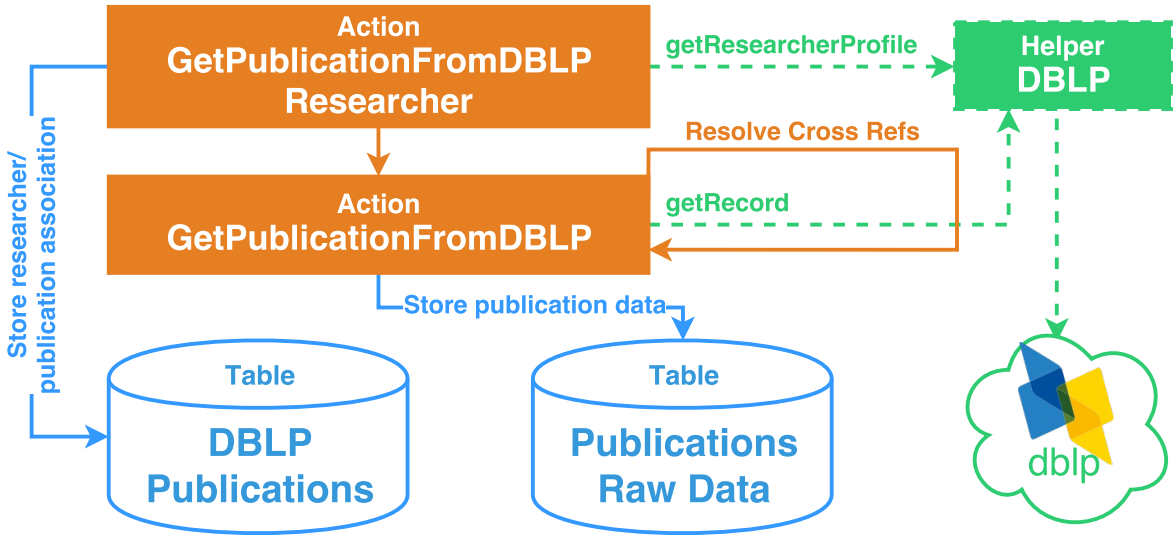


Figure 4.6: Import publications from DBLP.

two steps, first, for a given researcher, obtain from DBLP a list of all of its publications ids and store them. The action `GetPublicationsFromDblpResearcher` together with method `getResearcherProfile` from the helper `DBLP` perform this operation. In the second step (`GetPublicationFromDblp`), we process the list stored in the first step, and for each publication from this list we get the publication metadata from DBLP (method `getRecord`) and transform into an Authenticus publication raw data. Before retrieving the publication metadata we check if the publication already exists in Authenticus database, thus avoiding creating duplicates and limiting the number of XML requests to DBLP.

Other aggregators

Currently, Authenticus treats WOS, Scopus and DBLP metadata sources as the only valid and reliable sources. Data retrieved from those sources is very accurate and contextually rich - an important aspect for correct author name identification. WOS and Scopus as citation databases, have high bibliographic coverage, and index publications from many journals and conferences. Records are updated and citations counts are always updated and used worldwide for bibliometic purposes. DBLP as covering only one area - Computer Science, has much less bibliographic coverage, however, it is considered as one of the best in the area, keeps the data clean and of a high quality.

At this moment, we also import publications from other external systems, further called aggregators. Aggregators are not citation databases do not index journals and do not

create publication metadata, aggreators only aggregate lists of publications from other citation databases. Over the Internet there are many publication aggreators. Some of them are very well developed and have quite big bibliographic collections. Many researchers wordlwide use those systems to keep track of their scientific publications, thus some of them have quite complete coverage of researcher's work. An important aspect of publication aggregators is that many of them allow users to manually create publication records. Publications created manually are often full of errors and uncertainties and, what is most important, very often do not have coverage on any citation database.

Authenticus is also a publication aggregator, we import publications from other citation databases, but we do not create new untrusted Authenticus publication records. All publications at Authenticus have coverage in one of the three sources we trust, WOS, Scopus and DBLP, and only those publication records are displayed on Authenticus interfaces. However, as mentioned above, we also import publications from other aggregators, but we have a very strict policy of treating the data. To ensure data quality, we assume that on retrieving publications we only treat those publications, which have WOS, Scopus or DBLP external identifiers. If a publication has none of those three identifiers, we search for any information that allows us to get the metadata from the original source. Failing on identifying the original source, causes Authenticus not to keep the publication record in the database.

Next, we list two external systems/aggregators - ResearcherID and ORDCID, and describe how Authenticus processes data obtained from those systems. The last part we shortly describe integration of Authenticus with Google Scholar.

ResearcherID

ResearcherID [9] is an aggregator provided by Thomson Reuters [10]. A researcher after creating an account at ResearcherID website, has assigned a unique identifier and gets access to interfaces where he/she can manage hers publication lists, track times cited counts and h-index, identify potential collaborators etc. ResearcherID is fully integrated with Web of Science, publications can be added automatically from Web of Science, but also researcher can create new publications not indexed by any citation databases.

The process of importing publications from ResearcherID at Authenticus is shown

[9] ResearcherID - http://www.researcherid.com/
[10] Thomson Reuters: www.thomsonreuters.com

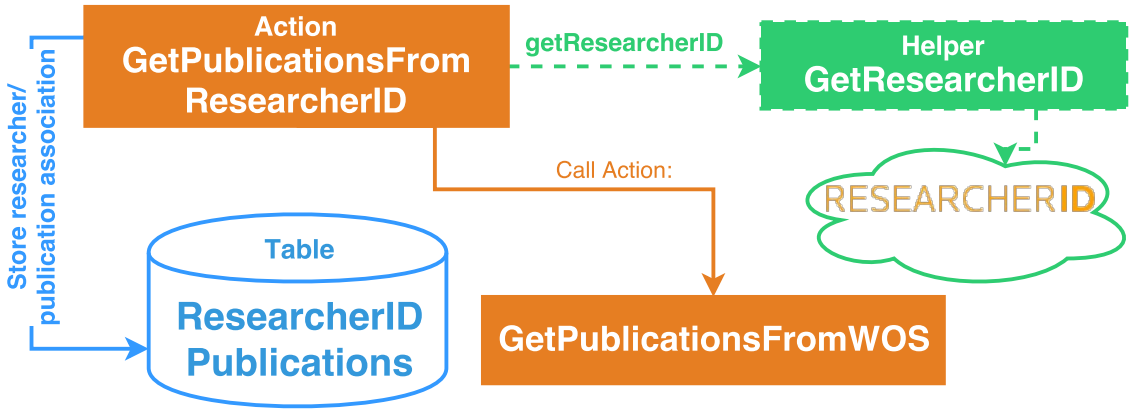


Figure 4.7: Import publications from ResearcherID.

in Figure 4.7. Similarly as in the case of the DBLP import, it also requires that Authenticus researcher provides its ResearcherID identification number. Knowing the researcher's ResearcherID, Authenticus first gets the publications list of a researcher from his ResearcherID website (action `GetPublicationsFromResearcherID`). Some publications of the retrieved list have associated WOS number, others, those added manually to ResearcherID profile, do not have any identification number. In the second step Authenticus gets the publications metadata, but only for publications that have a WOS number associated (action `GetPublicationsFromWOS`). After succeeding in importing the original publication metadata, Authenticus stores the publication record in the database.

This process is scheduled in Authenticus as a periodical background action, meaning that once a researcher provides his ResearcherID identification number, Authenticus periodically and automatically makes updates of publication lists from his ResearcherID website.

ORCID

ORCID[11] "is an open, non-profit, community-driven effort to create and maintain a registry of unique researcher identifiers and a transparent method of linking research activities and outputs to these identifiers." [11] ORCID aggregates publication records from many sources, such as Scopus, ResearcherID, LinkedIn or CrossRef and it provides an API that support system-to-system communication and authentication.

Authenticus is integrated with ORCID, in a way that it synchronizes Authenticus

[11]ORCID - http://orcid.org/

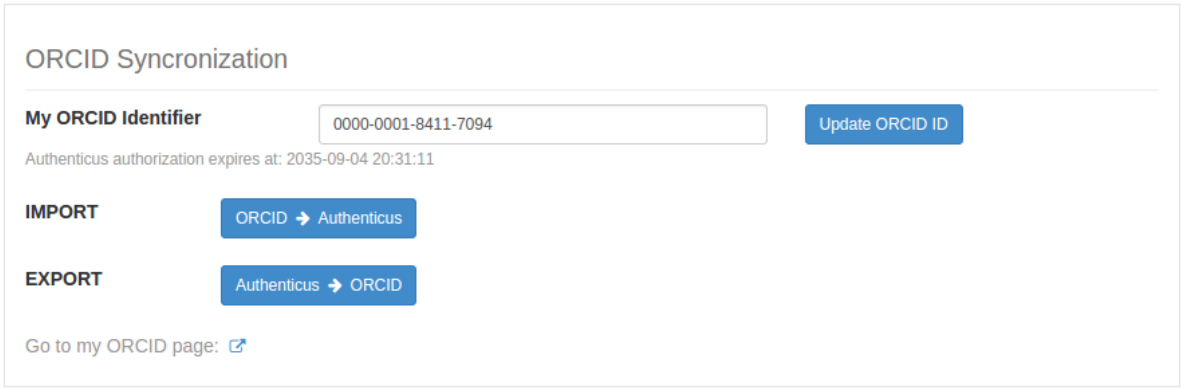


Figure 4.8: The Authenticus interface for ORCID synchronization.

researcher profile with ORCID researcher profile, imports researchers' list of publications from ORCID and exports a confirmed list of researcher' publications to ORCID. For Authenticus to be able to interact with ORCID records it needs to be a "Trusted Organization", an ORCID member organization to which an ORCID member gives the right to view, edit, and/or deposit specific data in its ORCID record. To facilitate Authenticus users to authorize Authenticus as a trusted organization at its ORCID record, we prepared a special interface dedicated for ORCID authorization/synchronization. The process has to be initiated by the Authenticus user within his researcher profile. Once the user provides his ORCID number, he/she is redirected to a page where he/she authorizes Authenticus as a trusted entity. Authenticus ORCID authorization is valid for 30 years, thus users have to grant the authorization only once.

The Authenticus interface of Figure 4.8 for ORCID synchronization with Authenticus consists of two actions: (1) "ORCID–>Authenticus" to import a researchers' list of publications from Orcid into Authenticus, and (2) "Authenticus–>ORCID" to export a confirmed list of the researcher' publications to ORCID. Those actions can be performed by the user at any time, and all the changes on the researchers publication list have to be confirmed by the user. Periodical background actions are executed to automatically update the publications list of all researchers by importing new publications metadata from ORCID ("ORCID–>Authenticus"), and for researchers that have granted Authenticus trust to update ORCID, whenever a publication in his list is validated it is automatically exported to ORCID if it is not there yet (i.e. performs "Authenticus–>ORCID").

The background action of importing publications from ORCID is illustrated in Figure 4.9. After retrieving a list of publications from ORCID (action `GetPublicationsFromOrcid` with the help of `GetOrcid`), we iterate over the list and

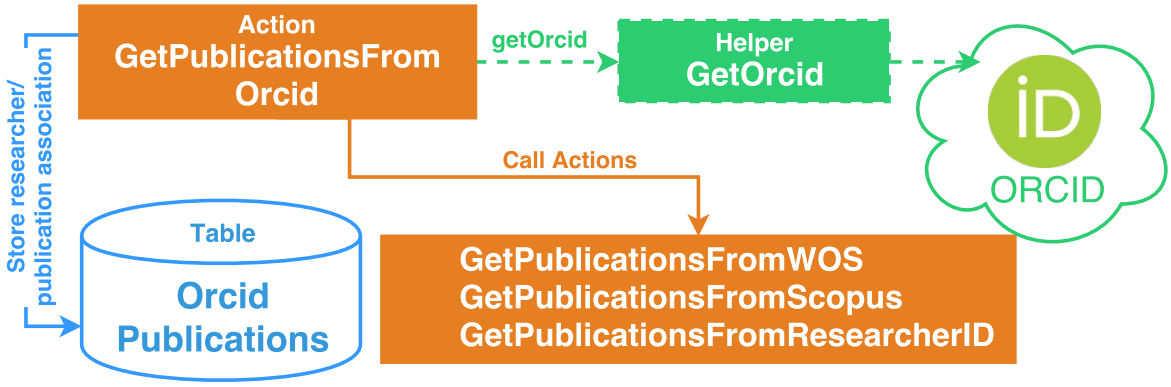


Figure 4.9: Import publications from ORCID.

analyse only those which have a valid source, meaning Scopus and Web of Science. The publication records retrieved from ORCID are in BibTeX format, which besides the basic publication metadata, contains a set of publication source URLs. If one publication has as a source URL an `eid` parameter, it means that it is indexed by Scopus and the value of that parameter is the publication' Scopus identification number. If the publication BibTeX id has a pattern `RID:AA-BB` it means that publication is indexed by Web of Science, where `AA` is an index that allows us to get the ResearcherID and `BB` is the index of the publication on `AA` profile. In case that some publication record retrieved from ORCID does not already exist in Authenticus, we call `GetPublicationsFromWOS`, `GetPublicationsFromResearcherID`, or `GetPublicationsFromScopus` to retrieve the publication metadata.

Google Scholar

Google Scholar[12] is a "freely accessible web search engine that indexes the full text or metadata of scholarly literature across an array of publishing formats and disciplines" [15].

At Authenticus, the Google Scholar import are the least developed. We have implemented support to allow Authenticus users to associate their researcher profile with the Google Scholar identification number. This enables Authenticus to retrieve the publications list from the researcher's Google Scholar profile and only store in Authenticus publications which are from trusted sources, WOS, Scopus or DBLP. Due to the constant changes of the Google Scholar profiles and lack of stable Google Scholar API, or scraper to retrieve publications metadata, the integration of Authenticus with

[12]Google Scholar - https://scholar.google.pt/

Google Scholar is very poor. Although we had this functionality available to our users, we decided to suspend it temporarily until a solution is found that can surpass the limitation imposed by Google on leaving our server in quarantine for 24 hours if the number of requests to their servers surpassed a certain limit.

In the near future, we consider the possibility to allow users to create manually publication records in Authenticus. This is important to accommodate areas such as the social sciences that are very much under-represented in the sources we currently use. This, however, requires a strict policy to be defined and a validation mechanism to be in place to assure a minimum data quality for it to be useful.

4.2.2 Creating Authenticus Publication Record

The second step after importing publication metadata from multiple sources and storing it as a raw publication data, is the creation of, or update of an existing, APR.

Every source of publication metadata represents one publication in a different way. The sources differ on their formats, type of fields, or schemas of the data. To facilitate our work of transforming raw publication data into Authenticus Publication Record we carefully devise a representation, specific to Authenticus, that is maximal in the representation of the data originating from the multiple sources. To accomplish such a task we created an helper `RawPublicationToAuthenticus` class that takes a raw publication data and converts into the Authenticus Publication Record format. In table 4.1 we show an example of the conversion of the authors field from various sources into Authenticus format.

Before storing the publication record in the database, we verify if the imported publication is redundant, in other words, if it already exists in the Authenticus database. This is done by the"Duplicates Detection Algorithm", which searches for potential duplicates and determines a duplicate score between the current publication and the potential duplicate candidates. This algorithm is detailed in the next section.

If the publication is found to be redundant, that is a duplicate is found with high certainty, then a call to the helper `MergePublications` is issued. It takes multiple raw publications and produces a unique publication with the metadata of different sources. We can parametrize the merge helper to give more priority to some sources. If the "Duplicates Detection Algorithm" can not make a decision, for example it detects more than on duplicate, or the duplicate detected is the same source, it does not update the

Authors	**Authors Full Name**
Marzulo, LAJ; Alves, TAO; Franca, FMG; Costa, VS	Marzulo, Leandro A. J.; Alves, Tiago A. O.; Franca, Felipe M. G.; Costa, Vitor Santos

(a) WOS

Authors
Marzulo L.A.J., Alves T.A.O., Franca F.M.G., Costa V.S.

(b) Scopus

Author
Leandro A. J. Marzulo
Tiago A. O. Alves
Felipe M. G. Franca
Vítor Santos Costa

(c) DBLP

Initials	**Given Names**	**Surnames**
L; A; J	Leandro	Marzulo
T; A; O	Tiago	Alves
F; M; G	Felipe	Franca
V; S	Vítor; Santos	Costa

(d) Authenticus

Table 4.1: Author's name representations in the various sources.

existing publication but instead creates a new one and registers the possible duplicates for future manual decision. If it does not find any duplicate candidate, a new publication is created.

Figure 4.10 illustrates the process of creating/updating new Authenticus Publication Record. It is implemented in the action `InsertPublication($raw_publication_id)`.

4.2.3 Duplicates Detection Algorithm

To keep the database of publications clean and of high quality, we want to assure that on importing new publications from different sources we do not create redundant publication records. To check if one publication already exists in Authenticus database we designed a "Duplicates Detection Algorithm". The first part of the algorithm searches for potential duplicates of the current publication. It is done using MySQL `MATCH()` [13] function which performs a natural language search for a string (in our case title) against a text collections (all titles of publications in Authenticus).

In the second step, the algorithm shown in Figure 4.11 is used by Authenticus to

[13]MYSQL: Natural Language Full-Text Searches - https://dev.mysql.com/doc/refman/5.6/en/fulltext-natural-language.html

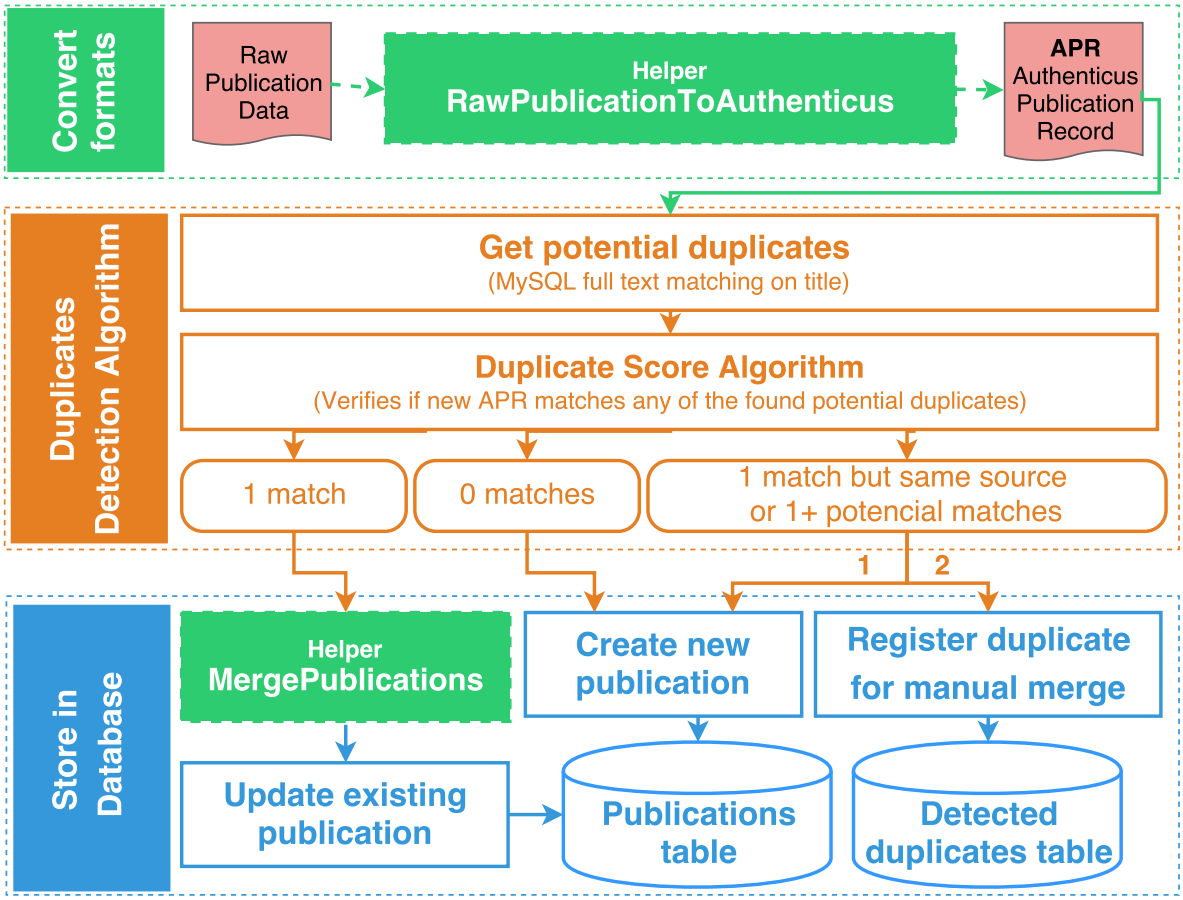


Figure 4.10: Authenticus process of creating/updating new Authenticus Publication Record.

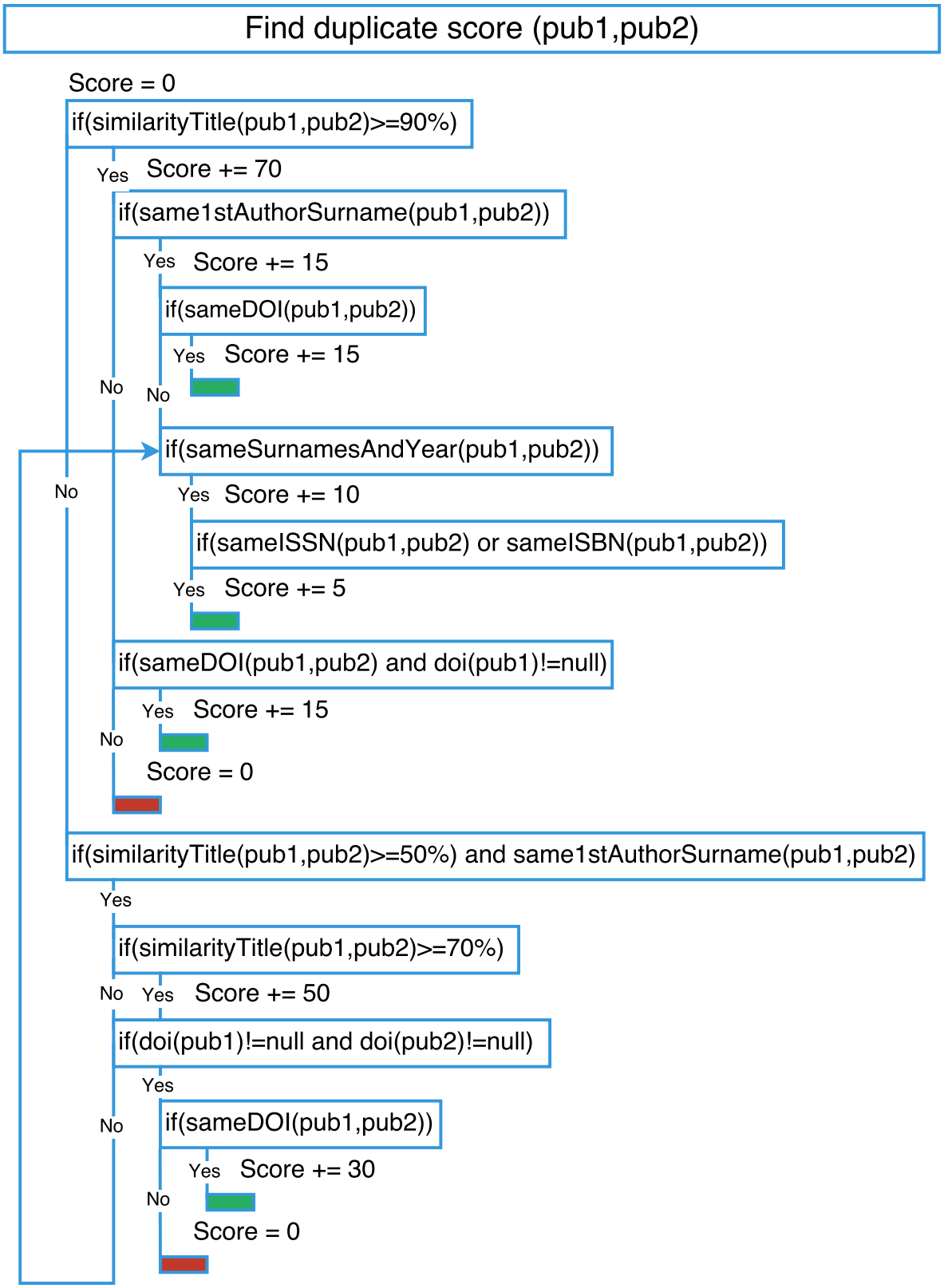


Figure 4.11: Authenticus algorithm to determine the duplicate score between two publications.

determine the duplicate (or similarity) score between the current publication and the potential candidates. For two given publications, say `pub1` and `pub2`, if the similarity score is greater or equal to 100, and the publications are from different sources, we consider them the same, and automatically activate the merge publications action that joins the metadata of both publications in just one record. Otherwise, these publications will exist, each with its own record, and will be listed in the duplicates list for the researcher to confirm, or not, whether they should be merged. For a publication being imported, we do an approximate match based on the title and select the 10 best matches. Each of these, will be tested against the publication being imported using the algorithm described in the figure. The algorithm starts by evaluating the similarity of the titles among the two publications. If similarity is above 90%, then a minimum score of 70 is ensured and further tests are performed, namely, if the surname of the first author in both publications is the same, another 15 points are added, if the DOI in both publications is the same another 15 points are added to the score and we stop with a score of 100. If DOI is not the same and not null, then we validate all the surnames of all authors in the same order. If those are confirmed, 10 points are added and then we check ISSN or ISBN for further validation. If the similarity of the title is less than 90% but higher than 50%, then we take further validations for a lower score but with some chance of still being a duplicate. The year must be the same, venue/conference must be the same, and the DOI can not be different, and the number of other fields that are equal on both publications must be the same. To check if two venues are the same the system check ISSN/ISBN. Cases in which the score is zero, are automatically dismissed and considered different.

Some duplicate scores may be low, but it will capture cases in which the title in one source is both in English and Portuguese, and in another source is only in one of the languages. We do not gain many points from the title, but still gain evidence from the other field comparison tests.

Performance impact of the duplicate detection algorithm

Duplicates detection is an essential part of the publications import process. Without it, the Authenticus database would be filled with many duplicate publications making it very difficult to report duplicates to researchers and also making it impossible to calculate reliable bibliometric indicators. On the other hand, by being able to identify duplicates, we can more easily compare how well different aggregators index the publications and count citations. This is however an operation that has to be done frequently

and thus needs to be rather efficient.

Table 4.2 provides some statistics regarding the main operations of the import process. We tested with 6366 new publications being added to the database and of these, 2593 were found to be unique, that is new, 2706 were considered potentially duplicated, that is the duplicate score is greater than zero, and 1067 were considered duplicates (score greater or equal 100) and were automatically merged. Detecting if a given publication is a duplicate or not, that is applying our algorithm is rather fast as it takes less than 0.3s on average. The small differences observed with new, duplicates and merged are mostly due to discrepancies that there may be in the publications (publications with more authors do usually take longer to process). In the case of the duplicates, often the algorithm terminates early and thus the smaller time. The creation time is really very low, less than 0.06s. Creating a merged publication takes longer, as signaled in the table by (*), as one have to first disassociate already associated authors to some publications (accounts for 0.12s) being merged and then create a new record with new associations. This is done to ensure that the new publication and corresponding associations of authors to researchers appear correctly. The last line of the table shows the total time for the operation, and we see that even in the case were a merge is performed we take less than half a second to do this processing.

	New	**Duplicated**	**Merged**	**Total**
Number of samples	2593	2706	1067	6366
Detect duplicates time (s)	0.29	0.26	0.30	0.28
Create/update time (s)	0.03	0.04	0.18 (*)	0.06
Total time (s)	0.32	0.30	0.48	0.34

Table 4.2: Impact of the duplicates detection algorithm.

4.2.4 Associate Authenticus Publication Record with other models

As some fields in the publication metadata do not have a standard way to be written (such as the journal, conference name, association between the authors and their email and publication addresses), we need to pre-process and normalise them. For that we use a sub-module of the Authenticus Name Identification Algorithm that will associate:

- publications to one of the journals or conferences already in Authenticus database;

- emails addresses of the publication to the respective authors;
- publications to the institutions that can be derived from the email and the publication addresses, and if possible associate each institution to an author.

With the pre-processing done, we are ready to run the Name Identification Algorithm and associate publications to researchers.

Whenever the Authenticus Name Identification Algorithm, or some allowed user, identify one publication author with a researcher, we have to save that association and update the profile of the corresponding researcher. The process is rather complex as we need to:

1. check if the author is already associated to some researcher; if it is, and is not validated, we need first to undo the association and all related data;
2. check if the researcher is already associated with another author of the same publication; if it is, undo the association;
3. associate the author to researcher, for it appear in the publication list;
4. associate the information of the author and the publication to the researcher's profile, as:
    - researcher name variants: author name, full name, emails;
    - keywords;
    - subject categories;
    - science fields;
    - co-authors;
    - institutions;
    - venues;

The action that does this, associate and disassociate a publication author to a researcher, is `AssociatesPublicationToResearcher` (Figure 4.12) that execute `IdentifyPublicationAuthor` to associate the author to the researcher, and execute `AssociatesPublicationDataToResearcher` to associate the data to the researcher profile.

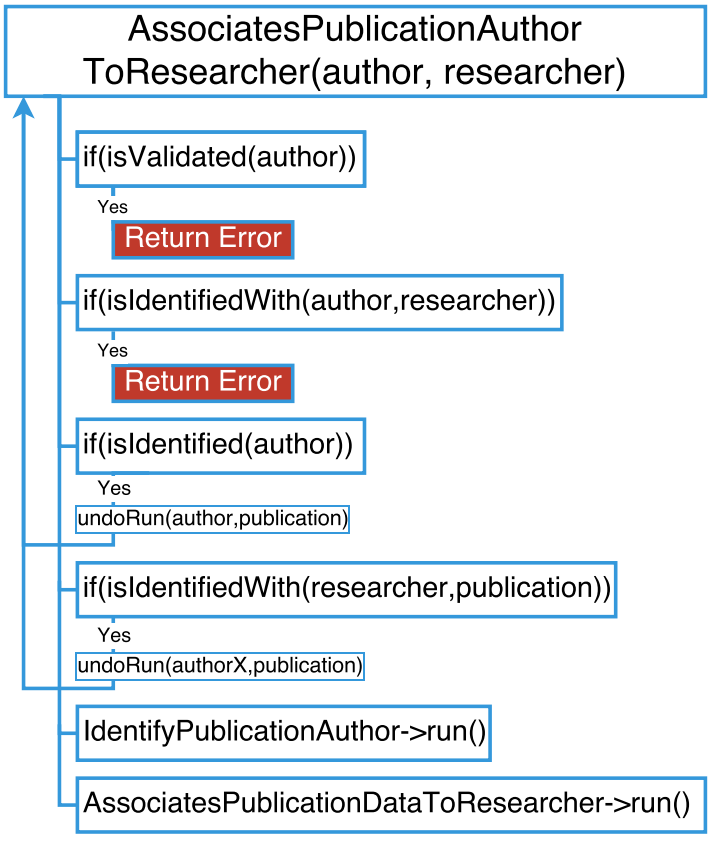


Figure 4.12: Action AssociatesPublicationToResearcher

Since the pre-processing could take quite some time to run, as it highly depends on the number of authors and addresses in the publication, we split the action `AssociatesPublicationDataToResearcher` in two steps, one that can be run before the pre-processing, and another that will run after the pre-processing. Thus, we could associate almost all the data to the researcher as soon as the publication is inserted, and when the pre-processing finishes the remainder is associated.

4.3 Publication Interfaces and Profiles

This section focuses on an aspect of publication visualization at Authenticus. We first define what types of publication interfaces exists on Authenticus. The later part of this section explains what is a publication profile, how it is defined and managed.

4.3.1 Publication Interfaces

Authenticus has several interfaces to visualize publications and manage functionalities related with the publication module. The following lists and shortly describes the most important publications interfaces.

Publication search and search results

Publications search page is where an user enters terms and starts a search for publications. It is possible to define in which field of a publication to search. The available fields are: title, author, year and publication Id, where publication Id can be one of the following identifiers: AuthentiucsID, DOI, PUBMED, WOS, SCOPUS or DBLP identifier. Search can be simple, using only one criteria or complex where the user can choose one criteria for each field. The search results page is where search results are returned to an end user. The window is divided in two parts: right with the list of publications returned as a result of the search and left with the filters to refine the search. Every publication in the returned list of results is a short publication description controlled with the publication profile defined for the current user. The filters to refine search have confirmed search options and additional filters retrieved from the returned result. The additional filters and data retrieved from the result are also controlled with the user' publication profile.

Publication record view

This is the page with the details about a publication. The amount of information displayed on the publication record depends on the user profile group the user is assigned to. Guest users, general public and users without b-On license can see only basic data, users with b-On license have access to the full record. Publication record view has additional options such as: export to various formats and update citations counts. Authenticated users associated with Authenticus researcher, every time they view a publication not belonging to their official list of publications, can add a currently viewed publication to their profile by clicking the button 'Add to My Publications'.

Publications temporary list

A "Marked List" is a temporary custom list of publications managed by users of Authenticus. Every user of Authenticus, while browsing Authenticus publications interfaces, can select one publication or a set of publications and add them to the temporary list, Marked List. On the main Marked List interface, users can preview the list of publications they created, remove some records from the list or completely clean the list. Authenticated users have access to additional Marked List functionalities, such as print list to HTML format, print list to PDF or export list into various formats, such as: BibTeX, EndNote, CSV RIS or ISI.

Publications statistics

This page shows the basic statistics about Authenticus publications collection. It is public available and updated daily. At this moment it shows three charts: "Publication metadata sources" which shows percentage of publications in Authenticus with metadata from a given source, "Number of Publications in Authenticus" - a timeline presenting the growth of publications in Authenticus over time. The last chart/table is called "Publications in numbers" contain the number of publications by source and how they overlap within the various sources.

Publication source and merge preview

This is an interface available for administrators and editors of Authenticus. It presents the source composition of one Authenticus publication record, meaning, user can view

Figure 4.13: Interface for merging two redundant publications.

in which sources publication is indexed, the raw source metadata and what is the final result of the merge of raw metadata.

Publications duplicates

This is a page dedicated to handle redundant publication records. Users can manually select two publications to be merged and send a merge request to Authenticus administrators/editors. After the request is reviewed, users receive an email notification with the merge decision. There is an interface that lists potential duplicates of publications detected by the "Duplicates Detection Algorithm". Figure 4.13 shows an example of an interface for merging two redundant publications.

4.3.2 Publication Profiles

In Authenticus, an access to publication's metadata is limited by the b-ON license. According to this license only institutions that hold the b-ON license have access to the full publication record. To make Authenticus available for any kind of users, we define publication profiles. Publication Profiles are different configurations of metadata of publication, where some configuration contains all publication metadata, other contain only basic, publicly available publication metadata. An Authenticus module responsible for defining and managing publication profiles is called "Publication Profile Module". Profiles can be easily assigned to various groups of users or to specific functionalities that handle publication data.

Authenticus defines three standard publication profiles:

- **Admin** - it is available for all administrators of Authenticus. The Admin profile have access to all information that Authenticus has about publication records.

- **Full** - it has all information about publication that can be exposed to users of Authenticus. Full profile is assigned to researchers and institutions that belong to a group with a valid b-ON license.

- **Short** - it is assigned to general public and users from a group without valid b-ON license. The Short profile contains only basic data about publications, not more than what is publicly available all over the Internet.

Administrators of Authenticus can create additional publication profiles to be used for special requests or functionalities. The three standard publication profiles are presented in the Appendix C. This is the current representation of the main profiles, data is divided into two tables, one representing only data about publication, other showing data retrieved from the relations publication has with other Authenticus models.

4.4 Interoperability

For Authenticus to communicate and exchange data with end users, external systems and software applications, we deployed a few modules and data schemes which allow us to manage and control Authenticus data interoperability. This section describe the basic Authenticus publication XML scheme, publication export module and in the last part it details the Authenticus Publications Retrieval Web Service.

4.4.1 Authenticus Publication Record XML Schema

Authenticus XML Publication Record schema defines the structure and constraints about the Authenticus publication. It is a base XML schema used in XML transformations to produce other publication export formats and to send publication records to an external system using the Authenticus Publications Retrieval Web Service. Appendix D contains the complete XML schema. It has the following main elements: publication_data, authors, editors, journal, conference, affiliations, keywords, categories, citations_history, times_cited, identifiers and publication_sources.

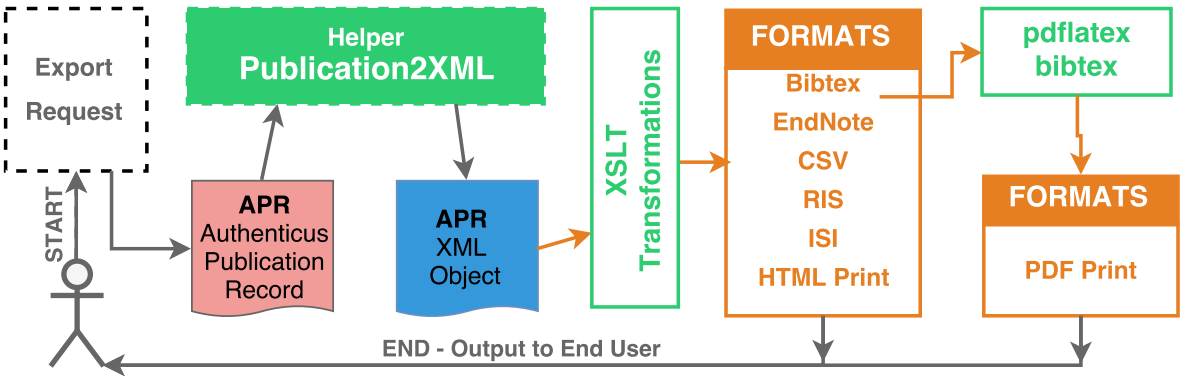


Figure 4.14: Work flow of an export of one publication for a single end user request.

4.4.2 Publication Export Module

This is a set of methods to handle export of an Authenticus publication record into multiple formats, such as: BibTeX, EndNote, CSV RIS or ISI and to prepare print lists of publication in HTML and PDF format. Figure 4.14 presents the workflow of an export of one publication for a single end user request. One Authenticus publication record (APR) is transformed into an APR XML object using helper `Publication2XML`. The new XML document is validated using the XML scheme described in the previous section. To produce a document in a format requested by an user, we use a XSLT transformation, which transforms XML documents into other XML documents, or into other formats such as HTML, plain text, or XSL Formatting Objects. To export a list of publications into PDF, we use `pdflatex`, a default LaTeXengine which outputs text directly in PDF file and `BibTeX` a reference management software for formatting lists of references. With a list of publications we first create a list of references in BibTeX format (example: file.bib) and then run a script to transform the list of BibTeX references into PDF. This is a typical procedure used while writing documents in LaTeX.

One of the issues we had to handle while developing the export module, was the mapping of Authenticus publication types into types allowed by all export formats.

Table 1 in Appendix B presents publication document type mapping on Export. For exports into CSV, PDF and HTML formats we do not map the publication document type but keep the Authenticus document type assigned while importing publication metadata. For the other exports, ENDNOTE, ISI, BibTeX and RIS the document type mapping is as presented in the Table 2 in Appendix B.

4.4.3 Authenticus Publications Retrieval Web Service

Authenticus Publications Retrieval Web Service APR-WS is an automated service that supplies publications metadata from the Authenticus database. It allows an external service to interact with Authenticus in a way that they can use Authenticus as their main source of publications per researcher.

At this moment there are two versions of the web service, one service which is deprecated, however still in production but used only by the University of Porto Information Services (SIGARRA), and a second service which is under intensive development, ready to be released in the nearest future. In this dissertation, we will focus on the second, new version of the web service and briefly describe its main features, syntax of the requests and responses and present a few examples.

Interfaces and Authentication

Authenticus Publications Retrieval Web Service is available to institutions that register to access it. Institutions with b-On license, can retrieve publications metadata available within the full publication profile. Institutions without b-On license have access to the metadata defined within the short or public publication profile only. Full and Short publication metadata profiles were described in Section 4.3.2.

Institutions wanting to have access to the APR-WS interfaces must send an email to `support@authenticus.pt`, specifying the public institution IP address that will be used and the interface needed. Authenticus team verifies if the institution can be granted access. Successful validation grants an API-key that is associated with the IP address of the institution and gives access to the full or short publication metadata. The API-key must be present in all service requests.

APR-WS Basic requeset and response

Request should be made preferably through POST HTTP, where Content-Type is Application/XML. Example:

```
POST https://authenticus.pt/ws HTTP/1.1
Host: authenticus.pt
Content-type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
```

```
<authenticus></authenticus>
```

The basic request structure is:

```
<?xml version="1.0" encoding="UTF-8"?>
<authenticus>
   <request>
       <api_key>API-KEY</api_key>
       <action>
           <name>ACTION-NAME</name>
           <version></version>
           <options>
               <!-- changes with the action/optional -->
           </options>
       </action>
   </request>
   <data>
       <!-- changes with the action/optional -->
   </data>
</authenticus>
```

Depending on the web service actions/operation, the values of `options` and `data` are filled. All the received requests are validated by an XML schema, which can be found here: `https://www.authenticus.pt/webservice/`.

The basic response structure is:

```
<?xml version="1.0" encoding="UTF-8"?>
<authenticus>
    <request_info><!-- the request that returns the following success/
       error -->
        <api_key>API-KEY</api_key>
        <action>
            <name>ACTION-NAME</name>
            <version></version>
            <options>
                <!-- changes with the action/may not appear -->
            </options>
        </action>
    </request_info>
    <success><!-- appears if no error -->
        <message></message>
        <results>
            <!-- changes with the action/may not appear -->
        </results>
```

```
    </success>
    <warning><!-- may appear along with success, useful information
       for the system administrator. E.g. Deprecated action, will be
       removed October 22.  -->
        <message></message>
    </warning>
    <error><!-- appears if the action does not complete successfully
       -->
        <code></code>
        <message></message>
    </error>
</authenticus>
```

The field `request_info` contains the parameters of the request made. They may not be exactly the same as the parameters used to make the request. They may differ in the number of the records received per page. If one client makes a request for 1000 results per page (`page_size` = 1000) and the specific client has an imposed limitation to receive at most 100 results per page, thus in the `request_info`, the field `page_size` will be 100 instead of 1000.

If a request is successfully validated, the response will contain a success tag with two extra parameters: message containing information related with the executed action, and results containing the returned data. The success tag may also contain extra information relevant for system administrators. If the request fails the validation, the response contains an error tag, with two extra parameters: code and message. The error code returned can be:

0. internal error
1. no data for the request
2. missing permissions
3. researcher not found

Warnings may appear in a response to raise awareness for a future event. If the request does not validate with the schema definition, the response returns without a `request_info` tag.

Currently, the external service may request two types of actions/operation:
(1) `GetUpdatedPublications`
(2) `GetResearchersWithPublicationsToConfirm`.

GetUpdatedPublications

The `GetUpdatedPublications` action request retrieves a list of publications of one or all researchers associated with a given institution. It is a periodic or on-request service of the APR-WS. All requests of this action face a limit on the number of the publications that are `page_size` returned. The default limit is 50 records, however institutions may request for this limit to be increased.

Requests may include filters, such as:

- `institution` - limits results to a specific institution/sub-institution.
- `researcher_institution_id` - limits results to one specific researcher associated with the institution making the request. If this filter is empty, all publications of all researchers of that institution will be returned.
- `page_timestamp` - Indicates the last time in which publications were retrieved, or provides a date since when publications will be retrieved. For the first request of the institution, the page time stamp is '0'. Each response returns a `next_page_timestamp`, corresponding to the time of the most recently updated publication that is included. A response provides just one page size of requested publications. Since a request may originate a response with a number of publications higher than the fixed page size, each response includes a remaining variable to indicate how many publications remain to be sent. It is up to the client to make the necessary requests to get the remaining publications.

A request may define additional publication fields which are not part of the short or full publication profile. Authenticus only returns these fields if institutions have the right permissions to access this information. Those fields are:

- **isi_fields** - WOS Publication categories.
- **subject_categories** - Subject Categories of the journal from the ISI JCR.
- **isi_fields** - WOS Publication categories.
- **isi_info** - List of parameters specific for ISI publication (isi.isi_id, isi.isi_publication_url, isi.times_cited, isi.times_cited_last_update, isi.isi_citations_url)
- **scopus_info** - List of parameters specific for Scopus publication (scopus.scopus_id, scopus.scopus_publication_url, scopus.times_cited, scopus.times_cited_last_update)

- **journal_id** - The Authenticus ID of the journal associated with the publication.
- **researcher_id** - Field related with publication author. Specifies what is the local institutional identifier of a researcher identifying the publication author.

A response to this action, besides the standard `request_info` tag with a repetition of request and a `status` tag containing also a `result` tag, defines info about:

- `returned` - Number of publication records returned. Maximum number of returned records is equal to the page size (page_size) defined in the request.
- `remaining` - Number of the remaining results to obtain.
- `next_page_timestamp` -Time stamp to be used to retrieve the remaining number of publications. This value should be used in the following request in the field `page_timestamp`.
- **publications** -The set of publications, which may have the following options:
  - `publication_deleted` – it only occurs when a publication that was previously associated with a researcher or an institution is no longer associated with it. For example, if a publication that was associated to a researcher of the requesting institution is later deleted from the list of publications of that researcher in the Authenticus database, then the Authenticus Web Service sends information specifying the ID of the deleted publication (`<publication_deleted>1234567</publication_deleted>`). `publication` – standard publication in a short or full profile.

`GetResearchersWithPublicationsToConfirm`

`GetResearchersWithPublicationsToConfirm` is a periodic or on-request web service action of Authenticus to retrieve a set of pairs, researcher id and number of publications to validate.

The specification of a request and response of this action is very similar to the previous `GetUpdatedPublications` action. The request has a limit on the number of pairs returned (`page_size`), has filters and it is also possible to request extra parameters/data.

The tag `results` in the response of this action contains the same elements as `GetUpdatedPublications` action, namely: `returned`, `remaining`, `next_page_timestamp`

and the last one `researchers` which contains a set of researchers data. For each returned researcher the following fields are returned:

- `authenticus_id` - AuthID, Researcher Authenticus ID.
- `number_of_new_publications` - Number of publications of the researcher to validate.
- `institution_id` – Optional field. Local institutional identifier of the researcher.

In the near future, we expect to improve Authenticus services to become more interoperable with other CRIS systems of the PT-CRIS [9].

In this chapter we described the Publications module of Authenticus. It is a main module as it is responsible for the importing the publications metadata from the multiple sources from which Authenticus currently imports publications, preprocessing and storing such data, verifying for redundancy, merging with other sources for the same publication metada, if they exist, thus creating the Authenticus publication record. At last, the module invokes the identification algorithm to assign the authors of the publication to reasearchers and the publication to institutions, and thus updating related indicators.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Authenticus is a Web based application designed and developed to become a national aggregator of metadata of scientific publications produced by researchers associated with Portuguese institutions. It includes a specialized name identification algorithm to automatically associate publication authors to known reaseachers from institutions and research units. It has been designed and implemented envisioning its efficiency and scalability to support a large number of concurrent users, while being dynamically updated with publications from known existing indexing sources. As we saw in the previous chapters, this dissertation focused precisely on the design and implementation decisions at the architecture level and mechanisms that were necessary to achieve the goals just enumerated.

A key contribution of this work was the core component of the Action Center. It included a flexible SQL Query Builder to simplify query construction, prevent SQL injections by the use of prepared statements, and to allow caching of recurrent queries. It implements, controls, monitors and logs action's execution for error or logging purposes. An action allows for database isolation, while at same time ensures strictness control of an operation implementation and its reverse.

Other important contributions detailed, particularly in Chapter 4, included an algorithm to detect redundant or duplicated publications, mechanisms to automatically and periodically import publications from known existing sources such as ISI, Scopus, and DBLP, interfaces to display information about publications with different view

constraints based on role based access rules, Web services for publications retrieval from Authenticus, provides a publication export module that produces XML as an intermediate format and from which many others can be derived. The integration of Authenticus with other aggregators such as ResearcherID, ORCID or Google Schoolar was also included.

5.2 Future Work

As with any research and development work, there are still many other functionalities left to be achieved. Here we enumerate a few that are in our short list but that could not be accomplished within the time frame of this dissertation:

- Accomplish the integration of other aggregators, specially Crossref[1], but also recover Google Scholar by finding a way to overcome the limitation that Google imposes on server requests.
- Terminate the development of the Institutions module as it is a crucial module to get institutions such as Universities and Research Unities using directly the Authenticus for reporting purposes.
- Develop and improve all the statistics and bibliometrics modules directed for the researcher and for the institutions.
- Develop non standard metrics that can be made available to researchers to foster cooperation or, sometimes, to simplify discovering who is working in similar topics as oneself.
- Improve integration with the PT-CRIS and CRIS (Current Research Information System) systems in cooperation with FCCN and FCT.

Other examples of future work could be raised. These are raised here only to illustrate that although Authenticus is already operational and being used it is still in development and there are still many functionalities and improvements that it can include.

[1]This is work currently underway and should go into production before the end of 2015.

References

[1] Sylwia Teresa Bugla. Name identification in scientific publications. Master's thesis, Faculty of Science, University of Porto, Portugal, 2009.

[2] CakePHP. Associations: Linking Models Together - CakePHP Cookbook. http://book.cakephp.org/2.0/en/models/associations-linking-models-together.html#creating-and-destroying-associations-on-the-fly. (Last visited 24/9/2015).

[3] CakePHP. CakePHP - the rapid development php framework. http://cakephp.org. (Last visited 24/9/2015).

[4] CakePHP. Containable - cakephp cookbook. http://book.cakephp.org/2.0/en/core-libraries/behaviors/containable.html. (Last visited 24/9/2015).

[5] Elsevier. Elsevier Developers. http://dev.elsevier.com/. (Last visited 12/10/2015).

[6] Martin Fowler. FluentInterface. http://www.martinfowler.com/bliki/FluentInterface.html, December 2005. (Last visited 24/9/2015).

[7] Theo Haerder and Andreas Reuter. Principles of Transaction-oriented Database Recovery. *ACM Comput. Surv.*, 15(4):287–317, December 1983.

[8] Michael Ley. DBLP: Some Lessons Learned. *Proc. VLDB Endow.*, 2(2):1493–1500, August 2009.

[9] J. Mendes Moreira, A. Cunha, and N. Macedo. An Orcid Based Synchronization Framework for a National Cris Ecosystem. *F1000Research*, 4, 2015.

[10] MySQL. MySQL :: The world's most popular open source database. https://www.mysql.com. (Last visited 24/9/2015).

[11] ORCID. What is ORCID. http://orcid.org/node/47. (Last visited 13/10/2015).

[12] Thomson Reuters. Web of Science. http://wokinfo.com/. (Last visited 12/10/2015).

[13] Raphael Saunier. *Getting Started with Laravel 4*. PACKT Publishing, 2014.

[14] Gijs Van Tulder. Storing Hierarchical Data in a Database. http://www.sitepoint.com/hierarchical-data-database/, April 2003. (Last visited 23/9/2015).

[15] Wikipedia. Google Scholar. https://en.wikipedia.org/wiki/Google_Scholar. (Last visited 13/10/2015).

Appendix A

Acronyms

APR Authenticus Publication Record

APR-WS Authenticus Publications Retrieval Web Service

MVC Model View Controller

RDBMS Relational Database Management System

DBMS Database Management System

ACID Atomicity, Consistency, Isolation, Durability

CRIS Current Research Information System

Appendix B

Authenticus Publication Document Type Mapping

This appendix contains of tables that characterize the Authenticus Publication Document Type Mapping. It consists of two tables:

1. Publication Document Type Mapping on Import.
2. Publication Document Type Mapping on Export.

1. Publication Document Type Mapping on Import.

	Authenticus	Web of Science	Scopus	DBLP
1	Abstract	Abstract of Published Item, Meeting Abstract	Abstract Report	
2	Article	Article, Article; Book Chapter, Article; Proceedings Paper	Article	article
3	Article in Press		Article in Press	
4	Bibliography	Bibliography		
5	Biographical-Item	Biographical-Item, Item About an Individual		
6	Book		Book	book
7	Book Chapter		Book Chapter, Chapter	incollection
8	Correction	Correction, Correction, Addition	Erratum	

Continued on next page

Table B.1 – continued from previous page

	Authenticus	Web of Science	Scopus	DBLP
9	Discussion	Discussion		
10	Edited Book			edited book
11	Editorial Material	Editorial Material, Editorial Material; Book Chapter	EditorialShort Survey	
12	Excerpt	Excerpt		
13	Fiction	Fiction, Creative Prose		
14	Letter	Letter	Letter	
15	News	News Item		
16	Note	Note	Note	
17	Patent		Patent	
18	Poetry	Poetry		
19	Proceedings			proceedings
20	Proceedings Paper	Proceedings Paper	Conference Paper	inproceedings
22	Reprint	Reprint		
23	Review	Art Exhibit Review, Book Review, Music Score Review, Record Review, Review, Review; Book Chapter, Software Review, Theater Review	Conference Review, Review	
24	Report		Report	

Table B.1: Publication Document Type Mapping on Import.

2. Publication Document Type Mapping on Export.

	Authenticus (CSV, PDF, HTML	**ENDNOTE**	**ISI**	**BibTex**	**RIS**
1	Abstract	Journal Articles	Abstract of Published Item, Meeting Abstract	misc	ABST
2	Article	Journal Articles	Article, Article; Book Chapter Article; Proceedings	article	JOUR
3	Article in Press	Generic	Article in Press	misc	GEN
4	Bibliography	Generic	Bibliography	misc	GEN
5	Biographical-Item	Generic	Biographical-Item, Item About an Individual	misc	GEN
6	Book	Book	Book	book	BOOK
7	Book Chapter	Book Section	Book Chapter	incollection	CHAP
8	Correction	Generic	Correction, Correction, Addition	misc	GEN
9	Discussion	Generic	Discussion	misc	GEN
10	Edited Book	Edited Book	Edited Book	book	EDBOOK
11	Editorial Material	Generic	Editorial Material, Editorial Material; Book Chapter	misc	GEN
12	Excerpt	Generic	Excerpt	misc	GEN
13	Fiction	Generic	Fiction; Creative Prose	misc	GEN
14	Letter	Generic	Letter	misc	GEN
15	News	Newspaper Article	News	misc	NEWS
16	Note	Generic	Note	misc	GEN
17	Patent	Patent	Patent	misc	PAT
18	Poetry	Generic	Poetry	misc	GEN
19	Proceedings	Conference Proceeding	Proceedings	proceedings	CONF
20	Proceedings Paper	Conference Paper	Proceedings Paper	inproceedings	CPAPER
21	Reprint	Generic	Reprint	misc	GEN

Continued on next page

Table B.2 – continued from previous page

	Authenticus	ENDNOTE	ISI	BibTex	RIS
22	Review	Generic	Art Exhibit Review, Book Review, Music Score Review, Record Review, Review; Book Chapter, Software Review, Theater Review	misc	GEN
23	Report	Report	Report	techreport	RPRT

Table B.2: Publication Document Type Mapping on Export.

Appendix C

Authenticus Publication Data Profiles

This appendix contains the tables characterizing the Authenticus Publication Data Profiles. They consist of two tables:

1. Authenticus Publication Profiles - Data about publication.
2. Authenticus Publication Profiles - Data retrieved from relations a publication has with other models.

1. Authenticus Publication Profiles - Data about publication.

Data Group	Data	Admin Profile	Full Profile	Short Profile
	Id + AuthenticusId	•	•	•
	document_title	•	•	•
	authors_string	•	•	•
	year_published	•	•	•
	abstract	•	•	
	publication_date	•	•	•
	iso_source_abbreviation	•	•	
	character_source_abbreviation	•	•	
	issn	•	•	•
	volume	•	•	•
	issue	•	•	•
	pages	•	•	•
	page_count	•	•	•
	reprint_address	•	•	

Continued on next page

Table C.1 – continued from previous page

Data Group	Data	Admin Profile	Full Profile	Short Profile
	doi	•	•	•
	book_doi	•	•	•
	coden	•	•	
	language	•	•	
	publisher	•	•	
	publisher_city	•	•	
	publisher_address	•	•	
	emails	•	•	
	funding_text	•	•	
	funding	•	•	
	conference_title	•	•	•
	conference_date	•	•	•
	conference_location	•	•	•
	scopus_conference_code	•	•	
	isbn	•	•	
	eisbn	•	•	
	conference_sponsors	•	•	•
	part_number	•	•	•
	editors	•	•	•
	conference_host	•	•	•
	book_group_authors	•	•	•
	book_series_title	•	•	•
	supplement	•	•	
	meeting_abstract	•	•	
	group_authors	•	•	•
	special_issue	•	•	
	eissn	•	•	
	article_number	•	•	•
	book_series_editors	•	•	•
	book_authors_full_name	•	•	•
	sponsors	•	•	
	tradenames	•	•	
	manufacturers	•	•	
	molecular_sequence_numbers	•	•	
	update_time/ create_time /last_identification /preprocessed	•	•	

Table C.1: Authenticus Publication Profiles - Data about publication.

1. Authenticus Publication Profiles - Data retrieved from relations a publication

has with other models.

Data Group	Data	Admin Profile	Full Profile	Short Profile
Publication Authors	author_id	•	•	•
	researcher_id	•	•	•
	author_name	•	•	•
	full_name	•	•	
	initials_list	•	•	
	givennames	•	•	
	surnames	•	•	
	junior	•	•	
	email	•	•	
	name_order	•		
	probability	•		
	validated	•	•	
	data_associated/update_time	•		
Document Type	authenticus	•	•	•
	dblp	•	•	
	scopus	•	•	
	wos	•	•	
Publication Affiliations	scopus	•	•	
	wos	•	•	
Publication Journal	journal_id	•	•	
	normalized_title	•	•	
Publication Conference	conference_id	•	•	
	conference_title	•	•	
Publication Categories (Web of Science Categories)	category_name	•	•	
Publication Subject Categories (Research Areas)	category_name	•	•	
Publication Keywords	keyword_id	•	•	
	keyword_name	•	•	
Publication External Keywords	scopus	•	•	
	wos	•	•	
External IDs	arxiv	•	•	•
	dblp	•	•	•
	pubmed	•	•	•
	scopus	•	•	•
	wos	•	•	•

Continued on next page

Table C.2 – continued from previous page

Data Group	Data	Admin Profile	Full Profile	Short Profile
	wos_document_delivery_number	•	•	
Publication Citations (history of citations per year)	scopus	•	•	
	wos	•	•	
Publication Time Cited (current number of citations)	scholar	•	•	
	scopus	•	•	
	wos	•	•	
Publication URLs	scopus	•	•	
	dblp	•	•	
	openurl_fulltext	•	•	
	pdf	•	•	
	link	•	•	
Publication Institutions (data about institutions identified and affiliated with publication)	institution_id	•	•	
	institution_path	•	•	
Publication Sources (data about metadata sources)	source_name	•	•	•

Table C.2: Authenticus Publication Profiles - Data retrieved from relations that a publication has with other models.

Appendix D

Authenticus Publication Record XML Schema

This appendix contains an Authenticus Publication Record XML Schema defintion.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.authenticus.pt/
   AuthenticusPublicationRecord"
  elementFormDefault="qualified" xmlns="http://www.w3.org/2001/
     XMLSchema"
  xmlns:authenticus="http://www.authenticus.pt/
     AuthenticusPublicationRecord">

    <element name="authenticus_publication_record">
    <complexType>
      <sequence>
        <element ref="authenticus:publication_data"></element>
        <element ref="authenticus:authors"></element>
        <element ref="authenticus:editors" maxOccurs="1" minOccurs="0"
           ></element>
        <element ref="authenticus:journal" maxOccurs="1" minOccurs="0"
           ></element>
        <element ref="authenticus:conference" maxOccurs="1" minOccurs=
           "0"></element>
        <element ref="authenticus:affiliations" maxOccurs="1"
           minOccurs="0"></element>
        <element ref="authenticus:keywords" maxOccurs="1" minOccurs="0
           "></element><!-- "type" = keywords|external_keywords -->
        <element ref="authenticus:categories" maxOccurs="1" minOccurs=
           "0"></element><!-- source="wos" type="categories", source="
           wos_jcr" type ="subject_categories" -->
        <element ref="authenticus:citations_history" maxOccurs="1"
           minOccurs="0"></element>
        <element ref="authenticus:times_cited" maxOccurs="1" minOccurs
           ="0"></element>
        <element ref="authenticus:identifiers" maxOccurs="1" minOccurs
           ="0"></element>
```

```
        <element name="publication_sources" type="string" maxOccurs="1
           " minOccurs="1"></element>
      </sequence>
    </complexType>
  </element>

  <complexType name="PublicationDataType">
    <all>
      <element name="authenticusID" type="string"></element>
      <element name="document_type" type="authenticus:DocumentType"></
         element>
      <element name="abstract" type="string" maxOccurs="1" minOccurs="
         0"></element>
          <element name="article_number" minOccurs="0" maxOccurs="1"
              type="string"></element>
          <element name="authors_string" minOccurs="0" maxOccurs="1"
              type="string"></element>
          <element name="book_authors_full_name" minOccurs="0"
             maxOccurs="1" type="string"></element>
          <element name="book_doi" minOccurs="0" maxOccurs="1" type=
             "string"></element>
          <element name="book_group_authors" minOccurs="0" maxOccurs
             ="1" type="string"></element>
          <element name="book_series_editors" minOccurs="0"
             maxOccurs="1" type="string"></element>
          <element name="book_series_title" minOccurs="0" maxOccurs=
             "1" type="string"></element>
          <element name="character_source_abbreviation" minOccurs="0
             " maxOccurs="1" type="string"></element>
          <element name="coden" minOccurs="0" maxOccurs="1" type="
             string"></element>
          <element name="conference_date" minOccurs="0" maxOccurs="1
             " type="string"></element>
          <element name="conference_host" minOccurs="0" maxOccurs="1
             " type="string"></element>
          <element name="conference_location" minOccurs="0"
             maxOccurs="1" type="string"></element>
          <element name="conference_sponsors" minOccurs="0"
             maxOccurs="1" type="string"></element>
          <element name="conference_title" minOccurs="0" maxOccurs="
             1" type="string"></element>
          <element name="create_time" minOccurs="0" maxOccurs="1"
             type="string"></element>
          <element name="document_title" minOccurs="1" maxOccurs="1"
              type="string"></element>
          <element name="doi" minOccurs="0" maxOccurs="1" type="
             string"></element>
          <element name="eisbn" minOccurs="0" maxOccurs="1" type="
             string"></element>
          <element name="eissn" minOccurs="0" maxOccurs="1" type="
             string"></element>
          <element name="emails" minOccurs="0" maxOccurs="1" type="
             string"></element>
```

```
            <element name="funding" minOccurs="0" maxOccurs="1" type="
               string"></element>
            <element name="funding_text" minOccurs="0" maxOccurs="1"
               type="string"></element>
            <element name="group_authors" minOccurs="0" maxOccurs="1"
               type="string"></element>
            <element name="isbn" minOccurs="0" maxOccurs="1" type="
               string"></element>
            <element name="iso_source_abbreviation" minOccurs="0"
               maxOccurs="1" type="string"></element>
            <element name="issn" minOccurs="0" maxOccurs="1" type="
               string"></element>
            <element name="issue" minOccurs="0" maxOccurs="1" type="
               string"></element>
            <element name="language" minOccurs="0" maxOccurs="1" type=
               "string"></element>
            <element name="last_identification" minOccurs="0"
               maxOccurs="1" type="string"></element>
            <element name="manufacturers" minOccurs="0" maxOccurs="1"
               type="string"></element>
            <element name="meeting_abstract" minOccurs="0" maxOccurs="
               1" type="string"></element>
            <element name="molecular_sequence_numbers" minOccurs="0"
               maxOccurs="1" type="string"></element>
            <element name="pages" minOccurs="0" maxOccurs="1" type="
               string"></element>
            <element name="page_count" minOccurs="0" maxOccurs="1"
               type="string"></element>
            <element name="part_number" minOccurs="0" maxOccurs="1"
               type="string"></element>
            <element name="preprocessed" minOccurs="0" maxOccurs="1"
               type="string"></element>
            <element name="publication_name" minOccurs="0" maxOccurs="
               1" type="string"></element>
        <element name="publication_date" minOccurs="0" maxOccurs="1"
           type="string"></element>
            <element name="publisher" minOccurs="0" maxOccurs="1" type
               ="string"></element>
            <element name="publisher_address" minOccurs="0" maxOccurs=
               "1" type="string"></element>
            <element name="publisher_city" minOccurs="0" maxOccurs="1"
                type="string"></element>
            <element name="reprint_address" minOccurs="0" maxOccurs="1
               " type="string"></element>
            <element name="scopus_conference_code" minOccurs="0"
               maxOccurs="1" type="string"></element>
            <element name="special_issue" minOccurs="0" maxOccurs="1"
               type="string"></element>
            <element name="sponsors" minOccurs="0" maxOccurs="1" type=
               "string"></element>
            <element name="supplement" minOccurs="0" maxOccurs="1"
               type="string"></element>
            <element name="tradenames" minOccurs="0" maxOccurs="1"
```

```
            type="string"></element>
        <element name="update_time" minOccurs="0" maxOccurs="1"
            type="string"></element>
        <element name="volume" minOccurs="0" maxOccurs="1" type="
            string"></element>
        <element name="year_published" minOccurs="1" maxOccurs="1"
             type="string"></element>
  </all>
  <attribute name="ID" type="string"></attribute>
</complexType>

<element name="publication_data" type="authenticus:
   PublicationDataType"></element>

<complexType name="DocumentType">
  <all>
    <element name="authenticus" type="string" maxOccurs="1"
      minOccurs="1">
    </element>
    <element name="wos" type="string" maxOccurs="1" minOccurs="0">
    </element>
    <element name="scopus" type="string" maxOccurs="1" minOccurs="0"
       >
    </element>
    <element name="dblp" type="string" maxOccurs="1" minOccurs="0">
    </element>
  </all>
</complexType>

<complexType name="ListType">
  <group ref="authenticus:ListItems"></group>
</complexType>

<group name="ListItems">
      <sequence>
          <element name="item" minOccurs="1" maxOccurs="unbounded">
              <complexType>
                  <simpleContent>
                      <extension base="string">
                        <attribute name="id" type="int" use="
                            optional"></attribute>
                        <attribute name="type" type="string" use="
                            optional"></attribute>
                          <attribute name="source" type="authenticus
                              :SourceIdentifier" use="optional"></
                              attribute>
                      </extension>
                  </simpleContent>
              </complexType>
          </element>
      </sequence>
  </group>
```

```
  <simpleType name="SourceIdentifier">
      <restriction base="string">
          <enumeration value="wos"></enumeration>
          <enumeration value="wos_jcr"></enumeration>
          <enumeration value="scopus"></enumeration>
          <enumeration value="scholar"></enumeration>
          <enumeration value="dblp"></enumeration>
          <enumeration value="link"></enumeration>
          <enumeration value="pdf"></enumeration>
          <enumeration value="openurl_fulltext"></enumeration>
          <enumeration value="arxiv"></enumeration>
          <enumeration value="pubmed"></enumeration>
          <enumeration value="wos_document_delivery_number"></
              enumeration>
      </restriction>
  </simpleType>

  <complexType name="CitationType">
    <sequence>
      <element name="times_cited" type="string" maxOccurs="1"
        minOccurs="1">
      </element>
      <element name="last_update" type="string" maxOccurs="1"
        minOccurs="1">
      </element>
      <element name="citations_url" type="string" maxOccurs="1"
        minOccurs="1">
      </element>
      <element name="url" type="string" maxOccurs="1"
        minOccurs="1">
      </element>
    </sequence>
    <attribute name="source" type="string"></attribute>
  </complexType>

  <element name="citation" type="authenticus:CitationType"></element
      >

  <complexType name="CitationListType">
      <sequence>
    <element ref="authenticus:citation" maxOccurs="unbounded"
      minOccurs="1">
    </element>
  </sequence>
  </complexType>

  <element name="authors" type="authenticus:AuthorsListType"></
      element>

<complexType name="AuthorsListType">
  <sequence>
    <element name="authors_count" type="int" maxOccurs="1" minOccurs
      ="1"></element>
```

```
    <element ref="authenticus:author" maxOccurs="unbounded"
      minOccurs="0">
    </element>
  </sequence>
</complexType>

<element name="editors" type="authenticus:EditorsListType"></element
   >

<complexType name="EditorsListType">
  <sequence>
    <element name="editors_count" type="int" maxOccurs="1" minOccurs
       ="1"></element>
    <element ref="authenticus:editor" maxOccurs="unbounded"
      minOccurs="0">
    </element>
  </sequence>
</complexType>

<element name="author" type="authenticus:PersonType"></element>
<element name="editor" type="authenticus:PersonType"></element>

<complexType name="PersonType">
  <all>
    <element name="name" type="string" maxOccurs="1"
      minOccurs="1"></element> <!-- author_name for author,
         editor_name for editor -->
    <element name="first_name" type="string" maxOccurs="1"
      minOccurs="0"></element>
    <element name="surname" type="string" maxOccurs="1"
      minOccurs="0"></element>
    <element name="fullname" type="string" maxOccurs="1"
      minOccurs="0"></element>
    <element name="email" type="string" maxOccurs="1" minOccurs="0">
       </element>
    <element name="default_author_name" type="string" maxOccurs="1"
      minOccurs="0"></element>
    <element name="given_names" type="authenticus:ListType"
      minOccurs="0" maxOccurs="1"></element>
    <element name="initials" type="authenticus:ListType"
      maxOccurs="1" minOccurs="0"></element>
    <element name="identifiers" type="authenticus:PersonIdentifiers"
       maxOccurs="1"
      minOccurs="0"></element>
    <element name="validated" type="boolean" maxOccurs="1"
      minOccurs="0">
    </element>
    <element name="name_order" type="int" maxOccurs="1" minOccurs="1
       ">
    </element>
  </all>
</complexType>
```

```
  <complexType name="PersonIdentifiers">
        <group ref="authenticus:ListPersonIdentifier"></group>
    </complexType>

    <group name="ListPersonIdentifier">
        <sequence>
            <element name="item" minOccurs="1" maxOccurs="unbounded">
                <complexType>
                    <simpleContent>
                        <extension base="string">
                            <attribute name="type" type="authenticus:
                               PersonIdentifier" use="required"></
                               attribute>
                            <attribute name="link" type="string" use="
                               optional"></attribute>
                        </extension>
                    </simpleContent>
                </complexType>
            </element>
        </sequence>
    </group>

  <simpleType name="PersonIdentifier">
    <restriction base="string">
      <enumeration value="external_institutional_id"></enumeration><
         !--Put institution abbreviation in link. Ex:<item type="
         external_institutional_ID" link="UP">3352343</item> -->
      <enumeration value="rebides_id"></enumeration><!-- Put year of
         rebides ID in link -->
      <enumeration value="orcid_id"></enumeration>
      <enumeration value="authenticus_ID"></enumeration>
      <enumeration value="scholar_id"></enumeration>
      <enumeration value="dblp_id"></enumeration>
      <enumeration value="webpage"></enumeration>
      <enumeration value="researcherID_id"></enumeration>
      <enumeration value="fct_id"></enumeration>
    </restriction>
  </simpleType>

  <element name="journal" type="authenticus:ListType"></element>
  <element name="conference" type="authenticus:ListType"></element>
  <element name="affiliations" type="authenticus:ListType"></element>
  <element name="keywords" type="authenticus:ListType"></element>
  <element name="categories" type="authenticus:ListType"></element>
  <element name="citations_history" type="authenticus:ListType"></
     element>
  <element name="times_cited" type="authenticus:CitationListType"></
     element>
  <element name="identifiers" type="authenticus:ListType"></element>
</schema>
```

Listing D.1: Authenticus Publication Record XML Schema.