UNIVERSITÀ DI PISA

Scuola di Dottorato in Ingegneria "Leonardo da Vinci"

Corso di Dottorato di Ricerca in
Ingegneria dell'Informazione

Tesi di Dottorato di Ricerca

Enhanced Publication Management Systems: a systemic approach towards modern scientific communication

Autore:

Alessia Bardi

Relatori:

Dr. Paolo Manghi

Prof.ssa Cinzia Bernardeschi

Anno 2016
SSD ING-INF/05

Sommario

Background L'impatto della rivoluzione digitale e l'adozione di massa dell'ICT hanno influenzato solo parzialmente la comunicazione scientifica. Gli scienziati svolgono le loro attività di ricerca eseguendo software e workflow scientifici, generando dati in formato elettronico e usando strumenti messi a disposizione da infrastrutture per e-science. Nonostante ciò, la disseminazione dei risultati di ricerca avviene ancora per mezzo dell'articolo scientifico, la cui forma è semplicemente passata da stampata a digitale. L'articolo scientifico da solo, però, non è sufficiente ad assicurare la riproducibilità di una ricerca scientifica: il raggiungimento di tale obiettivo è reso possibile dalla divulgazione e condivisione di tutti i prodotti relativi ad un'attività di ricerca.

Problema Nell'ultimo decennio, sull'onda dell'Open Science, la comunità scientifica ha affrontato il problema della pubblicazione di prodotti della ricerca diversi dall'articolo scientifico. Una delle soluzioni è il paradigma delle *enhanced publications* (EP). Le EP sono oggetti digitali che aggregano un articolo scientifico digitale e gli altri prodotti che sono stati usati o generati durante una ricerca e che sono utili per: (*i*) facilitare la comprensione dell'articolo, (*ii*) abilitare una più efficiente peer review e (*iii*) supportare la riproducibilità della ricerca scientifica. Teoria e pratica delle EP non sono ancora sviluppate e la maggior parte dei sistemi informatici per enhanced publication (Enhanced Publication Information System, EPIS) sono implementazioni su misura per servire specifiche comunità di ricercatori. Designer e sviluppatori di EPIS non hanno un supporto tecnologico adeguato orientato alle EP. Infatti, gli EPIS sono tipicamente realizzati con un approccio "from scratch", mirato alle specificità della comunità di ricercatori da servire.

Approccio Lo scopo di questa tesi è di proporre un approccio sistemico alla realizzazione di EPIS ispirato all'esperienza delle basi di dati. Lo stato dell'arte dei sistemi informatici e modelli dei dati per EP è stato analizzato per identificare fattori comuni fra diversi domini. Questi fattori comuni sono stati usati come base per la definizione di un modello dei dati e delle funzionalità per la rappresentazione e manipolazione di EP. La nozione di sistema di gestione per EP (Enhanced Publication Management System, EPMS) è introdotta per denotare sistemi informatici che forniscono a designer e sviluppatori un insieme di strumenti orientati alle EP per la realizzazione, operazione e manutenzione di EPIS. Si sono identificati i requisiti di EPMS e definita un'architettura software di riferimento che li soddisfa.

Contributi I principali contributi di questi tesi riguardano i campi della scienza dell'informazione e della comunicazione scientifica. L'analisi dello stato dell'arte degli EPIS ha prodotto una terminologia ed una classificazione che possono essere utili come riferimento per il confronto e discussione su EPIS. Un approccio sistemico, basato sul nuovo concetto di EPMS, è proposto come soluzione più conveniente alla realizzazione di EPIS, se paragonato al tipico approccio "from scratch". Un'architettura di riferimento per EPMS e un modello dei dati generico per EP sono proposti allo scopo di contribuire alla formazione di basi strutturate di quello che oggi sta diventando un'area di ricerca a sè stante.

Abstract

Background The impact of the digital revolution and the mass adoption of ICT affected only partially the scientific communication workflow. Scientists are today acquainted to scientific workflows, electronic data, software, e-science infrastructures for carrying out their daily research activities, but the dissemination of research results still relies on the bare scientific article, which simply shifted from being printed to digital. The scientific article alone, however, cannot support an effective assessment of research results or enable science reproducibility: to achieve this goal all products related to a research activity should be shared and disseminated.

Problem In the last decades, on the wave of Open Science, the scientific community has approached the problem of publishing research products different from the scientific article. One of the solutions is the paradigm of *enhanced publications* (EPs). EPs are digital objects that aggregate a digital scientific article and the other research products that have been used and produced during the research investigation described by the article and are useful to: (*i*) better interpret the article, (*ii*) enable more effective peer review, and (*iii*) facilitate or support reproducibility of science. Theory and practice of EPs is still not advanced and most Enhanced Publication Information Systems (EPISs) are custom implementations serving community specific needs. EPIS designers and developers have little or no technological support oriented to EPs. In fact, they realize EP-oriented software with a "from scratch" approach, addressing the peculiarities of the community to serve.

Approach The aim of this thesis is to propose a systemic approach to the realisation of EPISs inspired by the lessons learned from the database domain. The state of the art of information systems and data models for EPs has been analysed to identify the common features across different domains. Those common features have served as building blocks for the definition of a data model and functionalities for the representation and manipulation of EPs. The notion of Enhanced Publication Management System (EPMS) is introduced to denote information systems that provide EPIS designers and developers with EP-oriented tools for the setup, operation and maintenance of EPISs. The requirements of EPMSs have been identified and a reference software architecture that satisfies them is presented.

Contributions The main contributions of this thesis relate to the fields of information science and scientific communication. The analysis of the state of the art about EPIS results in a terminology and a classification that can be useful as reference for the comparison and discussion of such systems. A systemic approach, based on the novel notion of Enhanced Publication Management System (EPMS), is proposed as a more cost effective solution to the realization of EPIS, compared to the current "from scratch" strategy. A reference architecture for EPMSs and a general-purpose data model for EPs are proposed with the intent of contributing at building structured foundations of what is today becoming an area of research on its own.

The emerging digital world is supposed to emulate the printing world, but do its copying faster, more efficiently, more accurately.
This is precisely the point that must be questioned.
Jean-Claude Guédon[1]

[1] Niels Stern, Jean-Claude Guédon, and Thomas Jensen. Crystals of knowledge production. an intercontinental conversation about open science and the humanities. *Nordic Perspectives on Open Science*, 1(0):1–24, 2015

Acronyms

AAS Auhentication and Authorization Service
API Application Programming Interface
CERIF Common European Research Information Format
CMS Content Management System
CQL Contextual Query Language
CRUD Create, Read, Update, Delete
DBMS Data Base Management System
DC Dublin Core
DL Digital Library
DLS Digital Library System
EAD Encoded Archival Description
EP-DMDL EP Data Model Definition Language
DOI Digital Object Identifier
EP Enhanced Publication
EPIS Enhanced Publication Information System
EPMS Enhanced Publication Management System
FTP File Transfer Protocol
GUI Graphical User Interface
HTTP Hypertext Transfer Protocol
ICT Information and Communication Technology
JATS Journal Article Tag Suite
JDBC Java Database Connectivity
JISC Joint Information Systems Committee
JDAP Joint Data Archiving Policy
LDAP Lightweight Directory Access Protocol
METS Metadata Encoding and Transmission Standard
OAI Open Archive Initiative
OAI-ORE Open Archive Initiative - Object Reuse and Exchange
OAI-PMH Open Archive Initiative - Protocol for Metadata Harvesting

PID Persistent Identifier
PLOS Public Library of Science
RASH Research Articles in Simplified HTML
RDBMS Relational Data Base Management System
RDF Resource Description Framework
SOA Service-oriented architecture
SOAP Simple Object Access Protocol
SPARQL SPARQL Protocol and RDF Query Language
SQL Structured Query Language
SWORD Simple Web-service Offering Repository Deposit
URI Uniform Resource Identifier
URL Uniform Resource Locator
VRE Virtual Research Environment

Contents

List of Figures

List of Tables

1

Introduction

Scientific communication is intended as the ecosystem used to create, evaluate, preserve, share, discover, and access scientific knowledge. Different types of agents (or actors) are part of the scholarly communication with different roles, namely: authors, collaborators, reviewers, editors, publishers, readers, and organisations such as funders (e. g. Wellcome Trust, the European Commission) and institutions (e. g. universities, research centres). The sequence of steps performed by agents from the production to the dissemination and consumption of scientific knowledge is called scientific communication chain (or scientific communication workflow). Figure 1.1 depicts the different stages of the scientific communication chain and the main agents involved at each stage:

1. *Production of scientific knowledge*: scientists conduct their research activities and run experiments until their hypothesis are confirmed or refused. At any time of their work, they can decide to share the results of their (possibly ongoing) activities and produce a textual document, possibly embedding tables and figures, that describe their research and the (intermediate) results: the scientific article;
2. *Evaluation of scientific knowledge*: the article is read and assessed by a number of reviewers, who should verify the correctness of the methods and of the results described by the authors. Eventually, if reviewers accept the article, editors proofread the text and prepare the document for publishing;
3. *Preservation of scientific knowledge*: preservation of scientific articles and all materials submitted to a journal is one of the publishers' task. However, it is common that authors deposit their articles into their institutional repositories to ensure both the preservation and the accessibility to the scientific knowledge from within the original institution;[1]

[1] The possibility of depositing the publisher's version of an article is sometimes not possible for license limitations. In those cases usually it is possible to deposit a pre-print version of the article (i. e. without the style enhancements applied by the publisher). The advent of the initiative for Open Access [37] (access without payment fees or paid subscriptions) made deposition in institutional repositories simpler, as the researchers do not need to worry about breaking any license directives when depositing.

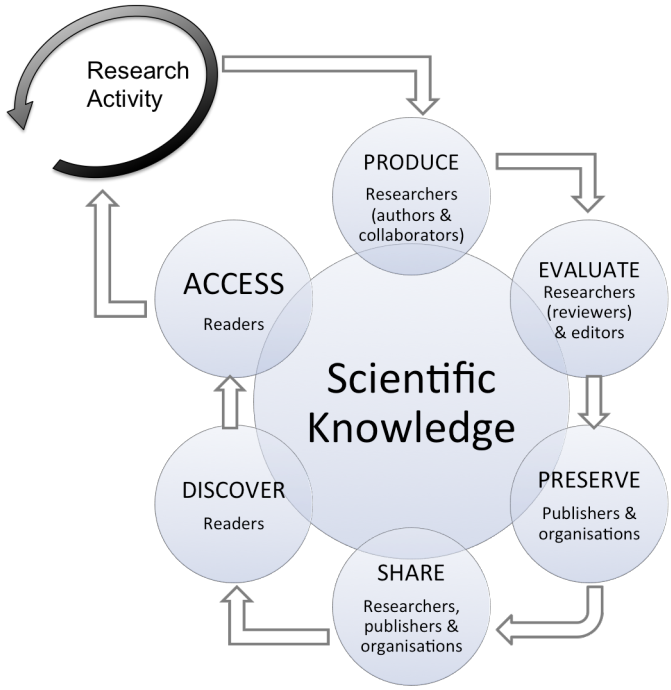


Fig. 1.1. The scientific communication workflow

4. *Sharing (or dissemination of) scientific knowledge*: all publishers operate a web site where new published article are advertised, but there are other ways a researcher can share and disseminate his/her results. Buzzes among colleagues and email exchanges are always useful to share research findings, but a wider audience can be reached by exploiting social scholarly communication tools, whose worldwide adoption is in rapid increase. For example, research social sites like Mendeley[2] and Research Gate[3] are a good opportunity to increase the visibility of a research activity and share published articles among researchers in the network. Uploading presentations that summarise the research findings on web site like figshare[4] is also a way to reach potentially interested researchers and users;
5. *Discovery of scientific knowledge*: a number of tools and services are available to discover scientific knowledge, mostly based on search engines that create and keep up-to-date indices on the metadata and on the full-text of publications. Users of those tools (humans or machines) are generally enabled to perform simple keyword searches or advanced searches that identify a set of possible interesting publications matching the specified search criteria;

[2] Mendeley, free reference manager and academic social network: `https://www.mendeley.com`
[3] Research Gate, social network for researchers: `https://www.researchgate.net`
[4] figshare, a repository where users can make all of their research outputs available in a citable, shareable and discoverable manner: `http://figshare.com`

6. *Access to scientific knowledge*: once an interesting publication has been discovered, users typically want to read it to actually access the scientific knowledge it contains. Typically users download the scientific publication in textual format (e. g. PDF) or read the publication directly on-line, where basic or advanced reading tools may be available depending on the source that is providing access to the publication.

The digital revolution and the consequent mass adoption of Information and Communication Technology (ICT) had a strong impact on our society, especially in the way people communicate with each other and manage information. A similar impact was also expected to apply to scientific communication, with radical changes in the way scientists conduct their research, report about their outcomes, and how scientific knowledge is disseminated. The expectations had been met only partially. On the one hand, scientists changed radically their research tools and are acquainted to software, electronic data and digital infrastructures for carrying out their daily research activities [67]. On the other hand, although the potential of digital scholarly communication were clear since the '90s, the major advances in this direction simply shifted from "printed articles" to "digital articles". The possibilities of sharing other digital products of science [99, 113] , such as research data, software, tools, in order to enable effective repeatability and reproducibility of science have not been exploited, still relying on the scientific article to be the omni-comprehensive description and evidence of a scientific output.

In the last decade the first effective cultural and technological changes in digital scientific communication, pushed by demands of funders, research organizations, and scientists, have been witnessed. In many disciplines research data has gained a primary role in scientific communication [91, 33, 17, 146, 149, 80] and other products of science are today following. This metamorphosis is posing several challenges on how the aforementioned facets of production, evaluation, preservation, sharing, discovery, and access should be address for products different from scientific articles. Among the solutions proposed to these challenges, and of interest to this thesis, one of the most relevant is that of Enhanced Publications (EPs).

EPs are digital objects composed of a digital publication and other research products of different types (e. g. research data, workflows, software) that have been used or produced during a research investigation. Figure 1.2 shows an example of EP that aggregates different research products (a digital publication, a dataset from a remote database, software code and a CSV dataset) and their descriptive metadata, forming a graph of resources representing a research investigation and its scientific results.

Although of interest to scientific communities, theory and practice of EPs is still not advanced and most Enhanced Publication Information Systems (EPISs) are custom implementations serving community specific needs. EPIS designers and developers have little or no technological support oriented to EPs. In fact, they realize EP-oriented software with a “from scratch” approach, addressing the peculiarities of the community to serve and integrating technologies that are general purpose (e. g. databases, file stores) and Digital Library (DL)-oriented (e. g. repository software, cataloguing systems). The integration of such different technologies is not easy from the point of views of realization and

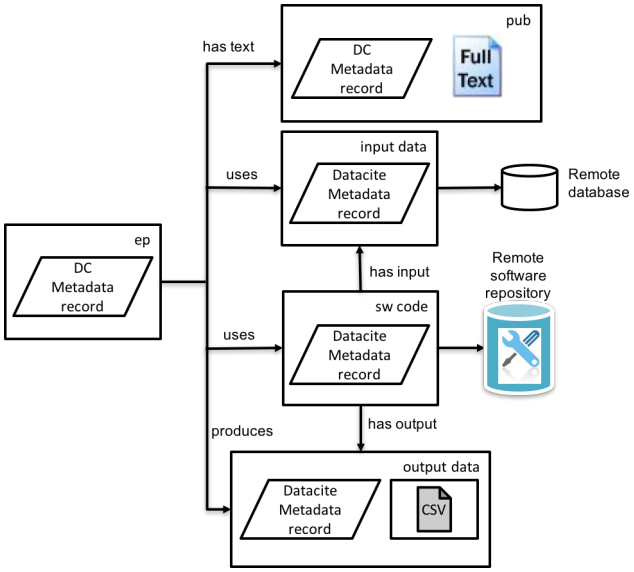


Fig. 1.2. Example of Enhanced Publication (EP)

maintenance. It is not a surprise, in fact, that the main agents in the realisation of EPISs are big publishers that have human and economic resources to afford such an investment (e. g. Public Library of Science (PLOS), Elsevier, and Nature). Research institutions, offering traditional Digital Library systems and repositories to researchers, cannot generally afford the realisation and maintenance costs of a new information system and thus they are currently excluded from the shift from traditional digital publications to EP.

This theses advocates the need of a systemic approach to the realisation of EPISs and presents the notion of Enhanced Publication Management System (EPMS) as a more cost effective solution to the realization of EPISs, compared to the "from scratch" strategy. The term Enhanced Publication Management System (EPMS) denotes an information system that provide EPIS designers and developers with EP-oriented tools for the setup, operation and maintenance of EPISs. A reference architecture for EPMSs and a general-purpose data model for EPs are proposed with the intent of contributing at building structured foundations of what is today becoming an area of research on its own.

Contributions of the thesis

The main contributions of this dissertation relate to the fields of information science and scientific communication:

- The analysis of the state of the art has produced a classification that can be useful as reference for the comparison and discussion of EPISs and EP data models.
- A systemic approach, based on the novel notion of EPMS, is proposed as a more cost effective solution to the realization of EPISs, compared to the current "from scratch" strategy.

- A reference software architecture for EPMSs is presented with the goal of setting up the foundations for the design of concrete EPMSs implementations that support the realization, maintainance and operation of customised EPISs.

Outline of the thesis

Chapter 2 gives some insights on the the impact of ICT on scientific communication and highlights that only in recent years real attempts of exploiting the digital nature of scientific publications have been proposed and adopted, although the potentialities of a digital scientific communication had been discussed since the '90s. Chapter 3 presents a classification methodology of EPISs based on their functional goals and the features of the data models they adopt. Existing EP data models and EPISs are then surveyed and classified. Chapter 4 proposes a general-purpose data model for EPs and a reference software architecture for EPMSs. Chapter 5 concludes the thesis.

2

Historical background

Nowadays, the primary channel for scientific knowledge dissemination is the scientific publication (e. g. scientific article published in a journal, conference paper). The scientific publication appeared in the second half of 17th century, when the first scientific journal was founded [70].[1] Since then, scientists wrote articles, which were submitted to print journals, reviewed, and then published and distributed to libraries.[2]

As a consequence of the digital revolution started in the '50s, scientific articles turned from print to digital and, thanks to the internet, they are now accessible worldwide. Nevertheless, it is only in the early '90s that scientific publications have been massively digitized thanks to national and international digitization programs, which were promoted for the realization of networks of Digital Libraries (DLs) and an "information infrastructure" [28, 39]. At the same time, researchers started generating born-digital publications by writing articles on their computers, which were then submitted to journals via on-line procedures. Even though the scientific communication chain after the digital revolution is not so different from the one in the 17th century, the step from print publications to digital (or electronic) publications brought several benefits:

1. The adoption of standard metadata formats for the description of scientific publications facilitated both the dissemination and the discovery phase of the scientific communication workflow. Publication repositories and DLs could easier exchange scientific publications – also thanks to the adoption of standard exchange protocols such as Open Archive Initiative - Protocol for Metadata Harvesting (OAI-PMH) and Simple Web-service Offering Repository Deposit (SWORD) –, increasing the dissemination of scientific knowledge across the world. Also, the discovery of publications became easier with respect to the past, when manual inspections of library catalogues was

[1] The first scientific journal is the *Philosophical Transactions of the Royal Society of London*, established in 1665 and still active (`http://rstl.royalsocietypublishing.org`). Copies of the first edition are preserved at the Bodleian Library of the University of Oxford, where I had the privilege to closely look at the copy of Sir Christoph Wren, one of the most famous English architects in history.

[2] Before the advent of journals, scientists were used to exchange information about their scientific discoveries and ideas via mail letters [70].

needed, thanks to the possibility of automatically creating and maintaining indices with homogeneous structure and that could be used to perform advanced searches;
2. The on-line availability of full-texts allowed faster access to and easier exchange of digital publications, also across geographically distributed communities of researchers;
3. Libraries had the capability to serve copies of digital publications to a higher (possibly infinite) number of users, while saving resources typically needed for preservation purposes such as physical storage space and mechanisms for environmental control;
4. The costs for publishing a journal decreased, especially in terms of printing and distribution costs, allowing publishers, scholarly societies and research communities to launch new journals.[3]

In 1991, Okerson [124] claimed that in 2000 all publishers would have delivered publications on-line, abandoning the traditional printing system. King [92] had a less radical vision, according to which digital publications are just "*extraordinarily useful complements to journals in print format*." In 1999 Treloar [154] wrote about a "revolution" of the scientific communication, where researchers exchange their outcomes on-line without geographical barriers and foresaw that the "*majority of print scholarly journals will have completed their transition to an online existence by 2010 and we will then see another period [...] of relative stasis*." Treloar's analysis turned out to be realistic. Although literature from the '90s shows the great expectations of the research communities with regards to digital publications, the digitization has not yet been exploited to its maximum capacity. Added features such as the integration of multimedia items in the text, navigation of the article via hyperlinks, inclusion of research data to be downloaded, possibility to attach user comments, and to update/version the published article were considered natural consequences of the process of digitization and the advent of the Internet [99, 113]. However, in the majority of cases digital publications are just mere copies of printed publications. Publishers and other stakeholders of the scientific communication adopted digital formats for publications (e. g. text files, PDF, and XML), but they did not fully exploit the capabilities of digital formats. Because of that, researchers have not been "traumatized" by the digital shift, but they also had little opportunity to experience the benefits. A survey conducted by Hitchcock et. al in 1996 [77] confirms that the majority of journals had not yet started to implement digital publications with novel features and preferred to retain the traditional paper layout.

In 2005 Owen extensively discussed the impact of Information and Communication Technology (ICT) in scientific communication and concluded that ICT transformed the scientific communication chain at the level of infrastructure but not its substance: scientific communication evolved to embrace ICT and adapted the form of the unit of communication, i. e. the scientific publication, to the digital infrastructure [126].

[3] It has to be noted that the decrease of printing and distribution costs had not affected the subscription fees paid by libraries: because of the growth of available journals, libraries had a harder job in selecting the journals to subscribe for, while publishers were instead able to increase their profits [71].

It is only in recent years that the scientific communication started to exit the phase of stasis depicted by Treloar [154] by recognizing that digital publications are only one type of research products and that the "traditional" model for the representation of scientific knowledge (i. e. a textual document and its metadata) is not sufficiently rich to support an effective assessment of "modern" data-intensive science. In data-intensive science research questions are answered by analysing the huge amount of research data generated by simulations and captured by high-throughput scientific instruments (telescopes, sensor networks, satellites, etc.) [67]. Research data is more and more considered "first-class citizen" of the scientific communication, rather than just a sub-product of a scientific publication [47]. In fact, a number of international activities are promoting best practices for the description, publishing, sharing, re-use, and citation of research data [91, 33, 17, 146, 149, 80]. Likely, the scientific processes used for the elaboration of research data are being regarded as fundamental resources for scientific knowledge assessment, because they represent the provenance of the data: how the research data supporting researchers' discoveries was generated and manipulated (e. g. for harmonisation purposes) [29, 21].

The light blue circle in figure 2.1 shows some of the main research products in traditional scientific communication: textual documents such as articles, scientific reports, project deliverables and books. However, those are only a subset of types of digital artefacts that researchers produce during their research activities. Researchers collect *primary* or *raw data* from different kind of *devices*, apply harmonisation and transformation *processes* to generate the so called *secondary data*, which is data in a format that is easier interpretable by humans and machines. The generation, manipulation and analysis of data is performed by running *software* and *workflows* with a specific configuration and on a specific hardware. The availability of these products, in the dark blue circle in figure 2.1, and the relationships that can occur among them, is fundamental to support a more effective assessment of research and to support or facilitate the reproducibility of science. The introduction of different types of research products allows a more contextualised and complete description of a research and its results, bringing a number of benefits at different stages of the scientific communication chain:

- Better support the assessment of scientific research, with tools that help reproducing research activities, thus reducing the chances of scientific fraud;
- Better support multi-disciplinary research, with tools that help interpreting and understanding research activities;
- Reward researchers for activities beyond the publishing of a scientific article, taking into account all facets of research life-cycle;
- Ease the discovery of and access to existing scientific knowledge within the "literature deluge" witnessed by researchers
- Reduce the cost of research by promoting re-use of results;

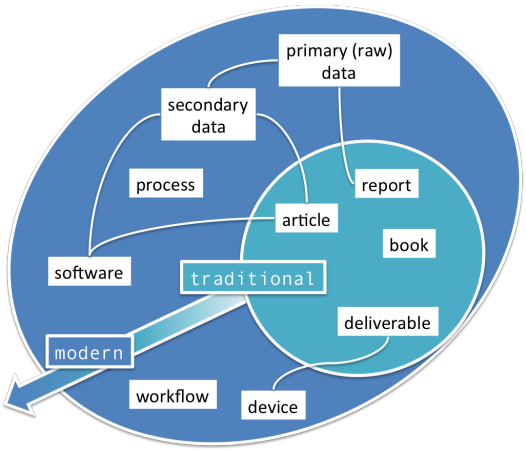


Fig. 2.1. Products of research activities in the traditional and the "modern" scientific communication workflow

Better support the assessment of scientific research

Reviewers are having a hard job in assessing the quality and value of a research described in a digital publication [30, 29, 88]. Often the lack of detailed information about the provenance of data and the processes used for its manipulation makes it impossible for them to identify errors or verify that the authors' conclusions are actually supported by the results of data analysis. In some cases, having the data, the processes, and the software used by authors does not ensure that an experiment can be reproduced, as the processing environment should be provided as well. For an experiment to be reproducible, all its parts should be made available.

Better support multi-disciplinary research

One of the impact of ICT in science and scientific communication is the promotion of international and multi-disciplinary research networks. For example, computer scientists collaborate with philosophers for the studies on ontologies and the semantic web, with researchers in cultural heritage preservation in order to improve results of artworks restoration, with biologists in order to generate more accurate algorithms for DNA analysis. In the latter case, the strict collaboration between computer scientists and biologists generated the new discipline of bioinformatics. It is not uncommon that a researcher in a field need to understand a research conducted in another field. In such a case the researcher probably does not fully understand the jargon used in the publication and needs tools that can help him/her understand the semantics of what she/he is reading. Also, the researcher might find useful to first look at other types of research products used for dissemination purposes by the authors of the publication: presentation slides for a non-technical audience, speech at conferences and blog articles are only some examples of research products that can be linked to the publication to help non-technical readers or newcomers in understanding the research activities [151].

Reward researchers for all their research products

Although researchers generate different typologies of research products such as publications, data, software, and processing workflow, they are evaluated only for one typology: publications. Research communities, with the support of other stakeholders of scientific communication, elaborated and implemented a workaround to this problem, so that researchers can be evaluated also for the research data they produce and curate. The workaround is to write publications that describe research data, known as "data papers", and to publish them in dedicated journals, known as "data journals" [94, 49, 38]. Examples of Open Access data journals are the Geoscience Data Journal,[4] the Biodiversity Data Journal,[5] and the Journal of Big Data.[6] A comprehensive survey on data journals conducted in 2014 can be found in [40]. With data papers and data journals, research data enters the scientific communication chain "disguised" as scientific article. Still, the underlying problem has not been tackled at its roots, as other types of research products are not considered in the criteria for the evaluation of researchers and their institutions.

Moreover, the adoption of ICT technologies, especially internet, introduced new communication channels for sharing scientific knowledge. In addition to conferences, e-journals, DLs, etc., newer communication means such as forums, blogs, and social networks can be included now in the list of novel dissemination means for scientific knowledge. Those new communication channels are often considered parts of *informal* scholarly communication, while the traditional channels are parts of the *formal* scholarly communication [20]. The distinction between formal and informal scholarly communication is not just theoretical. Researchers are officially rewarded for the results they share via formal scholarly communication. In the evaluation of curriculum vitae, scientific publications published in journals and conference proceedings give high scores, while research results disseminated via informal scholarly communication are instead typically not considered at all [74, 10, 125]. The same distinction is also applied in bibliometrics and scientometrics: researchers whose works are cited by journal articles are more positively evaluated than those who are cited in blogs, forums, social networks, and wikis. Studies on alternative metrics for research evaluation and citation metrics are ongoing in order to cope with the informal scholarly communication and its impact on scientific research [134].

Ease discovery of and access to scientific knowledge

A 2010 estimate on scientific production calculates that in 2009 the number of peer-reviewed, published articles reached 50 millions [86]. In 2014, Research Trends reported in [133, 123] that in the last ten years the number of published publications per year almost doubled from 1.3 millions in 2003 to 2.4 millions in 2013, according to data gathered

[4] Wiley Geoscience Data Journal: `http://onlinelibrary.wiley.com/journal/10.1002/(ISSN)2049-6060`

[5] Pensoft Biodiversity Data Journal: `http://biodiversitydatajournal.com`

[6] Journal of Big Data, a SpringerOpen Journal: `http://www.journalofbigdata.com`

from Scopus.[7] This phenomenon of high growth rate of published scientific articles is also known as *literature deluge*. While on one side researchers are pushed to publish more articles to improve their careers ("publish or perish") [62], on the other side they struggle to find relevant related work or interesting studies because of the literature deluge. As discussed at the beginning of this thesis, tools for discovering research articles are mostly based on full-text indices that allows to perform advanced search on the metadata and on the full-texts of publications. The inclusion of other types of research products in the scientific communication could help both for enabling more advanced queries, but also for fostering the adoption of different discovery patterns. For example, researchers may find relevant articles based on the same research data [114] or using the same workflow for data processing.

Reduce the cost of research by promoting re-use of results

A number of international activities are promoting best practices for describing, publishing, sharing, re-using, and citing research data [91, 33, 17, 146, 149, 80]. The main argument behind this growing movement in support of research results re-use is to maximize the generation of scientific knowledge from existing research products and, therefore, maximize the return of investments of funders. In particular, both Tenopir et al. in [149] and Arzberg et al. in [12] highlight the following main economical and social benefits in re-using research data:

- Optimizing the use of resources by minimizing re-collection of data;
- Maximizing the analysis of research data that can not be re-collected (e. g. atmospheric data)
- Generating new research data by combining data from different sources, especially useful in interdisciplinary researches;
- Facilitating the education of new researchers by supporting replication studies.

Scientists are interested in getting scientific reward for their efforts and in learning about ongoing research, as well as in being able to re-use the outcome of other's research to reproduce similar experiments, save time and avoid pointless mistakes. To meet such requirements, modern research is being increasingly conducted in so-called e-Science infrastructures and by adopting the collaborative approach of e-Research, which strongly promotes sharing and re-use of all research results, including traditional publications, datasets [104, 67, 115] and research experiments [142]. The uptake of such trends is leveraged by the fact that also funding agencies and organizations, which are crucial stakeholders in the research chain, are advocating and increasingly making mandatory the publishing and citations of any research products, in order to measure their return of investment, improve their funding strategies, or gain visibility and scientific rewards.

In summary, all stakeholders of scientific communication are now ready and willing to exploit new forms of scientific publishing and to change the scientific communication

[7] Scopus, the largest database of peer-reviewed literature, Elsevier: `http://www.scopus.com`

chain not only at the level of infrastructure, but also at the level of the unit of communication of scientific knowledge. The traditional approach – where results of a research are represented by a textual document (the scientific publication), in either paper or digital form, together with its descriptive metadata – cannot cope with these new needs of scientists [47].

3

Enhanced Publications (EPs): data models and information systems

In Chapter 2 it has been discussed that, since the beginning of the digital era, agents of the scientific communication chain understood the potentiality of digital publications but did not exploit it, opting instead to retain the traditional paper layout. The changes in the way scientists conduct their research are finally pushing for changes also in the way research results are represented and disseminated, so that novel information systems, which we denote Enhanced Publication Information Systems (EPISs), begin to be realised to manage Enhanced Publications (EPs).

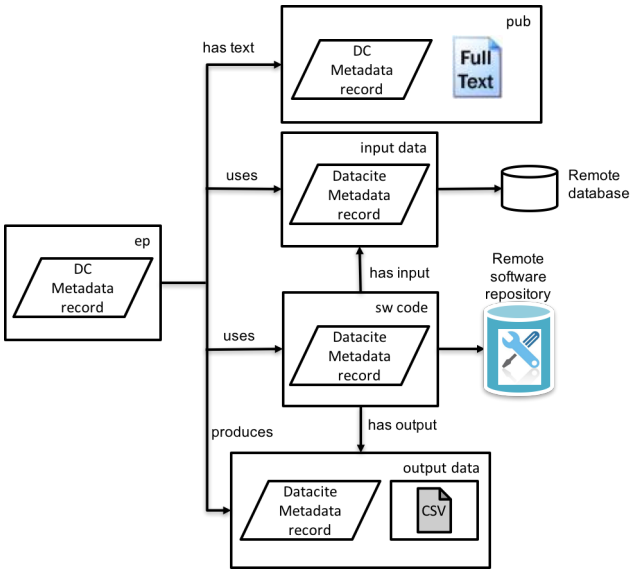
The notion of *EP* is an emerging paradigm for the representation and dissemination of research results. An EP enriches a digital publication with additional research products to complete the description of the research. As a traditional digital publication, an EP is characterized by an *identity* and *descriptive metadata*, but its nature enables improved and modern ways of implementing dissemination of and access to research products. Example 1 shows an EP that contextualises a traditional digital publication (full-text and metadata) with the research products used and produced in the experiment it describes: the input dataset (from a relational database), the software used for the elaboration of the input, and the generated output dataset in CSV format.

The need of enhancing digital publications with contextual information and resources is not under discussion: all agents of the scholarly communication recognize the importance of this process and the literature reveals a growing interest in data models for EPs and information systems for their management: EPISs. Existing EPISs may significantly vary in terms of their functional goals, e. g. Web 2.0 reading and discovery capabilities, and, in recent manifestations, re-production and assessment of scientific experiments. In general, they implement data models that describe EPs as consisting of parts, whose purpose is to define one mandatory textual description of the research conducted (as for the traditional article) and its relationships with other research products whose nature varies depending on the scientific context (e. g. datasets, images, tables, workflows, devices, services) and consumption purposes (e. g. visualization, experiment repetition).

The state of the art on EPs has today reached the point where some kind of common understanding and definition is required, in order to provide the terminology for scientists

Example 1 An EP that contextualises a traditional digital publication with input/output data and software

Different research products are aggregated by one EP in order to create a graph of resources representing a research investigation.
The figure below shows an example of EP for the representation of a research experiment that processed a dataset hosted on a remote database (*input data*) with a software (*sw code*) producing some output data in a CSV file (*output data*). All three research products comes with descriptive information in form of metadata records compliant to the Datacite format [3]. *input data* and *sw code* references remote resources (a dataset from a remote database and a software project on a software repository, respectively), while *output data* includes the actual CSV file containing the data processing results. The experiment is presented in a traditional digital publication (*pub*), represented as a full-text and descriptive metadata in Dublin Core format [56].
The EP aggregates all the research products and becomes a sort of "entry point" to the whole research investigation and not only to its description like traditional digital publications. All the research products related to the experiment are easily discoverable and accessible from the EP thanks to the semantic associations *has text*, *uses*, *produces*, *has input* and *has output*.



to classify, compare, analyse, or simply discuss, the multitude of solutions in the field. A preliminary study in this direction was carried out by the SURF Foundation [159] in the context of the DRIVER-II EC project [105]. As a result, EPs were defined as "*dynamic, versionable, identifiable compound objects combining an electronic publication with embedded or remote research data, extra materials, post publication data, database records, and metadata*". The investigation performed an analysis of requirements for the modelling and management of EPs in a cross-discipline scenario and resulted in an abstract and optimal data model, which was used as basis for a prototype of an EPIS [159]. Such work represented an important step in motivating and highlighting the existence of a novel re-

search field and, most importantly, defined the term ”Enhanced Publication“. This chapter extends these initial efforts in three ways:

1. Providing the terminology necessary to describe existing EP data models in terms of their structural and semantic features (Section 3.1);
2. Describing the issues EPIS designers and developers must cope with to address content, functional, and architectural requirements of EPISs (Section 3.2);
3. Proposing a classification of existing EPISs in terms of their functional goals (Section 3.3).

The goal is to help researchers in the field at better discussing and motivating their results, but also newcomers on this subject at organizing and structuring their understanding of the literature.

3.1 EP data models and features

Enhanced Publications (EPs) are digital objects characterized by an identifier (possibly a persistent identifier) and by descriptive metadata information. The constituent components of an EP include one mandatory textual narration part (the description of the research) and a set of interconnected sub-parts. Parts may have or not have an identifier and relative metadata descriptions and are connected by semantic relationships. In general, EP data models vary in the way they define the structure of their parts, metadata, and relationships, which reflect and support the functional goals of a given Enhanced Publication Information System (EPIS). For example in some approaches an EP is a “package” embedding all its sub-parts, i. e. sub-parts cannot be shared by different EPs. In other approaches, parts can instead be referenced, shared or passed as inputs to workflow engines. In some solutions the narrative part of an EP is intended in a traditional (for the digital world) sense as a readable file (PDF, DOCX, etc.); while in others, the text is structured into interconnected sub-parts, e. g. sections, figures, tables. Figure 3.1 shows two EPs consisting of the narrative part, representing the scientific article, and supplementary material: the slides presented at a conference, the video of the presentation, and one spread-sheet of related research data. The two publications have analogous structure for the supplementary material, but differ in the nature of the mandatory narration: a single PDF file versus a structured text. The first model is generally preferable in digital library settings where the traditional management of PDF articles is to be enriched with related research assets. The second model is more adequate for information systems supporting Web 2.0 GUIs.

An analysis of the literature on EP data models has been carried out to identify modelling patterns for their constituent parts, relationships between them, and associated metadata.

Metadata descriptions of the parts are provided at different interpretation levels, so as to enable both human and machine interpretation. Examples are bibliographic information, file information, provenance information, visualization information, execution

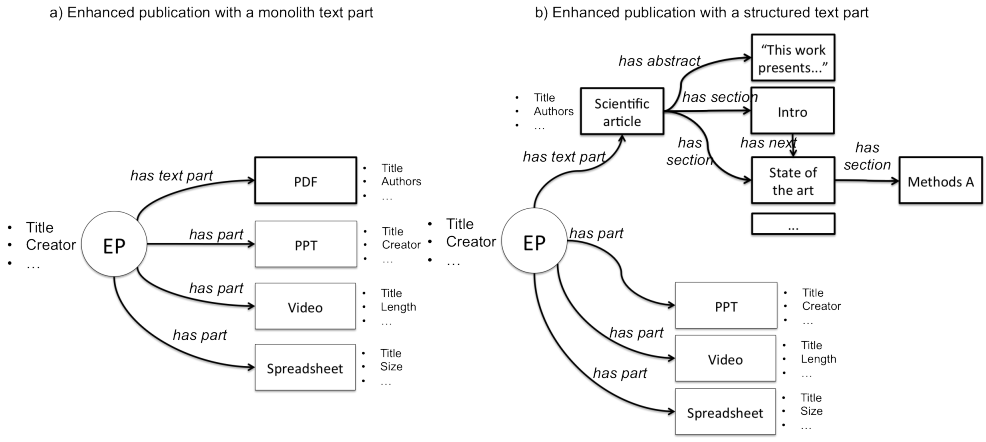


Fig. 3.1. Examples of EPs

information, versioning information, etc.. In fact, each part may bear several metadata descriptions in order to allow for multiple usages. Relationships between parts implicitly or explicitly (e. g. by a label) characterise the semantics of the association between two parts. Associations may indicate user-oriented links (e. g. chapter of, related with, uses dataset), application-oriented links (e. g. alternative visualization, external link, local link), and others typologies. Such classifications have been produced before [39] and are well-known to traditional information systems in the scientific communication chain, such as traditional digital library systems, digital archives, scientific data repositories, or scholarly communication infrastructures, e. g. OpenAIRE [107], Swedish ScienceNet [87], and CRIS systems [89]. As such, metadata formats and types of relationships may be regarded when describing the specificities of individual data models, but they are not able to capture the peculiar nature of EP data models. On the other hand, the constituent subparts of EPs, independently from the metadata descriptions and relationships accompanying them, are the most characterizing aspect of EP data models.

More specifically, the following classes of parts, or data model features, have been identified by dissecting existing data models (see figure 3.2):

- Embedded parts, e. g. enhancing a publication with supplementary material files;
- Structured-text parts, e. g. enhancing a publication providing an editorial structure of its textual sub-components;
- Reference parts, e. g. enhancing a publication with URLs to external objects;
- Executable parts, e. g. enhancing a publication with parts that include software and data to run an experiment.
- Generated parts, e. g. enhancing a publication with tables that may dynamically change depending on updates of given input research data.

In the following, descriptions of such classes are given, motivating their nature and exemplifying their occurrence

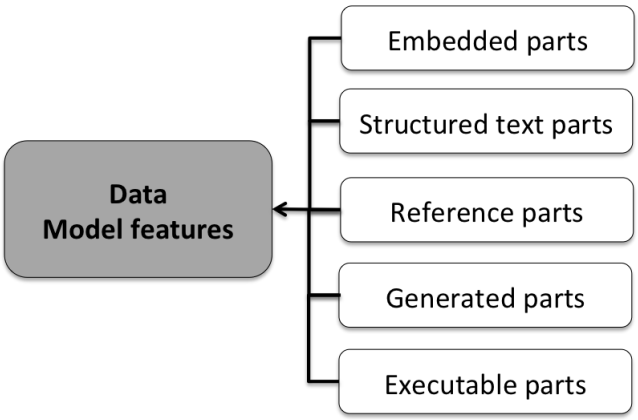


Fig. 3.2. EPs data model features

Embedded parts

A real-world example of data models with embedded parts is that of those publishers that intercepted the need of researchers to add “context" to scientific publications in order to improve the comprehension of their results. To this aim, they provide information systems where authors can upload so-called supplementary material along with the publication. Examples are presentation slides, appendixes to the article, data and description of data used for the research described in the article, high resolution images, tables that could not be inserted fully in the digital publication because of page limits.

More generally, data models with embedded parts describe EPs whose parts may be files, generally not described by metadata and without an identifier, hence not searchable or shareable by different publications. Typically, embedded parts are stored locally to the information system. The semantics of the relationships between the narrative part and supplementary material is often “silent”, although in some cases may bear information about the type or the meaning of the files [93] [41]. Figure 3.3 shows an example of an EP with embedded parts, where each part is accompanied by metadata descriptions.

Structured-text parts

Some modern approaches abandon the notion of textual publication as a single block of text, e. g. a PDF file, and experiment with the definition of publications as structured texts. Such solutions are often addressed by publishers to enhance publication readability via web 2.0 applications or given client applications. For example, Elsevier proposed the ”Article of the Future“ [9], which implements a Web-oriented viewer of a scientific publication, where the reader can browse through abstract, sections, paragraphs, view and download tables and images, generate PDF manifestations, interact with tables, etc.

Data models with structured-text parts describe EPs with a narration part that is a structured object composed of several interconnected parts, such as abstract, sections, figures, tables, bibliography. I

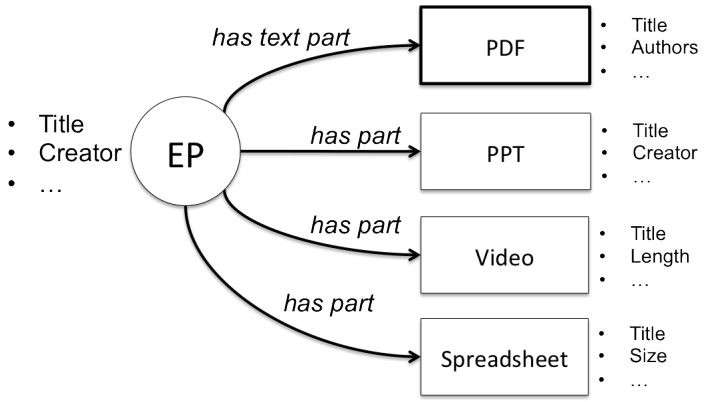


Fig. 3.3. EP with embedded parts

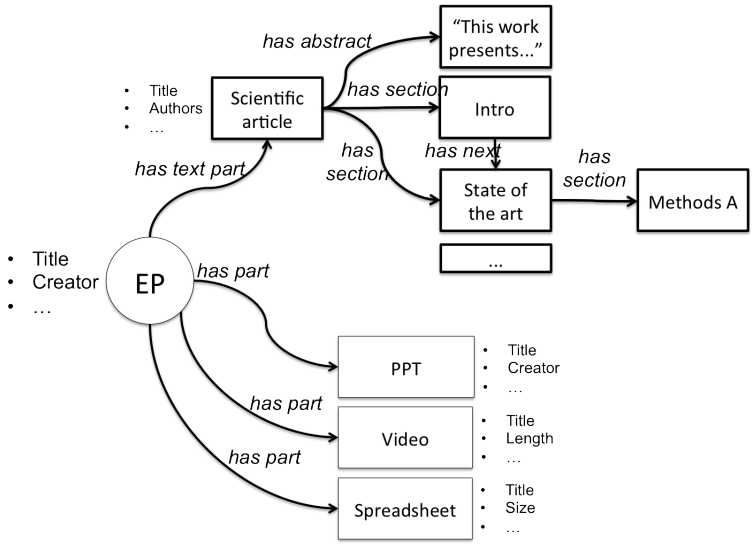


Fig. 3.4. EP with structured-text parts

Reference parts

Nowadays, a fully "embedded parts" approach is generally not sustainable nor effective. Firstly, the information system at hand should be able to store and handle all parts required to contextualize a publication and disseminate it as an EP. For example, publication and dataset management (e. g. storage, preservation, description) are typically complex and separate activities, handled by professionals through specialized tools [47]. Secondly, in many cases the objects to be referred by the EPs already exists, i. e. they are in fact re-used, and are stored into "external" information systems. Finally, embedded parts do not promote sharing and re-use since EP sub-parts are reachable and re-usable only via the mandatory narrative part, i. e. they do not have identity and metadata. These

limitations are overcome by EPs that feature links to remote research outputs, such as datasets, other (enhanced) publications (e. g. citations), or supplementary material (e. g. web sites, presentations). For example data papers [118] are the result of the recent trend to give scientific value to the production of datasets and reward their creators by publishing datasets as peculiar research outcomes. Data papers are EPs with a narrative part and a persistent reference part to the dataset stored in a remote data repository. Similarly, in many communities, literature reporting on experiment results is often referring to datasets used or generated by the experiment and vice versa. Other examples of reference parts are those supported by information systems capable of mining publication text to extract URL pointers to scientific knowledge [100, 117]), concepts represented in general-public Web sources (e.g. Wikipedia [64, 72, 135], DBPedia), or discipline-specific databases, ontologies, and taxonomies (e. g. WoRMS , UNIPROT).

Fig. 3.5 shows an example of an EP with reference parts. Data models with reference parts describe EPs whose parts may be references to objects that are "external" to the EP, hence are possibly shared with others. Such references may be community-specific identifiers (e. g. PubMed identifiers), unique persistent identifiers (e.g. DOIs), or URLs. The simplest implementation consists of a traditional PDF article with relative metadata that together become an EP with reference parts thanks to the addition of dedicated metadata fields containing references to external objects. This solution is generally low cost as it marries the traditional file-metadata approach of Digital Libraries (DLs), but referenced parts can be discovered only via the article metadata. In other systems, reference parts are explicit EP sub-parts enriched with metadata information inherited (somehow collected) from the referred object, thereby enabling the proper consumption (e. g. discovery, visualization) of the part. For the same purposes, in some cases the semantics of relationships between sub-parts is provided. (e. g. SURF EPs [159]).

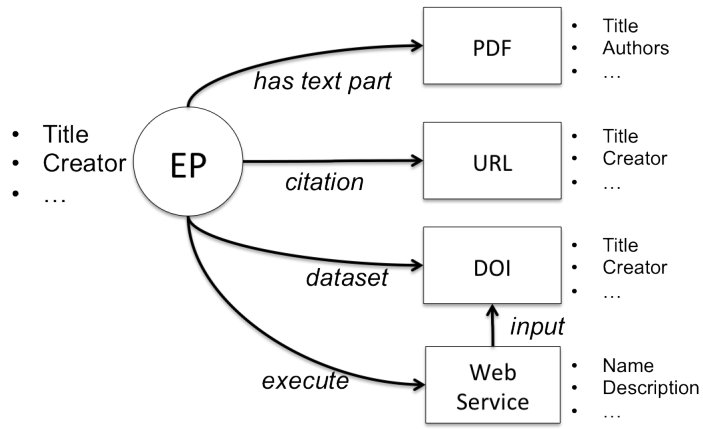


Fig. 3.5. EP with reference parts

For EPs with reference parts the supporting information systems might have to face the issues of "broken links". This is the case when the objects identified by such references are no longer available for any reason. As a result, the EPs may become inconsistent and recovery or invalidating policies may be applied.

Executable parts

Data-intensive e-Science brought in the novel requirements of disseminating traditional digital publications with a "research experiment context", which would allow for better interpretation and validation by-repetition of the research conducted. EPs with reference parts to datasets certainly represent a step ahead in this direction, but are not enough to address these needs. Indeed, in order to share an experiment, scientists should make available to the community both data and processes they used. To this aim, information systems capable of managing and consuming EPs with executable parts have been realized. Such parts carry information required to execute a process, such as a reference to a web service used in an experiment, a workflow to be executed by a given engine, or, more generally, code which can be dynamically executed by a given run-time.

The most important studies on enhancing publications with executable parts for the purposes of supporting peer-review, research validation and re-usability have been conducted in the context of Virtual Research Environments (VREs). VREs are defined by Joint Information Systems Committee (JISC) as a digital, distributed platforms that enable "collaboration between researchers and provide access to data, tools and services through a technical framework that accesses a wider research infrastructure" [46]. VREs offering functionalities for the planning, execution and sharing of in-silico experiments are also referred to as e-laboratories. Examples of e-laboratories are myExperiment [142], Collage [122], IODA [144], SHARE [65], D4Science [41].

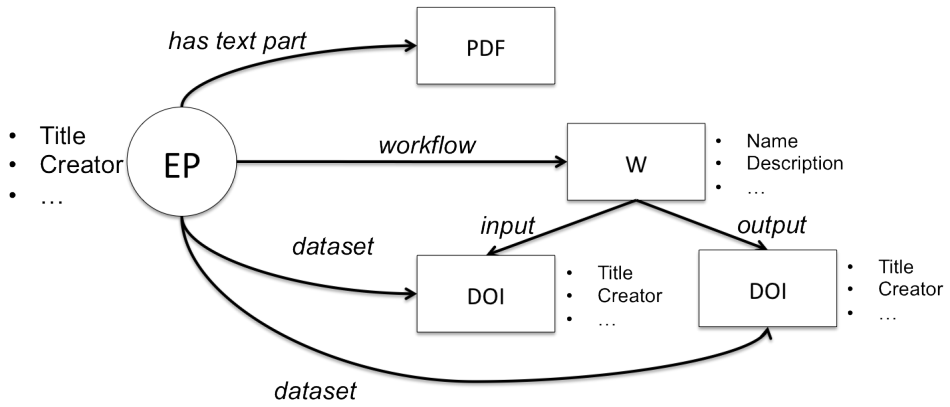


Fig. 3.6. EP with executable parts

Fig. 3.6 sketches an EP about an experiment, described by a traditional PDF publication, relative to a workflow *W* that generated a given *output* dataset from a given *input*

dataset. To this aim, the EP includes all parts required to reproduce the experiment: the PDF, the workflow, the two datasets (or a link to their location) and their metadata information. The latter convey information required to execute the workflows and services in the proper way (e. g. the workflow engine, operating system configuration). Researchers, but also reviewers, are therefore in the condition of reproducing the experiment and compare the results with those presented in the article, thus validating the research results. In addition, they can also apply the workflow *W* to their own datasets, hence effectively re-use tools produced by others or replicate the experiment.

Executable parts may be encoded in a variety of ways, which highly depend on the scenario supported by the information system that generates, stores and offers access to such EPs:

- The system hosts the processes themselves: parts point to an executable process (e. g. web service);
- The system relies on third-party execution environments: parts contain code to be executed (e. g. workflows in myexperiments.org [142]);
- The system relies on the ability of researchers to install and deploy software: parts include software (e. g. SVN references), how-to manuals, or configurations (e. g. virtual machine images [65]).

Generated parts

In Elsevier's Article of the Future [9] an EP contains reference parts that link to external databases, e. g. molecule entries in a chemical database. Such parts can be processed by an application of the information system so as to dynamically generate new parts of the EP. For example, a molecule can be processed to generate its 3D rendering. As such, the rendering is not a static sub-part of the publication, but rather a dynamically generated part. Another example of generated parts is that of the EP in figure 3.7, inspired to the "live documents" proposed in [42]. The publication includes a "scientific report" and a pointer to a dataset in a database, whose data is constantly updated. The scientific report includes a dynamic table whose content can be generated by running a given query over the dataset. When users visualize the scientific report, a local application executes the query and processes the results to generate the table with the content currently available in the dataset.

Generated parts can be defined in terms of:

1. The "static" parts of the EPs used as inputs (e. g. the reference to the molecule, the reference to the database dataset, the query);
2. The application used by the information system to consume the existing part (e. g. an application to generate 3D models of molecules, a software capable of sending the query to the database and generate the table from the result).

All the parameters and configuration needed to dynamically create a generated part are either in the status of the information system or stored with the metadata of the already

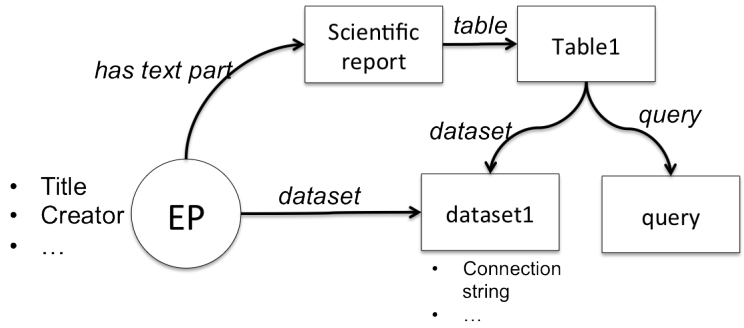


Fig. 3.7. EP with generated parts

existing parts. As a consequence, generated parts are tightly coupled with the information system at hand, with all the re-use limitations that such an approach may entail.

3.2 Enhanced Publication Information Systems (EPISs)

An EPIS offers functionality for the management of EPs to a user-community. The EPIS designers specify, based on the input from the target user community, the requirements of the system in terms of:

1. *Content requirements*: expected typology (structure and semantics) of EPs;
2. *Functional requirements*: expected functionality for the management and consumption of EPs.

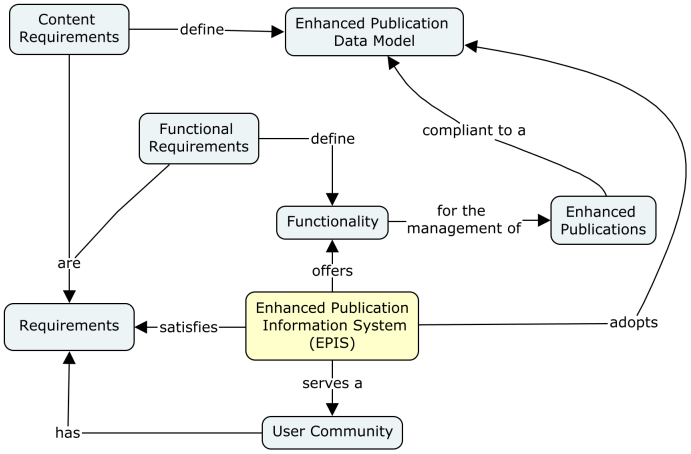


Fig. 3.8. Enhanced Publication Information Systems (EPISs)

In the following, the typical issues and challenges to be tackled by developers for the realization of EPISs satisfying a given set of content and functional requirements are discussed.

3.2.1 Addressing content and functional requirements

Content requirements specify the structure and semantics of the EPs, i. e. the EP data model, the EPIS should provide functionality for. EPIS designers conceptually define the EP data model by describing the structure and semantics of the parts the target users are interested in to form EPs. The EP data model can be considered as the "core" element of an EPIS: all functionality, tools and services offered by an EPIS assume that EPs are compliant to one data model. In fact, designers and developers must design and implement end-user functionality on top of the defined EP data model.

An EP data model is a graph model where different types of parts (graph nodes) are connected to each other via semantic relationships (labelled edges). Accordingly, each EP is itself a graph of parts, which can be connected to other graphs and form a forest of graphs via links between parts belonging to different EPs. Also, relationships between EPs can occur, if the data model allows them. Example 2 shows a possible conceptual data model (b) for the EP of Example 1 (a).

EPISs should be able to capture the graph nature of EPs: developers should implement all functionality, both management (e. g. creation, update, delete) and consumption (e. g. search, browse, access) functionality, keeping in mind that the digital objects to be manipulated are not simple, flat objects, but they are rather complex objects composed of many parts.

Management functionality

EPIS management functionality includes functionalities for creating, updating and deleting EPs.

Creating EPs

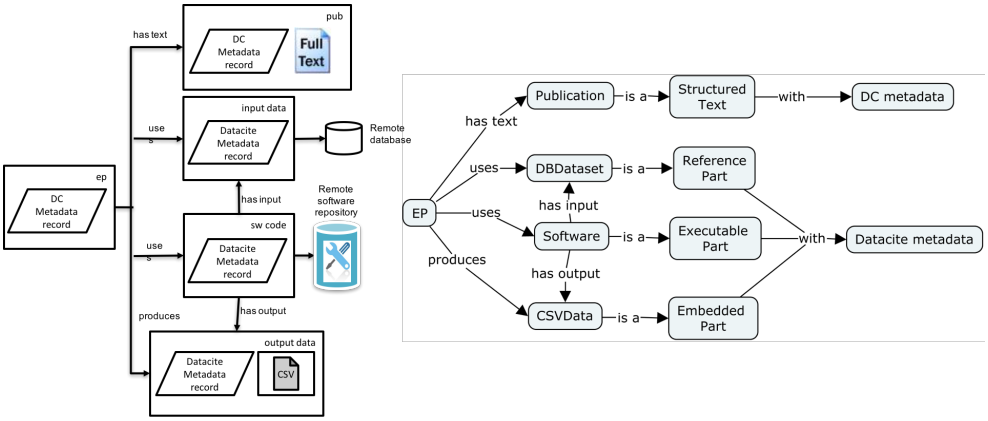
EPISs can enable users to create EPs with different strategies:

1. Via manual creation of parts (e. g. text-processing Graphical User Interface (GUI) for the creation and editing of scientific publication) and relationships;
2. Via manual upload of existing research products (files and metadata information);
3. Via selection from a set of external data sources.

The third option is particularly useful to enable users to create EPs by composing existing research products that are available on the internet or accessible from known data sources. In these cases users do not need to create or upload research products into the EPIS, but they just need to identify the products and provide a reference such as their identifier relative to the data source, their Uniform Resource Locators (URLs), or Persistent

Example 2 Example of EP and one possible data model

The EP of Example 1 (also in figure a) below) is compliant to the conceptual data model (b), expressed in terms of the data model features presented in Section 3.1 and Figure 3.2.
The conceptual data model specifies which are the types of parts that EPs can aggregate and which are the relationships that can be established among EPs and parts. *Publication* is a type for narration, structured-text parts. *DBDataset* is a type for modelling references to datasets hosted in remote databases. *CSVData* is a type for modelling CSV files embedded into the EP (i. e. embedded parts are discoverable and accessible only from within the containing EP). *Software* is a type for modelling executable software code. The model also specifies that *Software* instances can have instances of *DBDataset* as input and instances of *CSVData* as output.



a) A sample EP instance (see Example 1) b) A possible conceptual EP data model

Identifiers (PIDs) (e. g. Digital Object Identifiers (DOIs)). The community-specific nature of the EPIS allows designers and developers to build dedicated tools for the interaction with the data sources of interest for the community. For example, if the EPIS is serving a community in the domain of Life Science, interesting data sources are databases of biological entities (e. g. UniProtKB [16], European Nucleotide Archive [101]), Life Science literature repositories (e. g. Europe PMC [112], PubMed Central [141]) and biological ontologies (e. g. Gene Ontology [13], NCBI Taxonomy [58]). If instead the EPIS is serving a community of social scientists, the interesting data sources are archives of social science studies (e. g. DANS[1]), national and international surveys (e. g. International Social Survey Programme [148], European Social Survey[2]) and literature repositories in the field (e. g. Social Science Open Access Repository[3] , Social Science Research Network[4]). EPISs should be able to communicate with the interesting data sources in order to enable users to search for interesting research products to include as parts of EPs.

[1] Data Archiving and Networked Services (DANS): `https://easy.dans.knaw.nl`
[2] European Social Survey (ESS): `http://www.europeansocialsurvey.org`
[3] Social Science Open Access Repository (SSOAR): `http://www.ssoar.info`
[4] Social Science Research Network (SSRN): `http://www.ssrn.com`

The three strategies for the creation of EPs are generic as they do not depend on the specific application domain of the target community of the EPIS. Their configuration is instead domain-specific because formats of files and metadata and the data sources to integrate depend on the specific domain of the community: dedicated software code is needed for their integration. However, EPIS developers use to implement the whole creation strategy from scratch due to the lack of EP-oriented tools providing configurable strategies for EP creation.

Updating EPs

The possibility of updating an EP is of great benefit to users that want to keep enriching the EP with new related research products that were not known, or not available, at the time when the EP was created. Also, EPs can be considered as dynamic resources that can change over time to reflect the advancement of scientific knowledge. For example, new studies may invalidate a previous research represented as EP and an EPIS may allow users to add a new semantic link between the EP and the new research, so that it is clear to future consumers that the scientific knowledge in the first EP has been invalidated by another study. Finally, humans can make mistakes when creating EPs and the EPIS should make it possible to recover from those mistakes. In all cases, the EPIS should deal with authentication and authorization of users: depending on the policies of the community, EPs could be up-datable only by its creator(s) or by a selection of authenticated users (i. e. the EPs curators). In other cases, instead, an opener policy can be preferable, which enable the "crowd" curation of EPs by allowing any user to perform (or suggest) updates to existing EPs. For the implementation of authentication and authorization functionality, EPIS developers can implement a layer over standard Auhentication and Authorization Services (AASs) such as Lightweight Directory Access Protocol (LDAP), or possibly exploit built-in functionality of well known Content Management Systems (CMSs) such as Joomla[5] or Drupal[6].

Deleting EPs

Like for the update of EPs, an EPIS should deal with user authentication and authorization for the deletion of EPs, according to the policies decided by the community. Apart from that aspect, the EPIS should carefully consider the graph structure of the EP when a deletion of a whole EP or one of its parts is performed. First of all, parts of one EP can also be parts of other EPs. Consequently, when a part is deleted from one EP, the EPIS should not remove the part from the system, but rather "unlink" the part from the EP at hand (i. e. delete the relationship, not the target of the relationship). Accordingly, the deletion of the whole EP should not trigger the deletion of all its parts, but rather apply a strategy for the deletion (or non deletion) of all the relationships created in the context of the EP. If we consider, for example, the deletion of *EP1* in Figure 3.9, it is clear that all relationships in

[5] Joomla!: `https://www.joomla.org/`
[6] Drupal: `https://www.drupal.org/`

the outbound star of *EP1* must be deleted. However, it is not straight forward to identify the better strategy to apply for the relationships between the parts. The relationship between *input data* and *sw code 1* existed in the context of *EP1* and, as such, should be deleted. On the other hand, the deletion of the EP does not imply that the two research products are not related any more and, in such case, the user maybe would expect the relationship not to be deleted. Usually, the strategy adopted by one EPIS depends on the use-cases of the target community, as they are reported in the EPIS design phase.

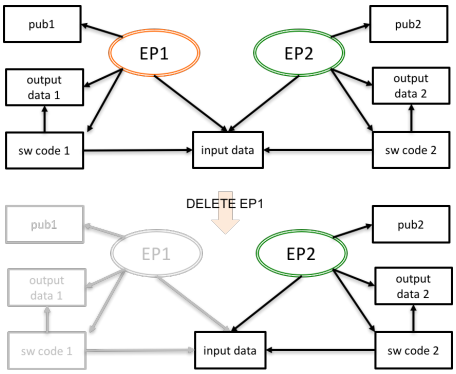


Fig. 3.9. Deletion of an EP

Consumption functionality

Consumption functionality for sharing, accessing and executing EPs and their parts are the main driving factors to the realization of EPISs. Depending on the best practices in use at the research community to serve, different techniques and technologies can be adopted to offer a set of export and access modes to EPs for different target consumers, both humans and machines.

As shown in Example 2, each EP is a graph of parts that can be connected to other EPs, i. e. other graphs, to form a forest. For an effective implementation of consumption functionalities, an EPIS should then be able to:

- Capture the graph nature of EPs;
- Implement strategies to navigate the forest of graphs;
- Identify the boundary of each EP.

Figures 3.10 and 3.11 show two different strategies for the identification of EP boundaries. In figure 3.10 a strict boundary has been identified, so that one EP includes only the parts that are directly aggregated. Figure 3.11 shows instead another approach, where the boundaries of the EPs are identified by following all relationships between parts, until another EP instance is reached. It has to be noted that there is no optimal strategy for the identification of boundaries of EPs that can be adopted by every EPISs. Different strategies

may be useful to realize different functionality over the graph of EPs. EPIS could also be realised to support multiple strategies for the identification of boundaries and to allow consumers to specify the one to adopt.

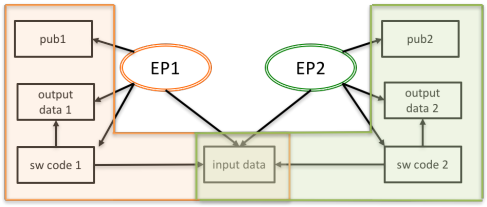


Fig. 3.10. Strategies for the identification of boundaries in a forest of graphs of EPs: strict boundary includes parts that are directly connected to the EP

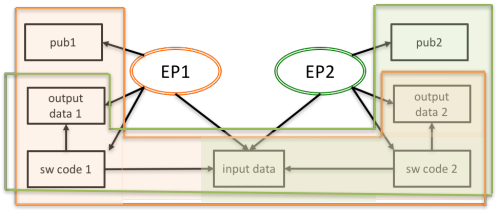


Fig. 3.11. Strategies for the identification of boundaries in a forest of graphs of EPs: the boundary of one EP includes every parts that are reachable from it, until another EP is found

The main access mode to EPs for humans is via GUIs that allow to:

- Search and browse EPs and their parts;
- Navigate the graph of EPs and explore existing semantic relationships between their parts;
- Download a package that contains all research products aggregated in one EP;
- Support the execution of executable parts.

In addition to humans, machines (e. g. third-party software) can also be interested in consuming EPs, thus EPISs might offer a set of Application Programming Interfaces (APIs) to offer different kinds of access to the graph of EPs:

- Bulk download: retrieve the whole graph of EPs or a known sub-graph of it;
- Selective access: perform searches on the graph of EPs and retrieve only the EPs, or parts of EPs that satisfy the search criteria;
- EP graph navigation: explore the graph of EPs by navigating through the existing relationships.

Bulk download

A bulk download access mode is useful to clients willing to get a copy of the graph, or a sub-graph, of EPs. A typical use case is that of clients that aggregate metadata records to establish a common entry point from which the users can search for content across different data sources. In the scientific communication domain, it is very common that information systems support those clients with the standard Open Archive Initiative - Protocol for Metadata Harvesting (OAI-PMH) protocol. The Open Archive Initiative - Protocol for Metadata Harvesting (OAI-PMH) protocol [97, 45] is particularly relevant to enable metadata interoperability among digital libraries, institutional and thematic repositories. OAI-PMH provides an application-independent interoperability framework based on metadata harvesting. Two classes of actors participate in the OAI-PMH framework: data providers, which expose metadata records via an OAI-PMH publisher service over Hypertext Transfer Protocol (HTTP); and service providers, or aggregators, which harvest metadata records from data providers. Metadata records exported by providers can be grouped into OAI sets, so that harvesters can collect only a portion of the exported metadata records. The protocol also mandates that metadata records must be available at least in Dublin Core (DC) XML format.

With the advances in Semantic Web, the Resource Description Framework (RDF) gained importance and visibility also in the scientific communication domain. RDF [54] is a standard model for data interchange on the Web. In RDF every item of interest (called "subject") is identified by a Uniform Resource Identifier (URI) and can be linked to other items (called "object") via semantic relationships (called "predicates"), identified as well by URIs. RDF triples (subject - predicate - object) can be serialised in XML, N-Triples [23], RDFa[7, 69], JSON-LD [145], or TriG[25], enabling the bulk export of the graph of EPs as RDF dumps.

Selective access

A selective access API is useful to clients willing to find and access EPs and parts that satisfy specific search criteria such as the author's name, subject keywords. Selective access API typically allows to find matches of the search criteria in:

- Metadata records of EPs and their parts;
- Content of textual parts of EPs (e. g. structured-text parts, PDFs).

Let's consider the data model in Example 2 and the graph of EPs in figures 3.10 and 3.11: a selective access API may enable clients to find all EPs or all parts of type *Software* using *input data*. Selective access API usually have custom syntax in terms of accepted parameters and response format, although some standards can be used to ease the realisation of software clients such as OpenSearch[7], OpenAPI[8] and Contextual Query Language (CQL) [152].

[7] OpenSearch, `http://www.opensearch.org`
[8] Open API Initiative, `https://openapis.org`

EP graph navigation

Accessing EPs by navigating the graph is useful to clients that want to find and access EPs and parts that satisfy search criteria on the existing relationships. Also, navigation is a powerful approach to discover a graph without knowing its structure in advance. There are standard techniques to expose a graph in such a way clients can easier navigate through its relationships and nodes. The most relevant for the scientific communication domain are based on Linked Data. The term Linked Data [26, 76] refers to a set of best practices for publishing and interlinking structured data on the Web. The principles were first introduced by Tim Berners-Lee in [24]:

1. Use URIs as names for things.
2. Use HTTP URIs, so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL Protocol and RDF Query Language (SPARQL)).
4. Include links to other URIs, so that they can discover more things.

The Open Archive Initiative - Object Reuse and Exchange (OAI-ORE) protocol [98, 95, 96] adopts the above principles and defines standards for the description and exchange of aggregations of web resources. Through OAI-ORE, clients can access and navigate through the resources aggregated by an aggregation via HTTP. However, it does not allow to perform advanced graph traversal on the graphs. This functionality is instead implemented by SPARQL endpoints. SPARQL is a set of specifications that provide languages and protocols to query and manipulate RDF graphs in triplestores.[9]

It should be now clearer that EPIS developers and designers must consider a plethora of details when realising an EPIS, even details that are not community-specific but are instead general requirements that must be satisfied in order to support the implementation of domain-specific requirements. In addition, it has been highlighted how content and functional requirements are close to each other: some functionality cannot be realised if the data model is not designed to support them. As a consequence, adding and refining functionalities may require data model updates and, in some cases, changes to the data model may invalidate the correct operation of existing functionalities so that EPIS developers must update also their implementation.

3.3 Survey of EP data models and EPISs

The main motivations behind EPs are to be found in the limits of traditional scientific literature to describe the whole context and outcome of a research activity. The goal is to move

[9] An introduction to the concept of triplestores (or RDF stores) is available on Wikipedia at `https://en.wikipedia.org/wiki/Triplestore`, where triplestores are defined as *A triplestore or RDF store is a purpose-built database for the storage and retrieval of triples through semantic queries. A triple is a data entity composed of subject-predicate-object, like "Bob is 35" or "Bob knows Fred"*.

beyond the simple PDF [31, 4] to support scientists with digital and automated access to the literature and any form of research outcome (e.g. research datasets, ontologies), still without losing the narrative spirit of "the publication" as a dissemination mean.

A wide variety of information systems for EP are available out there, which provide communities of scientists with selections of functionalities handling EPs conforming to the most variegated data models. More specifically, such systems offer a set of management functionalities for the creation, deletion and updates of EPs, and a set of consumption functionalities for the usage of these publications, e. g. reading, sharing, executing.

As to management functionalities, EPs are typically created (deleted and updated) via end-user interfaces that guide end-users at providing publication parts and specifying relationships between them. Depending on the data model, parts may be provided by uploading files from the file system or providing references to files (e. g. URLs, DOIs). In some cases sub-parts may be identified or generated at run-time by the applications. In other cases they may be produced by systems (e. g. scientific infrastructures, virtual environments) for example to provide the machine-executable parts necessary to share a repeatable experiment.

Consumption functionalities, being the driving requirements and motivation for this field, require more attention. Their focus vary and may include for example: exporting EPs as packages of parts (i.e. "compound objects"), improving readability of the narrative text via web interfaces or clients (e. g. navigating and visualizing sub-parts), browsing/accessing parts following relationships (e. g. links from publications to datasets), machine execution of parts (e. g. execution of a dataset processing workflow), etc.. In particular, four main functional goals of EPISs related to the consumption of EPs have been identified (Figure 3.12):

1. Providing advanced reading facilities;
2. Packaging related research products;
3. Interlinking research products;
4. Supporting re-production and assessment of scientific experiments.

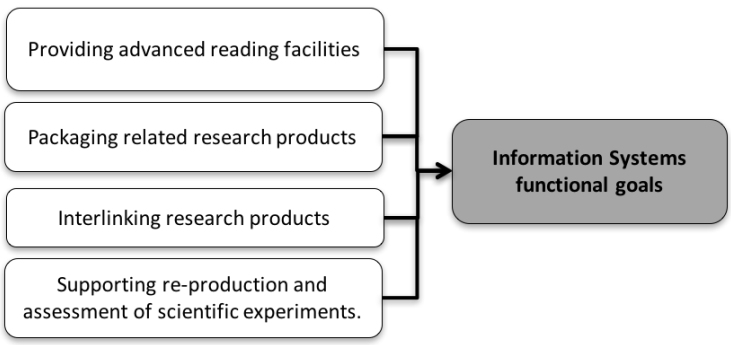


Fig. 3.12. The main four functional goals of EPISs

An EPIS can address all the functional goals above or focus only on some of them, depending on the requirements of the target community. In any case, the adopted EP data model has specific features that enables the EPIS to support its functional goals. In the following, a selection of EP data models and EPISs in the literature are presented in terms of their features and the main functional goal that they support.

3.3.1 Providing advanced reading facilities

Since the beginning of the digital era, agents of scholarly communication understood that digital publications could be enhanced to offer advanced reading facilities [99, 113]. However, stakeholders preferred to retain the paper layout of the publication and only introduced a minimal set of the expected changes. Those changes were basically related to the possibility of creating activable web links, so that readers could open a web page by clicking on the link on the downloaded PDF. Some experimentations started in the '90s to deliver publications in HTML instead of PDF, but only later the HTML view became a common option in journals' web sites. The HTML version is typically richer than the PDF version and integrates all tools made available by the web infrastructure and its data sources [85], such as DBPedia, ontologies, public domain-specific databases.

Specifically, publications are enhanced to offer advanced reading facilities by:

- structuring the narrative text into interconnected sub-parts to ease navigation through sections and editorial parts of the article;
- re-using the universe of web resources to enrich the text (e. g. show definition of terms from DBpedia, Wikipedia or domain-specific ontologies, taxonomies, and vocabularies);
- including dynamic forms of content within the text (e. g. 3D structure of a molecule, table with dynamic content);
- generating the graph of citations;
- suggesting related publications based on content and subject similarity.

Advanced reading facilities support strategic and horizontal reading,[10] by making it easier for readers to:

- identify the different parts of the text of a publication;
- find other relevant publications that might be of interest;

[10] A strategic reader is a reader who has a reading task and applies reading techniques to complete the task effectively without reading the whole text at hand. One typical technique is reading sections in relevance order, rather than in narrative order. For example, if a researcher is interested in the methodology, he or she jumps directly to that section, and reads the other sections only after, if needed. The same technique can also be applied to paragraphs, in order to read first the sentences that look more relevant for the reading task [138, 150]. Another widely adopted technique is "horizontal reading" [121, 120]. In horizontal reading, the reader does not focus on one text, but rather reads (possibly strategically) many texts, jumping from one to another until the reading task is reached.

- access the information they are looking for, not only in the traditional textual version, but also in more direct and user-friendly visual representations.

In addition, the enrichment of text with definitions from domain-specific web resources support readers that have not a deep knowledge of the topic of the publication they are reading. In these cases, also definitions from more generic web resources such as on-line encyclopaedia may be effective to introduce jargon terms to newcomers.[11]

Information System	Embedded parts	Structured-text parts	Reference parts	Generated parts	Executable parts
D4Science	X	X	X	X	
SciVee	X	X			
PLOS Neglected Tropical Diseases	X	X	X	X	
Veteran Tapes project	X	X			
Utopia documents	X	X	X	X	
Rich Internet Publications	X	X	X		
SOLE		X	X	X	
Article of the future	X	X	X	X	
Bookshelf	X	X	X		

Table 3.1. Providing advanced reading facilities: information systems and data model features

The D4Science infrastructure and live documents

The notion of "live documents" introduced in [42] was implemented in the D4Science [43] infrastructure, a research infrastructure supporting the iMarine project,[12] whose community is composed of fishery scientists. Live documents consist of textual publications (typically research reports) that embed data descriptions, tables, histograms, summaries, and statistics based on "live data". Live data is data generated at access time and updated in the publication by the underlying infrastructure. A publication can therefore be "instantiated" in a given moment in time to describe current status/results for a given scenario.

[11] Online encyclopaedia such as Wikipedia and DBPedia may be useful depending on the domain. For example, studies about the quality of Wikipedia entries related to health shows that Wikipedia is an appropriate source of information for nursing, healthcare students [72, 135]. In the fields of gastroenterology and hepatology instead, Wikipedia articles have been considered not reliable for students [15].

[12] The iMarine project web site: `http://www.i-marine.eu`

SciVee

SciVee [59] is a system that allows authors to create EPs by uploading an article they have already published and a video or podcast presentation that describes the highlights of the paper. The author can synchronize the video with the content of the article (text, figures, etc.) so that the relevant parts of the article appear as the author discusses them during the video presentation. Images are extracted from the publication and available to the user in a dedicate viewer. However, the system does not process the narrative part of the EP to identify sections, references, etc.: the narrative textual part is still visualized as a single block of text and the system does not offer browsing and searching functionalities.

PLoS Neglected Tropical Diseases

Shotton, Portwin, Klyne and Miles experimented different enhancements for research articles published in PLoS Neglected Tropical Diseases in order to improve the user reading experience [143]. Articles are enriched with semantic tags that identify keywords (e.g. names of diseases, organisms, proteins) and interesting entities such as institutions, dates, and persons. Moreover, interactive tables, figures, and maps are also provided.

The Veteran Tape Project

The Veteran Tapes project experimented EPs In humanities [155] Researchers were asked to manually create some samples of EPs. Researchers were provided with a web application to select sound fragments of interview to link to phrases of their publication. The resulting EP appears to the reader as a traditional digital publication, but parts of the narrative text are associated to audio fragments and highlighted for easy recognition. The reader can click on those parts to view the metadata and the transliteration of the audio fragment. An audio player is also provided for listening to that specific fragment of the interview.

Utopia Documents

Utopia Documents [14] is a desktop PDF reader that integrates visualization and data-analysis tool with published research articles. Utopia Documents has been used by editors of the Biochemical Journal to curate pre-prints by annotating terms with links to online resources such as Wikipedia, UniProt entries, and ontologies. By exploiting the content of supplementary material attached to a publication and content of related datasets, the software is also able to provide dynamic views of tables and images.

Rich Internet Publications

Breure, Voorbij and Hoogerwerf [35] presented the concept of “Rich Internet Publications” (RIPs). A RIP is an online publication composed of narrative text, images, videos, links to data and other material. Text has not a major role with regards to the other components: all parts are “first-class citizens” in RIP’s data model. The importance of one specific

part of a RIP just depends on the actual reader, who might prefer to start his or her reading from an image, rather than from the description of the data. Since all components are “peers”, users have the possibility to choose non-linear (i.e. not guided by the text) reading. Breure et al. support their interpretation by providing several real-case examples in the cultural heritage domain.

SOLE

SOLE [129] is a tool for linking research papers with associated science objects, such as source codes, datasets, annotations, workflows, packages, and virtual machine images. Authors of SOLE are investigating the possibility of enabling re-use of datasets linked by a SOLE document via dedicated services.

Article of the Future

Elsevier presents the so-called “Article of the Future” [9], where a publication is structured in well-defined parts (e.g. sections, figures, tables, images, links to external data or applications) to provide end-users with advanced visualization functionalities such as interactive charts and tables, access to geospatial data on Google Maps, 3D fossil reconstruction and chemical compound viewers. The underlying technology mines the narrative parts to identify chemical and science material compound terms (keys to external databases) and assigns them to web services that display detailed information or graphical representations.

Bookshelf

Bookshelf [78] is a system for life science and health care literature resources managed by the National Center for Biotechnology Information (NCBI). Documents in Bookshelf are represented in XML format and are tagged by curators with references to several molecular databases such as OMIM and GenBank. Publications are enriched with supplementary files, structured-text parts that enable advanced navigation of the publications, and links to molecular database entries added by curators.

3.3.2 Packaging related research products

The goal is to create a package consisting of a digital publication and other research products that are useful to improve the readers’ comprehension of the research described in the publication. For example, a social scientist may package his/her publication with the surveys he/she analysed and discussed in the publication. As a consequence, interested readers can not only learn about the results of the author, but also validate the author’s conclusion by accessing the original surveys or even re-use the surveys to analyse aspects that were not considered by the original author.

Publishers intercepted this need relatively soon and allowed authors to submit supplementary material along with the publication [157]. Supplementary material is typically peer reviewed together with the publication and made available for download from

the web site of the journal. Supplementary material welcomed by publishers includes presentation slides, appendixes to the article, data and description of data used for the research described in the article, high resolution images, and tables that could not be inserted fully in the digital publication because of page limits. Supplementary material is typically not discoverable and not accessible outside the context of the related publication, i. e. users can find and access the additional research products only after they have discovered and accessed the article.

Information System	Embedded parts	Structured-text parts	Reference parts	Generated parts	Executable parts
Journals with policies on supplementary material	X	depends on the journal	depends on the journal	depends on the journal	
Modular article	X	X	X		
LORE			X		

Table 3.2. Packaging related research products: information systems and data model features

Journal Information Systems

Several scientific journals (e.g. publishers Elsevier, SAGE journals) offer authors the possibility to upload with their article any relevant material that is too big or that does not fit the traditional article format or its narrative, e.g. datasets, multimedia files, large tables, animations, code, etc.Scientists accessing the article online may also download such material to complete their understanding. Supplementary material is typically stored locally into the information system of the journal and it is not discoverable and not accessible outside the context of the related article, i. e. users can find and access the material only after they have discovered and accessed the article.

Modular articles

Kircz proposed in [93] the“modular article” data model. A modular article aggregates and connect with meaningful links several “modules” in order to create a coherent, self-contained, and complete unit describing a research. The modular structure facilitates re-use and re-purposing of modules and enables the realization of web-oriented publishing/reading tools. A module describes, with a specific set of metadata, an entity related to a research. The proposed modules are: meta-information (i.e. bibliographic info, structure of the article, classification terms, references, acknowledgements, abstract), goal and setting (i.e. problem definition, methods, techniques, and goals), results (i.e. raw and fitted data), discussion, and conclusions. Modules may describe/include different types of entities, which reflect the kinds of supplementary materials usually submitted along with a publication, such as: sounds, videos, data sets, and images.

LORE and compound objects

LORE [63] is a plugin for Mozilla Firefox that allows creating "compound objects". Compound objects are here intended as EPs composed of multiple Web resources and related bibliographic records. The main goal of LORE is that of supporting e-Learning in humanities. Teachers and researchers are provided with an easy to use tool to create packages of related resources. Those packages are easy to export and share on the Web, helping students in finding interesting self-contained resources about a specific topic or a course. Resources composing a compound object are related by typed relationships defined in a dedicated and configurable OWL ontology.

3.3.3 Interlinking research products

The attachment of research data as supplementary material of the digital publication is considered not a feasible practice for several reasons [131]. Publishers typically have limited disk space, hence the storage and preservation of large data sets is an issue. In addition, it is uncommon for publishers to perform data curation (e. g. migration of the format of datasets as soon as the original format becomes obsolete), so that the usability of the data is not ensured over time. In summary, publishers don't want to play the role of data centres, which, in turn, offer data management functionalities ensuring storage, discoverability, accessibility, and usability of data over time. Publishers are also concerned about the peer review of supplementary material. Most publishers claim that supplementary material is peer reviewed together with the relative article. However, they are concerned about the quality of the peer review and the effort needed by reviewers to check all the material. For example, the Journal of Neuroscience [111] and the Journal of Experimental Medicine [30] changed, in 2010 and 2011 respectively, their policies to drastically limit the presence of supplementary materials in their journals.

Today, the trend in many disciplines (e. g. life science, environmental sciences, astronomy, archaeology) is to deposit research data in a data centre (or repository) [132] or a discipline-specific database and add the link to the deposited data in the publication. According to this trend, the Public Library of Science (PLOS) updated its data policy and since March 2014 encourages authors to make research data openly accessible from public data centres and requires them to submit a "data availability statement" containing information on how to access the data [27].

Table 3.3 shows some of the most used data centres and the target disciplines they serve.

Scientific communities, organizations, and funding agencies are nowadays supporting and welcoming initiatives, standards and best practices for publishing and citing datasets and publications on the Web [137, 115]. Examples are DataCite[3] and CrossRef[1], which establish common best practices to assign metadata information and persistent identifiers to datasets and publications. Data publishing and citation practices are advocated by research communities, which need datasets to become first citizens of the

Data centre or database	Target disciplines
Archaeology Data Service [139]	Archaeology
Array Express (gene expression database)[34]	Life science
British Atmospheric Data Centre (BADC) [8]	Environmental science
Data Archiving and Networked Services (DANS) [2]	Arts, humanities and social sciences
DataOne [11, 53]	Earth and environmental science
Dryad [156]	Life science
Pangaea [55]	Earth and environmental science
Uniprot (protein database) [52]	Life science

Table 3.3. A non-exhaustive list of domain specific data centres and databases for the deposition of research data

scholarly communication chain. As claimed by Callaghan [38] and Parsons [127], datasets should be discoverable and reusable, while scientists who produce and share them should be scientifically rewarded.

According to this context, publications have been enhanced with links to research data that is stored and maintained at a different location. However standards and best practices are not yet universally adopted.

Several solutions for EPs are investigating the possibility of identifying relationships between publications and datasets either prior or post publishing. In case of post publishing, links between publications and research data are identified after the publishing step, either by humans or machinery (e. g. by text mining the narrative part of the publication). In case of identification of links prior publishing, authors include in a specific section of the publication text (such as the “data availability statement" in PLOS) or in the metadata the (persistent) identifiers (e.g. URLs, DOIs) of the related datasets or database entries. This is often possible thanks to reciprocal agreements between data centres and scientific journals, which mandate dataset archiving practices.

In the Life Science domain there is a long tradition of integration between articles and research data thanks to a number of agreements among journals in the field and biological databases. The first partnership was established in 1988 between the journal Nucleic Acids Research and the EMBL Data Library [90] and, since then, a number of important journals (e. g. Nature) require authors to submit data (e. g. DNA sequences, protein structures) in dedicated public databases and to cite the data in the submitted article with the identifier assigned by the database. Moreover, curators of the databases are usually in charge of keeping up to date the information about the deposited data, including the inverse link to the corresponding article. The above practice is today a common practice and de-facto standard in the Life Science community. More recently, in 2011, many journals, mainly in the field of evolution, adopted the Joint Data Archiving

Policy (JDAP)[6], which states that archiving data in a public archive is mandatory for an article to be published.

Information System	Embedded parts	Structured-text parts	Reference parts	Generated parts	Executable parts
SCOPE		X	X		
CENS ORE aggregations			X		
EuropePMC	X	X	X		
WormBase			X		
DANS Enhanced Publication Project			X		
BioTea		X	X	X	
OpenAIRE multi-disciplinary enhanced publications			X		

Table 3.4. Interlinking research products: information systems and data model features

SCOPE and scientific compound objects

SCOPE (A Scientific Compound Object Publishing and Editing System)[50] is a system designed to enable scientists to easily author, publish and edit scientific "compound objects". These are EPs encapsulating (or referring to) datasets and resources generated or utilized during a scientific experiment or discovery process. Such objects, whose structure is represented as an RDF named graph, are then published and exchanged to disseminate the overall context of a research activity.

CENS OAI-ORE

Pepe, Mayernik, Borgman and Van de Sompel (2009) presents an implementation of OAI-ORE [98] to aggregate publications, datasets, and metadata related to the seismology and environmental sciences at the Center for Embedded Networked Sensing (CENS). Entities are linked to each other via relationships whose semantics belong to the scholarly and scientific life cycle. The goal of the system is to link resources already published on the Web.

europePMC

europePMC, formerly known as UKPMC [112], is an information system that aggregates life science abstracts and Open Access full-text publications from several sources. Publications are enriched with references to entries in well-known biomedical databases (e. g. UniProt, ENA, PDB), ontologies and taxonomies (e. g. NCBI Taxonomy). Relevant entries are identified by applying text-mining techniques to the full-texts [136]. The data model of

europePMC supports EPs that consist of (*i*) one structured-text narrative part composed of abstract, sections, and bibliography; (*ii*) supplementary material, as it was originally delivered by the authors to the publisher; and (*iii*) a list of references to database, taxonomy and ontology entries.

WormBase

WormBase [160] is an information system that integrates a database about roundworms with research literature, genomic sequences and other biology related aspects. WormBase applies text-mining techniques to research articles to identify keywords in the published literature and links WormBase entries to the relevant publications. Collaborations with journals are currently active in order to support "back-links", from literature to WormBase entries, and increase the discoverability of both database entries and publications.

GreyNet and the EPs Project

GreyNet (Grey Literature Network Service) and DANS (Data Archiving and Networked Services) co-operates for the EPs Project [57]. The goal of the project is to enhance GreyNet's collections of conference pre-prints with links to the underlying research data. Full-text available on GreyNet can be linked to research data archived in the online archiving system of DANS or any other public data archive. DANS ensures to maintain bi-directional links in order to support discoverability of research data (full-texts in GreyNet link to DANS entries) and of literature (research data stored in DANS link to the related full-texts in GreyNet).

Biotea

The Biotea platform [61] aims at integrating scientific literature with the Web of Data. Biotea transforms XML encoded articles into the RDF format. RDF files are then enriched with machine-generated annotations about: (*i*) the structure of the article (e.g. sections, paragraphs.); (*ii*) references to biological databases, and (*iii*) names of entities corresponding to well known ontologies. Biotea features a prototype GUI for the visualization of documents where enriched content is displayed in an "interactive zone" (e.g. molecule viewer, additional information about a protein).

OpenAIRE and the EP demonstrators

In the context of the OpenAIRE scholarly communication infrastructure, Hoogerwerf et al. have built demonstrators of multi-disciplinary EP information systems in the fields of Social Sciences, Humanities, and Life Sciences [79]. EPs are created by researchers who select a to-be-EP from the OpenAIRE infrastructure information space (an aggregation of publication metadata from European institutional repositories) and identify links to datasets in several disciplines via semi-automated user interfaces. The EP is assigned a new identifier and relative metadata (the author is the researchers who created the EP), and can be searched for and visualized by other researchers.

3.3.4 Supporting re-production and assessment of scientific experiments

The availability of research data does not imply that research results can be effectively assessed or that research investigations can be reproduced: scientists and reviewers should be equipped with the tools necessary to repeat and replicate the experiment [21].

The publication can be enhanced with executable elements carrying information required to execute a process, such as a reference to a web service used in an experiment, a workflow to be executed by a given engine, or, more generally, code which can be dynamically executed by a given run-time. In some cases, sharing virtual machine images may be the only way to recreate the complete environment to enable the experiment reproducibility. In fact, this does not sound as a surprise in e-Science infrastructures, where researchers are urging for tools to disseminate and reuse the whole context of their research. To address such demands, several solutions were proposed in the literature on how to share executable components via enhanced publications. In enhanced publications with executable parts, narrative parts are accompanied by executable workflows with the purpose of reproducing experiments.

Information System	Embedded parts	Structured-text parts	Reference parts	Generated parts	Executable parts
Scientific Model Packages	X	X	X		X
myexperiment.org	X		X		X
IODA	X	X	X	X	X
Paper Mâché			X		X
Collage Authoring Environment	X	X	X	X	X
SHARE	X		X		X

Table 3.5. Interlinking research products: information systems and data model features

Scientific Model Package

Hunter [81] describes the concept of Scientific Model Packages (SMPs), and its evolution in Scientific Publication Packages [82]. SMPs are defined as compound objects that embed and relate heterogeneous components such as data, software, workflows, graphs, tables, or publications. Hunter also presents a VRE where scientists can create a workspace to share all materials about a research investigation. Scientists can then define SMPs by selecting components in the workspace. Different SMPs might be generated for different purposes (e.g. one for e-learning, one to support peer-review). An implementation where SMP are implemented and exported in RDF is also available [83].

myExperiment and research objects

In 2007, the project myExperiment.org, co-funded by JISC and Microsoft, was opened to the public. The goal of the project is to enable scientists to create, share, re-use and re-purpose workflows for data analysis. The earlier work on the design of the Virtual Research Environment (VRE) [142] led to the definition of the "Research Objects" data model [22]. A Research Object is defined as a "unit of knowledge" that aggregate resources related to a scientific experiment or a research investigation. Resources include publications, bibliographic metadata, results of an experiment, the data used and produced (or a link to it), methods applied to produce or analyse the data, and the people involved in the research. Named relationships link resources belonging to the same Research Object. The names of the relationships express the semantics of the associations in the context of the scientific investigation.

IODA and executable digital objects

IODA (Interactive Open Document Architecture) [144] is an XML-based data architecture for the creation of "Executable Digital Objects" starting from existing digital publications. IODA creates three layers over the publication: the data layer, the information layer, and the knowledge layer. The data layer identifies the structural entities of the article (sections, images, references, etc.) and enables the association of the paper with embedded or remote research data and executable code. The information layer allows authors to specify links between the structural entities and executable code (e. g. re-generate the plot by running this code when the user clicks on that figure). The knowledge layer allows authors to define links between parts of different Executable Digital Objects, thus enabling the construction of graphs of related Executable Digital Objects.

Paper Mâché and executable papers

The tool Paper Mâché [32] proposes a method based on virtual machines that provide an environment for authoring, reviewing, and publishing executable papers. Virtual machines, which include all the required tools and software setup for the reproduction of the experiment, are uploaded into the system with the publication. The virtual machine may also contain data (or a link to it), the required scripts and embedded code snippets to generate updated revisions of a paper and allow reviewers to trace back the steps and verify results of the authors. The data model features embedded, reference, and executable parts.

COLLAGE and executable papers

The Collage Authoring Environment[122] won the Executable Paper Grand Challenge organised by Elsevier in 2011. Collage is a VRE where authors can create and share with reviewers "Executable Papers". Authors can enhance their publication, thus transforming it into an Executable Paper, by embedding or linking to the primary research data and executable code. Executable code attached to a publication may be added to (*i*) enable

readers/reviewers to access primary data;(*ii*) enable interactive generation of figures and plots; (*iii*) allow the reader/reviewer to execute a computation.

SHARE

The second prize of the Executable Paper Grand Challenge went to SHARE (Sharing Hosted Autonomous Research Environments) [65]. The main point of SHARE is that whenever a publication is about on a new algorithm or software, the traditional publication is peer-reviewed, but the algorithm and software are not because reviewers might have non-compatible Operating Systems, insufficient hardware requirements, etc.SHARE tries to overcome these issues by supporting researchers in the creation of remote Virtual Machine Images (VMIs) where all software and data related to a publication can be deployed and configured. The resulting VMI is therefore an environment where reviewers can validate and evaluate the paper results without having to download or install anything on their local computers.

3.4 Conclusion

In modern science, scientific communities and funding agencies recognize the importance of sharing research results together with their experimental context. Given such objectives, the traditional publication paradigm reveals several limits and the notion of Enhanced Publication (EP) has been extensively investigated and developed to fill these gaps. As a result, the literature offers today a plethora of information systems specifically devised to manage EPs that address the scientific communication requirements of a given application domain. Examples of publication ”enhancements“ are: providing advanced reading facilities; packaging related research products; creating relationships between research products; and ultimately, supporting the re-production and assessment of scientific experiments.

In this chapter a common terminology and classification schemes have been introduced in order to shed some light and put some order in such a rich but foundation-less realm. More specifically, common structural features of data models have been identified and information systems have been classified in terms of their main functional goals. As preliminary output of this survey, two considerations can be made.

Existing information systems tend to be delivered adopting ad-hoc technical implementations. Their feature is that of thoroughly satisfying the needs of the final users for whom they were conceived, but in general they fail to be re-used in different contexts, where data model and functional requirements may slightly or heavily change. In fact, being in its early stage of factorization and foundation, this research field lacks general-purpose technical solutions, in the direction of Enhanced Publication Management Systems (EPMSs). Such systems would enable developers to easily realize and maintain Enhanced Publication Information System (EPIS) starting from their data models.

EPs are certainly worth consideration as proved by the strong interest and investment of scientific communities. Their wide adoption and usage is today hindered by their discipline specific character, which calls for radically different data models and information systems for different communities, and by the higher manual efforts required to draft EPs, which require authors to approach a learning curve without any certainty of getting scientific benefits. For such reasons, today EPs are more frequent in those contexts where their creation and access are imposed by given policies. Examples are scientific journals enforcing EPs formats, e-Science infrastructures enforcing procedures to publish publications together with reproducible experiments. Due to the strong motivations behind research data sharing and the valuable impact of research infrastructures, which provide communities with common services for experimental reproduction, it is hoped and expected these difficulties to dissolve in the future, both at discipline and cross-discipline levels.

4

Enhanced Publication Management Systems

Chapter 3 described a selection of Enhanced Publication Information System (EPIS) and highlighted their differences and commonalities in terms of data model features and functionalities. Those systems are usually based on a predefined data model, tailored to the target user community and its present requirements or use-cases. Though community-specific EPISs are effective for the community they target, they typically entail non-negligible realisation and maintenance costs. EPIS designers and developers have little or no technological support oriented to Enhanced Publications (EPs). In fact, they will realize EP-oriented software by integrating technologies that are general purpose (e. g. databases, file stores) and Digital Library (DL)-oriented (e. g. repository software, cataloguing systems). The integration of such different technologies is not easy from the point of views of design, realization and maintenance. Also, re-using software implementing common functionalities is often not possible due to the specificity of the EPIS under realization As a result, those common functionalities are re-implemented "from scratch" and another software is realised so that it is hardly re-usable and configurable for other communities. It should not surprise that the main agents in the realisation of EPISs are big publishers that have human and economic resources to afford such an investment (e. g. Public Library of Science (PLOS), Elsevier, and Nature). Research institutions, offering traditional Digital Library Systems (DLSs) and repositories to researchers, cannot generally afford the realisation and maintenance costs of a new information system and thus they are currently excluded from the shift from traditional digital publications to EPs.

That "vicious circle" for the realisation of information systems recalls the situation of software development before the advent of Data Base Management Systems (DBMSs), when each application had to implement its functionality for data access, manipulation and retrieval. The introduction of software capable of hiding the complexity of data storage and management (i. e. DBMSs) was crucial to the technology revolution and to the development of sustainable commercial software products [73][66]. Some of the features that made DBMSs so popular are:

- *General-purposeness*: DBMSs can be used in any application domain, since the tools they offer address generic tasks that are shared across domains;

- *Technology transparency*: DBMSs hide the complexity needed to store and retrieve data on different back-ends;
- *Standard Application Programming Interfaces (APIs)*: DBMSs offer user-oriented languages for data definition, manipulation, and access;
- *Data sharing*: DBMSs enable data sharing among different applications and users;
- *Portability*: DBMSs support data and application portability.

This way of thinking has not yet been applied to the scene of EPIS, where the realization "from scratch" is the norm and no attempts are known for the realization of Enhanced Publication Management Systems (EPMSs), intended as software frameworks that play the role of DBMSs in the world of EPs.

The term Enhanced Publication Management System (EPMS) denotes an information system that supports EPIS designers and developers with EP-oriented tools for the set-up, operation and maintenance of EPISs. Specifically, EPIS designers and developers are supported with an EP-oriented software suite that offers, like DBMSs do in the database scenario, the benefits of a systemic approach:

- *General-purposeness*: offering tools that address generic tasks that are shared across domains and user communities;
- *Technology transparency*: hiding the complexity of the implementation of domain-independent functionalities, so that EPIS developers can focus on the specificities of the target community, rather than re-implementing generic functionalities such as EP storage and retrieval functions;
- *Availability of APIs for EP sharing*: offering APIs for sharing EPs and their parts via standard export protocols to support data and metadata interoperability;
- *Availability of user-oriented languages*: offering user-oriented languages for EP definition, manipulation, and access;
- *Extensibility*: enabling the integration of new back-ends and technologies in order to be up-to-date with regards to Information and Communication Technology (ICT) advancements and to allow EPIS developers to select the best technology for a given functionality from a selection of supported technologies.

When supported by an EPMS, EPIS developers can focus on the definition of the EP data model and the configuration of the functionalities in order to satisfy the requirements of the target community.

In the following sections a reference software architecture for EPMSs is presented that sets up the foundations for the design of concrete implementations of EPMSs satisfying the requirements above.

Outline Section 4.1 presents a reference data model for EPMSs inspired by the data model features described in 3.1. The model is called EP "meta-model" because it is a data model that allows the representation of customised EP data models. The EP data model defines a set of "meta-types" (i. e. types of types, "kinds" in type theory jargon) that describe structural and semantic templates of objects together with the functionalities that can be operated over such objects and their metadata. Section 4.2 addresses in details

the functionalities to be offered on each meta-type. In Section 4.3 the requirements of EPMSs listed above are described in details. Finally, Section 4.4 describes the reference software architecture for EPMSs.

4.1 The EP meta-model

The analysis of the state of the art in Section 3.1 resulted in the identification of the typologies of parts that can form EPs:

- Embedded parts: identity-less parts that are packaged within the enhanced publication. Embedded parts are not directly discoverable, i. e. it is possible to access them only once the embedding resource has been discovered. Typical examples of embedded parts are the supplementary material of scientific articles managed by journals: users cannot search for supplementary material, but once they have found the relative article (i. e. its embedding part), then they can access it.
- Structured-text parts: structured textual documents, with well defined editorial components, such as publications in Journal Article Tag Suite (JATS) XMLs format.
- Reference parts: remote resources referenced via resolvable identifiers such as Uniform Resource Identifiers (URIs) or Persistent Identifier (PID) (e. g. handle, Digital Object Identifier (DOI)).
- Executable parts: parts that can be executed, such as workflows, web service instances, software code.
- Generated parts: parts that are generated from other parts. Examples are dynamic tables that are updated when the underlying data set changes; or a molecule 3D rendering generated by running a 3D molecule viewer with appropriate parameters.

Based on the above typologies of parts, a reference data model for EPMSs, called *EP meta-model*, has been defined. The EP meta-model is a data model for the representation of EP data models. Its role is similar to that of the relational model for Relational Data Base Management Systems (RDBMSs): providing the primitives for the definition of types that denote the universe of data (see Figure 4.1). Developers willing to realize a relational database with the support of a RDBMS must define their domain data model in terms of the primitives offered by the relational model: relations (or tables), attributes (or columns) and keys [51, 48]. Likely, developers willing to realize an EPIS with the support of an EPMS must define their EP data model in terms of the primitives offered by the EP meta-model: the meta-types.

Meta-types can be defined as “types of types” and correspond to “kinds” in type system jargon [44, 130]. A meta-type describes one structural and semantic template of objects together with the functionalities that can be operated over such objects. The instantiation of a meta-type yields the creation of a type. The instantiation of a type yields the creation of an object. The following notation will be used in the remainder of this chapter:

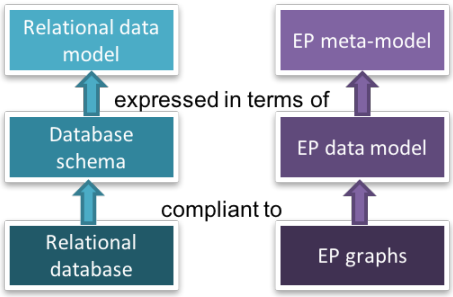


Fig. 4.1. The role of the EP meta-model in an EPMS is equivalent to that of the relational model in a RDBMS

- $T :: MT$ – T is a type, instance of the meta-type MT;
- $obj : T :: MT$ – obj is an object, instance of type T, with meta-type MT;
- $obj : T$ – obj is an object, instance of type T. This notation is used whenever it is not necessary to specify the meta-type of T;
- $obj :: MT$ – obj is an object with meta-type MT (note the double colon). This notation is used whenever it is not necessary to refer to the specific type of obj.

Figure 4.2 shows a visual representation of the EP meta-model and the meta-types it defines. An instance of *EPType* $T_{EP} :: EPType$ denotes a type of EP. Objects with this meta-type are EPs ($ep :: EPType$) that must be linked to at least one metadata record with meta-type *MetadataFormat* ($r :: MetadataFormat$) (see link labelled "described by" in Figure 4.2) and a textual narration part with meta-type *NarrationType* ($n :: NarrationType$). Based on the parameters specified at type creation time, EPs can aggregate additional parts ($p :: PartType$).

An instance of *MetadataFormat* $T_{metadata} :: MetadataFormat$ denotes a structured format of metadata that defines a tree of properties (or fields), where each property has a name and a type, which can be a structured type defining nested properties.

An instance of *NarrationType* $T_{narr} :: NarrationType$ represents the type of digital publications to enhance. The meta-type *NarrationType* can be seen as a "super-type" of the meta-types *RefType*, *FileType* and *StructuredTextType* and it is used in the meta-model to specify that a digital publication to enhance pub can be a remote resource ($pub :: RefType$), a file stored locally in the system ($pub :: FileType$), or a structured textual document ($pub :: StructuredTextType$).

The meta-types *RefType*, *FileType* and *StructuredTextType*, together with *GeneratedType*, *ExecutableType*, are also sub-types of the meta-type *PartType*. Instances of sub-types of *PartType* can be defined as "embedded" at type creation time, via the dedicated property defined by *PartType*.

The model allows also to define types of relationships via the meta-type *RelType*. An instance of *RelType* $T_{rel} :: RelType$ defines a type of relationships in terms of multiplicity and allowed labels, which express the nature of the association. This capability is visually

presented in Figure 4.2 via the dotted lines that link relationships between *RelType* and the other meta-types.

In addition to the meta-types that reflect the data model features described in Section 3.1, the model also offers a generic meta-type called *ObjType* that, like *EPType* and *PartType*, defines a mandatory relationship to *MetadataFormat*. The introduction of *ObjType* is needed to enable the representation of real world entities such as people, organisations and funding programmes. Those entities cannot be represented as "parts" of EPs because the semantics of "parts" as aggregated research products would be violated. Real world entities could have been modelled just using the meta-type *MetadataFormat*, but that would have prevented the possibility of objects to be associated to metadata records of different formats. In addition, it would have introduced a semantic inconsistency for objects with meta-type *MetadataFormat* $obj :: MetadataFormat$: some of them would have to be considered descriptions of an object (when associated to an EP $ep :: EPType$ or to a part $p :: PartType$), while others would have to be considered the actual digital objects representing the real world entity.

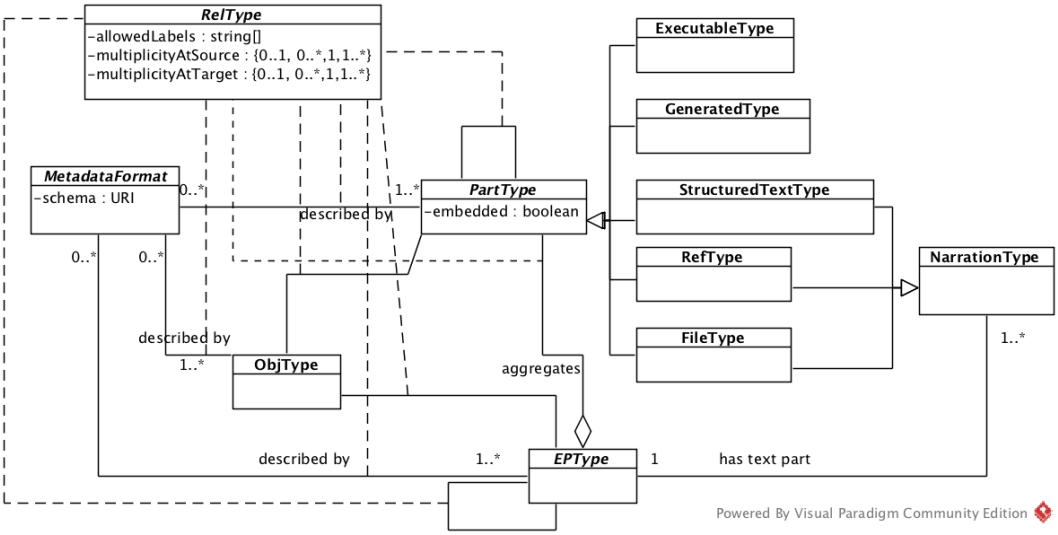


Fig. 4.2. The EP meta-model

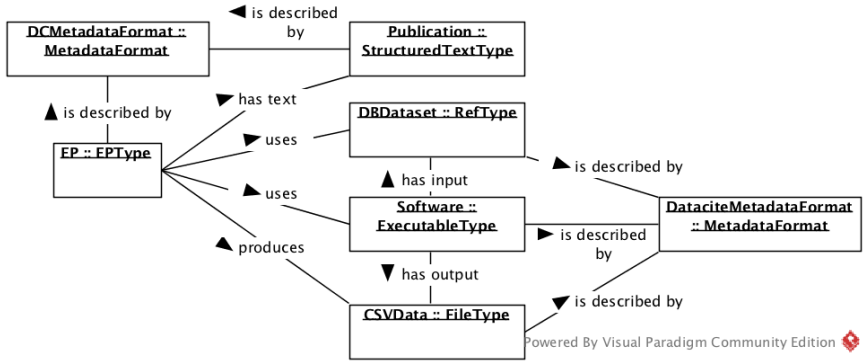
To further clarify, Example 3 shows how the simple EP data model of Example 2 can be represented in terms of the EP meta-model.

4.2 Meta-types and functionalities

Meta-types correspond to "kinds" in type system jargon [44, 130]: they describe structural and semantic templates of objects together with the functionalities that can be operated over such objects and their metadata. In the following, functionalities on meta-types and

Example 3 Representing the EP data model of Example 2 in terms of the EP meta-model

The figure shows how the types of the EP data model of Example 2 can be expressed in terms of the EP meta-model.
An EP $ep : EP :: EPType$ *is described by* a metadata record r in Dublin Core format ($r : DCMetadataFormat :: MetadataFormat$). The narration part of ep is a publication $p : Publication :: StructuredTextType$, also *described by* a Dublin Core metadata record. Instances of EP can aggregate parts (research products) of three types: datasets hosted by remote databases($DBDataset :: RefType$), software code that can be executed ($Software :: ExecutableType$), and CSV files ($CSVData :: FileType$).
Parts are *described by* metadata records in Datacite metadata format ($DataciteMetadataFormat :: MetadataFormat$) and are aggregated by an EP via relationships with specific semantics (*uses*, *produces*). The type $Software$ also defines relationships with the other two part types, in order to specify the type of the expected input (*has input*) and output (*has output*).



their metadata are presented. A summary is available in Table 4.1. Considering the strict connection between an object and its metadata, the meta-type *MetadataFormat* will not be considered on its own, but rather in the context of the other meta-types that are associated to it. In fact, some of the functionalities can be operated only on the metadata of objects of a specific meta-type.

4.2.1 Functionalities on *EPType*

Creation

The creation of an EP requires at least: metadata about the EP, a textual narration part and its metadata. EP metadata must be provided by the user who wants to create the EP. The textual narration part and its metadata can be manually created or uploaded by the user or automatically imported from an external data source. Depending on the EP data model defined by the EPIS developers, the user must be asked to provide further research products for the creation of additional (mandatory or optional) parts of the EP.

	EPType	**ObjType**	**FileType**	**RefType**
Creation	Metadata: user input. Mandatory narration part.	Metadata: user input, user upload or import from data source.	Metadata: user input, user upload or import from data source. Payload: user upload or import from data source.	Metadata: user input, user upload or import from data source.
Search/Browse	On metadata, metadata of parts, payload of parts (when applicable).	On metadata.	On metadata and payload.	On metadata.
Export	Metadata via OAI-PMH. EP graph exported via OAI-ORE and Linked Data, downloadable as "self-contained package".	Metadata via OAI-PMH, OAI-ORE, Linked Data.	Metadata via OAI-PMH, OAI-ORE, Linked Data. Payload via standard file exchange protocols (e. g. HTTP, FTP).	Metadata via OAI-PMH, OAI-ORE, Linked Data.
Visualization	Ready-to-use GUI exploits structural info of the EP data model.	Visualization of metadata.	Visualizer configurable based on the mime-type.	Visualization of metadata and properties specified by the type.
Execution	n.a.	n.a.	n.a.	n.a.

	StructuredTextType	**GeneratedType**	**ExecutableType**
Creation	User input, user upload or import from data source. Possible automatic generation of parts based on the document components.	Metadata: user input*.	Metadata: user input, user upload or import from data source *.
Search/Browse	On metadata, text and tagged elements.	On metadata.	
Export	Metadata via OAI-PMH, OAI-ORE, Linked Data. Payload via HTTP, FTP.	Metadata via OAI-PMH, OAI-ORE, Linked Data.	
Visualization	Advanced visualisers exploiting the structure of the text.	Visualization of metadata and properties specified by the type. Materialisation of the content on request.	Visualization of metadata and properties specified by the type. Ability of the user to start the execution.
Execution	n.a.	n.a.	Integrated/external execution.

*Additional info for the generation/execution of the part depend on the specific type and must be provided by the user.

Table 4.1. Summary of the functionalities available for each meta-type of the EP meta-model

Search and browse

EPs should be searchable and browsable (via faceted search) based on their metadata, the metadata of their parts, and the payload of their parts (when applicable, for example for parts with meta-type *FileType*). Regarding metadata records of EPs and parts, they contain descriptive information that is very useful for discovery purposes. Their structured and semantic nature enables the creation of one index per metadata field in order to support specific queries. The EPMS should support the creation of such indices and allow EPIS developers to specify the mapping between paths of the tree structure of the metadata format and index field names, which can be used to formulate queries.

Export

Exporting metadata records according to standard exchange protocols and formats is fundamental to enable machine consumption of EPs and one of the basic requirements to achieve data interoperability among different information systems. In the scientific communication domain some protocols and metadata formats are widely adopted and should be therefore natively supported by the EPMS. For example, the OAI-PMH protocol and the metadata formats Dublin Core (DC) and Datacite are particularly relevant for institutional and thematic repositories.

In order to further promote interoperability and re-use of data and metadata, the scientific communication domain is also embracing the Linked Data principles. Exports in RDF, OAI-ORE and the availability of a SPARQL endpoint for the navigation of the EP graph should be supported natively.

Finally, an EP should also be made exportable as a "self-contained package" that includes all metadata and payloads of its parts together with information about its structure. Structural information may be encoded as a Metadata Encoding and Transmission Standard (METS)[1] document that specifies how the parts are related to each other.

The EPMS should provide EPIS developers with tools for the configuration of the export modes described above. Although the described protocols cover the majority of needs of scientific communities in terms of export, in some cases EPIS developers might need to introduce new export modes that are not natively offered by the EPMS. To support developers in this task, the EPMS should offer a mechanism for the integration of new export components.

Visualization

The EPMS should offer ready-to-use Graphical User Interface (GUI) for creating, searching, visualizing and navigating EPs compliant to the EP data model defined by the EPIS developers. Such a customised GUI can be made available out-of-the-box by exploiting

[1] Metadata Encoding and Transmission Standard (METS) is a standard maintained in the Network Development and MARC Standards Office of the Library of Congress for encoding descriptive, administrative, and structural metadata regarding objects within a digital library `https://www.loc.gov/standards/mets/mets-home.html`. Last accessed: February 2016.

the structure of types and the configuration of functionalities defined in the EP data model. For example, metadata records are structured data whose form is defined by a metadata format. By exploiting the information provided by a metadata format, the EPMS can automatically generate GUI components for the manual creation and visualization of metadata records. An advanced configuration of the GUI can be made available in order to allow EPIS developers to select what to visualise and what to hide to end-users. For example, some metadata formats define fields with information useful to humans together with fields that ease machine interpretation but that do not convey any additional information. In this case, the EPIS developers may decide to hide to the end-user the fields that are not "human-oriented" (e. g. hide codes and show human-readable labels).

4.2.2 Functionalities on *ObjType*

ObjType is a meta-type introduced to model entities of the real world as objects that carry only metadata information. Examples are people, organisations and funding programmes. Those entities are relevant for the scientific communication domain but they cannot be considered "parts" of an EP, but rather contextual information. According to that view, functionalities on objects with meta-type *ObjType* are operated on the associated metadata records, which are objects with meta-type *MetadataFormat*.

Creation

The creation of an object with meta-type *ObjType* requires the creation of a metadata record. Users can create a metadata record by:

1. manually providing the values for the metadata fields via a GUI;
2. uploading a file containing a serialisation of the metadata record (e. g. XML);
3. selecting a metadata record from an external data source.

Search and browse

Objects with meta-type *ObjType* are searchable and browsable based on the content of their metadata records. The structured and semantic nature of metadata records enables the creation of one index per metadata field in order to support specific queries. The EPMS should support the creation of such indices and allow EPIS developers to specify the mapping between paths of the tree structure of the metadata format and index field names, which can be used to formulate queries.

Export

Metadata records of objects with meta-type *ObjType* are exportable via the standard protocols for metadata exchange supported by the EPMS and can be included also in the Linked Data exports.

Visualization

Objects with meta-type *ObjType* can be visualised only via the associated metadata records. By exploiting the information provided by the metadata format defined in the EP data model, the EPMS can automatically generate GUI components for the manual creation and visualization of those metadata records.

4.2.3 Functionalities on *PartType*

Before addressing functionalities on the different typologies of parts, this subsection provides an overview and collect their common functionalities.

Creation

Creating a part means creating its metadata, its payload (if applicable), and all properties defined by its type. All this information can be provided:

1. manually by the end-user via a GUI;
2. uploaded by the end-user;
3. automatically, importing the needed information from an external data source.

The three approaches are not exclusive: in some cases the automatic import or the upload of a research product cannot produce all the information needed by the EPMS to generate the part, so users can be asked to manually provide the missing information.

Search and browse

All non-embedded parts are searchable and browsable via their metadata records. Parts that include a payload such as those with meta-type *FileType* and *StructuredTextType* can also be searchable via the content of their payloads.

Export

All metadata records of non-embedded parts are exportable. Parts that include a payload such as those with meta-type *FileType* or *StructuredTextType* can also export their payloads via standard file exchange protocols (e. g. Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP)).

Visualization

All metadata records of parts can be visualised in dedicated forms whose structure depend on their actual type (instance of the meta-type *MetadataFormat*). Based on the specific meta-type of a part, special visualisation option may be available. For example, the GUI could enable the visualization of a video for objects with meta-type *FileType* and proper mime-type (e. g. video/mp4).

4.2.4 Functionalities on *FileType*

Creation

The creation of a part with meta-type *FileType* requires the creation of its metadata record (in the format defined in the EP data model) and the provision of a payload whose mime type depends on the specific type of the part under creation. The metadata record can be created as described in Section 4.2.3, while the payload can be uploaded by end-users, fetched from an existing data source, or automatically downloaded from a given Uniform Resource Locator (URL). The URL can be provided manually by the user or extracted from the metadata record.

Search and browse

When defining an EP data model, the meta-type *FileType* must be specialised to include parts whose files have a specific mime type. This feature allows the EPMS to provide developers with tools for the customisation of full-text indices in order to realise search and browse functionality on payloads. The EPMS should provide a (possibly extendible) set of "feature extractors" that, given a mime-type, are able to extract indexable information.

Export

Metadata records of objects with meta-type *FileType* are exportable via the standard protocols for metadata exchange supported by the EPMS and can be included also in the Linked Data exports. In addition, payloads can be exported via standard file exchange protocols such as FTP and HTTP.

Visualization

Like for the search and browse functionality, the mime type of the payload is very important to properly visualize parts of a given *FileType*. The EPIS developers may want to configure video payloads to be opened in a video player, image files to be visualised as thumbnails, etc..

4.2.5 Functionalities on *RefType*

Creation

A part of type *RefType* represents a remote resource that is reachable from a given URL. Examples of such parts range from remote files, databases, entries from an ontology. When defining a type with meta-type *RefType*, EPIS developers must specify which are the required properties that are needed to ensure the correct access to the referenced resource. Like objects with other meta-types, *RefType* objects must be associated to a metadata record. The metadata record can be created as described in Section 4.2.3, while the required properties defined by the type must be provided by the end-user via manual insertion or via extraction from the metadata record, if possible.

Search and browse

Objects with meta-type *RefType* are searchable and browsable based on the content of their metadata records. The structured and semantic nature of metadata records enables the creation of one index per metadata field in order to support specific queries. The EPMS should support the creation of such indices and allow EPIS developers to specify the mapping between paths of the tree structure of the metadata format and index field names, which can be used to formulate queries.

Export

Metadata records of parts with meta-type *RefType* are exportable via the standard protocols for metadata exchange supported by the EPMS and can be included also in the Linked Data exports.

Visualization

For the visualization of parts with meta-type *RefType* the GUI should take into consideration both the associated metadata records and the properties defined by the specific types in the EP data model.

4.2.6 Functionalities on *StructuredTextType*

Creation

A structured text is a textual document with explicit "tags" that identify its document components such as title, chapters, figures, tables, and external references. Several formats for structured documents are adopted in scientific communication. Some of them are XML-based, such as the ANSI/NISO standard JATS [5]; others are HTML-based, such as Scholarly Markdown [103] and Research Articles in Simplified HTML (RASH) [128]. If the format of the structured text is known to the EPMS, then it is possible to provide advanced visualisation features that exploit the editorial component tags and enable end-users to navigate through the text and access it in a non-linear way. The exploitation of the document component tags could be performed also to automatically create additional parts. For example, the existence of an external reference tag may lead to the creation of a part with meta-type *RefPart*. The mapping from document component tags and the types of the EP data model cannot be known a priori by the EPMS and must be defined by the EPIS developers.

Search and browse

The search and browse functionality on objects with meta-type *StructuredTextType* can be enabled both on the associated metadata records and on the full-text. The structured nature of the full-text enables the possibility of specifying different index fields for each document component tag. This allows end-users to formulate queries on specific subparts of text.

Export

Metadata records of parts with meta-type *StructuredTextType* are exportable via the standard protocols for metadata exchange supported by the EPMS and can be included also in the Linked Data exports. In addition, the payloads can be exported via standard file exchange protocols such as FTP and HTTP.

Visualization

The GUIs can take advantages of the structured format of the text to offer advanced visualization features and more interactive functionality for the end-users, as described in Section 3.3.1:

- Structuring the narrative text into interconnected sub-parts to ease navigation through sections and editorial parts of the article;
- Re-using the universe of web resources to enrich the text (e. g. show definition of terms from DBpedia, Wikipedia or domain-specific ontologies, taxonomies, and vocabularies);
- Generating the graph of citations.

4.2.7 Functionalities on *GeneratedType*

Creation

Generated parts are dynamically generated by combining other parts of the EP. The generation of the part is not performed upon its creation, but rather when the part must be exported or visualised, because its content is dynamic and may vary over time. Examples are dynamic tables that are updated when the underlying data set changes, a molecule 3D rendering generated by running a 3D molecule viewer. The creation of the metadata record to associate to a generated part must be provided by the user. Additional information for the generation of the part depends on the specific type and must be provided by the user as well.

Search and browse

Objects with meta-type *GeneratedType* are searchable and browsable based on the content of their metadata records. The structured and semantic nature of metadata records enables the creation of one index per metadata field in order to support specific queries. The EPMS should support the creation of such indices and allow EPIS developers to specify the mapping between paths of the tree structure of the metadata format and index field names, which can be used to formulate queries.

Export

Metadata records of parts with meta-type *GeneratedType* are exportable via the standard protocols for metadata exchange supported by the EPMS and can be included also in the Linked Data exports.

Visualization

Generated parts are parts whose content is produced on request by applying a function to one or more parts of the same container EP. Software components are needed for the generation of a part and should be defined by EPIS developers when they create a type with meta-type *GeneratedType*.

4.2.8 Functionalities on *ExecutableType*

Creation

Executable parts are parts that can be executed, such as workflows, web service instances, software code. Metadata records of an executable part can be created manually by end-users, uploaded from an existing file or imported from an external data source. When defining a type with meta-type *ExecutableType*, EPIS developers must specify which are the required properties that are needed to ensure the executability of the part. Values of this property can be provided manually by the user that is creating the part or extracted from the metadata, if possible.

Search and browse

Objects with meta-type *ExecutableType* are searchable and browsable based on the content of their metadata records. The structured and semantic nature of metadata records enables the creation of one index per metadata field in order to support specific queries. The EPMS should support the creation of such indices and allow EPIS developers to specify the mapping between paths of the tree structure of the metadata format and index field names, which can be used to formulate queries.

Export

Metadata records of objects with meta-type *ExecutableType* are exportable via the standard protocols for metadata exchange supported by the EPMS and can be included also in the Linked Data exports.

Visualization

Together with the visualization of the associated metadata records, the GUI should enable end-users to execute the part (e. g. by pushing a button).

Execution

Depending on the specific execution parameters, the part can be executed:

- locally in the EPIS, whenever the execution context and engine is integrated by the underlying EPMS;
- by a third-party, remote (web) application, e. g. a Taverna workbench,[2] where the user is redirected when requesting the execution.

[2] Taverna: powerful, scalable, open source and domain independent tools for designing and executing workflows. `http://www.taverna.org.uk`. Last accessed: February 2016.

4.3 Requirements of EPMSs

The main goal of an EPMS is that of helping developers at the realisation, maintenance and operation of an EPIS configured for a target community. To draw a parallel with the database world, an EPMS is for an EPIS what a DBMS is for a software application using a database. As such, an EPMS should feature:

1. *General-purposeness*: offering tools that address generic tasks that are shared across domains and user communities;
2. *Technology transparency*: hiding the complexity of the implementation of domain-independent functionalities, so that EPIS developers can focus on the specificities of the target community, rather than re-implementing generic functionalities such as EP storage and retrieval functions;
3. *Availability of APIs for EP sharing*: offering APIs for sharing EPs and their parts via standard export protocols to support data and metadata interoperability;
4. *Availability of user-oriented languages*: offering user-oriented languages for EP definition, manipulation, and access;
5. *Extensibility*: enabling the integration of new back-ends and technologies in order to be up-to-date with regards to ICT advancements and to allow EPIS developers to select the best technology for a given functionality from a selection of supported technologies.

Requirement 1 (General-purposeness)**.** The EPMS should address the complexity of common operations and functionalities of EPISs on behalf of the developers:

- Management of EP graphs: Create, Read, Update, Delete (CRUD) operations on parts, metadata and relationships;
- Navigation on EP graphs;
- Discovery of EPs based on their metadata, the metadata and payloads of their parts;
- Data and metadata export via standard exchange protocols and formats to support interoperability;
- Providing ready-to-use GUIs for the visualization of EP graphs;

Requirement 2 (Technology transparency)**.** Different functionalities may require the adoption of different technologies that better serve at their implementation. For example, a metadata record can be stored on a relational database but, for the realization of an efficient search functionality, it can also be indexed by a full-text engine. In fact, a type of back-end may be preferable to another to accomplish a given functionality at a desired level of quality of service. There are technologies such as graph databases and triplestores,[3]that natively support the navigation of graphs and the export of data ac-

[3] An introduction to the concept of triplestores (or RDF stores) is available on Wikipedia at `https://en.wikipedia.org/wiki/Triplestore` where triplestores are defined as *A triplestore or RDF store is a purpose-built database for the storage and retrieval of triples through semantic queries. A triple is a data entity composed of subject-predicate-object, like "Bob is 35" or "Bob knows Fred".*

cording to Linked Data principles [76]; full-text engines like Apache Solr[4] and Elasticsearch[5] are instead preferable to support search and browse functionalities. By default, the choice of a back-end technology for the implementation of a functionality should be transparent to EPIS developers, who can therefore focus on the customisation of the functionality rather than in the implementation details for the correct interaction with a specific technology. Nonetheless, advanced developers may find this approach limiting the capabilities of the system and may prefer to loose the benefits of technology transparency in order to gain more power on the configuration of the back-end technology. For this reason it is preferable that an EPMS include a mechanism through which developers can "put their hands" on low-level configuration details.

Requirement 3 (Availability of APIs for EP sharing)**.** In the scientific communication domain there are several standard protocols for sharing research products. Most of these protocols focus on sharing the metadata of research products, which bear descriptive information along with the terms of use of the described product (e. g. license, access rights), provenance details (e. g. authors, responsible organisations, devices) and the coordinates for retrieving the actual content of the research product (e. g. PIDs, URIs, domain specific identifiers such as PubMed for life science articles or Protein Data Bank identifiers for proteins). The most relevant standard protocols for data sharing in the scientific communication domain are:

- Open Archive Initiative - Protocol for Metadata Harvesting (OAI-PMH)
- Open Archive Initiative - Object Reuse and Exchange (OAI-ORE)
- Linked Data (Resource Description Framework (RDF) and SPARQL Protocol and RDF Query Language (SPARQL))

Being the most relevant protocols for the scientific communication domain, an EPMS should support them natively and should provide EPIS developers with tools for their configuration (e. g. OAI-PMH sets and metadata formats).

Requirement 4 (Availability of user-oriented languages for EP definition, manipulation and access)**.** RDBMSs offer the Structured Query Language (SQL) for the definition of tables, columns, keys, etc., the manipulation of and access to tuples. Likely, the EPMS should offer a language for the definition of types of EPs, i. e. EP data models, and the manipulation and access of EP instances compliant to the defined EP data model.

Requirement 5 (Extensibility)**.** The EPMS is designed to be general-purpose, so it supports common functionalities that are shared across different domains and user-communities. Some application domains, however, may need specific functions that are not natively supported by the EPMS. To tackle such cases, the EPMS should be extensible and offer a mechanism that allows EPIS developers to introduce new functionalities and new software components that implement them. In addition, the EPMS should be easily

[4] Apache Solr: `http://lucene.apache.org/solr`
[5] Elasticsearch: `https://www.elastic.co/products/elasticsearch`

extendible to support new back-end technologies in order to be up-to-date with regards to ICT advancements and to not lock EPIS developers up to legacy code and technologies.

4.4 Reference architecture

This section presents a software reference architecture for EPMSs that sets up the foundations for the design of concrete implementations of EPMSs satisfying the requirements presented in Section 4.3:

1. General-purposeness;
2. Technology transparency;
3. Availability of APIs for EP sharing
4. Availability of user-oriented languages for EP definition, manipulation, and access
5. Extensibility

The reference architecture here proposed is based on the architectural pattern of micro-services. The pattern of micro-services is an approach to Service-oriented architecture (SOA) where an application is built as a suite of small services, each implementing a specific functionality – respecting the design principles of separation of concerns and single responsibility [110, 109] –, running in its own process and communicating with lightweight mechanisms [102, 116, 119, 153].

With an architecture based on micro-services, the EPMS can benefit from the following characteristics, discussed in details by Richardson [140] and Newman [119]:

- Technology heterogeneity and polyglot persistence:[6] micro-services can adopt different technologies, choosing the one that best serve the functionality to be implemented;
- Resilience:[7] a failure of a micro-service does not lead the whole application to break and stop working. In fact, the system can be build to smoothly handle the failure and degrade functionality until the micro-service is recovered;
- Scaling: each micro-service can be scaled independently when needed;
- Ease of deployment: a micro-service can be deployed independently. Consequently, it is easier to update an existing micro-service (as long as there are no changes to its interfaces) and to introduce a new one that implements a new functionality of the system;
- Freedom of implementation choice: it is easier to distribute development tasks across development teams with different technical skills (also thanks to the first benefit of "Technology heterogeneity and polyglot persistence"). Developers must not stick to a given (legacy) technology stack, but can choose the programming language and framework they prefer to implement the micro-service;

[6] Polyglot Persistence: `http://martinfowler.com/bliki/PolyglotPersistence.html`, Martin Fowler, 16th Novemebr 2011. Last accessed: February 2016.

[7] In [84] Jackson defines resilience as the ability to prevent something bad from happening, becoming even worse, and to recover.

- Composability: micro-services can communicate with each other and compose existing functionalities to implement new ones;
- Optimizing for replaceability: being small, micro-services are easier to re-implement and to replace in order to offer better performances or to test new back-end technologies.

Those characteristics of micro-services are useful in supporting the requirements of EPMSs, however micro-services are not a *panacea*, as they come with additional complexities in terms of process orchestration, choreography and operational management. Wootton [158] and Richardson [140] summarized the drawbacks of micro-services as follows:

- Distributed system complexity and operations overhead: micro-service architectures are SOA and, as such, they inherit all the complexities of distributed systems. Distributed systems imply a number of concerns that are not relevant in monolithic architectures such as service discovery, network latency, message serialization, network failures, asynchronous communication.
- Substantial DevOps skills required: the micro-service approach pushes developers to follow the full software life cycle from coding to production deployment and to have a good understanding of a plethora of technologies (see "Technology heterogeneity and polyglot persistence" above);
- Implicit interfaces: more traditional SOA based on Web Services and Simple Object Access Protocol (SOAP) can rely on the fact that services can collaborate based on specific contracts defined by published interfaces. With micro-services, message formats and protocols are implicitly defined by unpublished interfaces. Small changes in the interface of a micro-service can lead to a chain of failures or errors in other micro-services that are not easy to detect in the development and testing phase;
- The CAP theorem [60, 36]: finding the right trade-off among consistency, high availability and tolerance to network partitions based on the specific system to be realised.

Figure 4.3 shows an overview of the reference architecture for EPMSs based on micro-services. For each meta-type, the EPMS specifies which functionality is supported and how it can be configured. It also specifies the interfaces that must be implemented by micro-services willing to support the functionality on a meta-type (partially covering the "Implicit interfaces" drawback). A micro-service that implements a functionality for a meta-type registers to the *EPMS registries* in order to be discoverable.

If a functionality is available for a meta-type (i. e. a micro-service is registered for that functionality on the given meta-type), then the tools for functionality configuration allow EPIS developers to customise the functionality on types derived from that meta-type. Section 4.4.2 describes in details the procedure for the definition of EP data models and the configuration of functionalities. Several micro-services can offer the same functionality for the same meta-type, but adopting different back-ends, like μs_2 and μs_3 in Figure 4.3. Clients are not aware of this difference, as they call the services via the same API, which does not depend on the specific back-end technology. Figure 4.3 also shows that

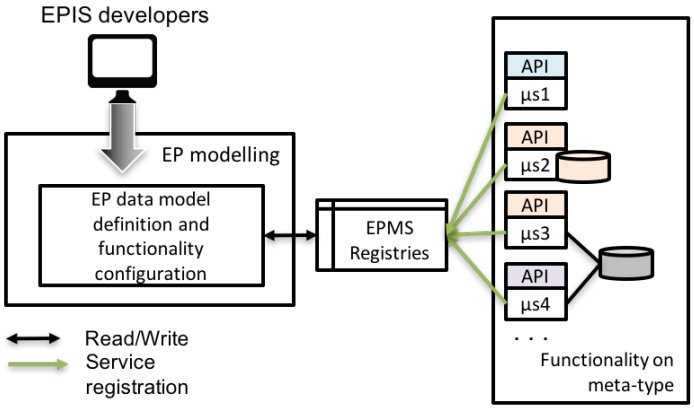


Fig. 4.3. EPMS architecture overview

micro-services implementing different functionalities can be based on the same back-end technology (μs_3 and μs_4). Such a pattern goes against the "pure" micro-services approach, which follows instead the "database per service" pattern to ensure that micro-services are loosely coupled.[8] In the context of EPMS, the re-use of data from the same back-end instance and the introduction of implicit dependencies between micro-services should be considered the result of a trade-off among service independence, simplicity of operation and use of hardware resources (e. g. virtual servers and disk space).

Figure 4.4 goes into the details of the reference architecture. For the sake of image clarity, the service registration links (the green links in Figure 4.3) have been omitted. The architecture is presented as composed of seven logical areas:

1. *Modelling and configuration*: includes tools and services for the definition of EP data models, the configuration of the functionalities and the management (registration, discovery and coordination) of services (Sections 4.4.1 and 4.4.2)
2. *Data source mediation*: includes tools and services for the import of research products from external data sources (Section 4.4.3);
3. *EP graph management*: includes tools and services for the management of EP graphs (Section 4.4.4);
4. *Search and browse*: includes tools and services for the creation, update and query of indices for searching and browsing for EPs and their parts (Section 4.4.5);
5. *Visualization*: includes tools and services for the provision of ready-to-use GUI for end-users (Section 4.4.6);
6. *Export*: includes tools and services for the export of EPs (Section 4.4.7);
7. *Execution*: includes tools and services for the execution of executable parts (Section 4.4.8);

[8] Database per service pattern: `http://microservices.io/patterns/data/database-per-service.html`. Last accessed: February 2016.

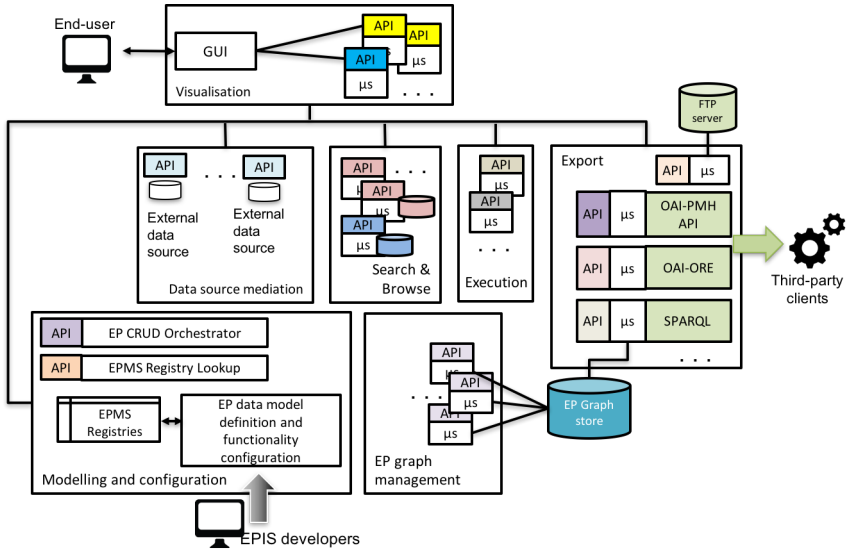


Fig. 4.4. Software reference architecture for Enhanced Publication Management Systems (EPMSs) based on micro-services

4.4.1 EPMS registries

The EPMS registries implement the mechanism for the registration and discovery of functionalities, types and micro-services. From a modelling point of view it is possible to identify a set of registries, each of them useful to support the creation of EP data models and the configuration of functionalities (Figure 4.5):

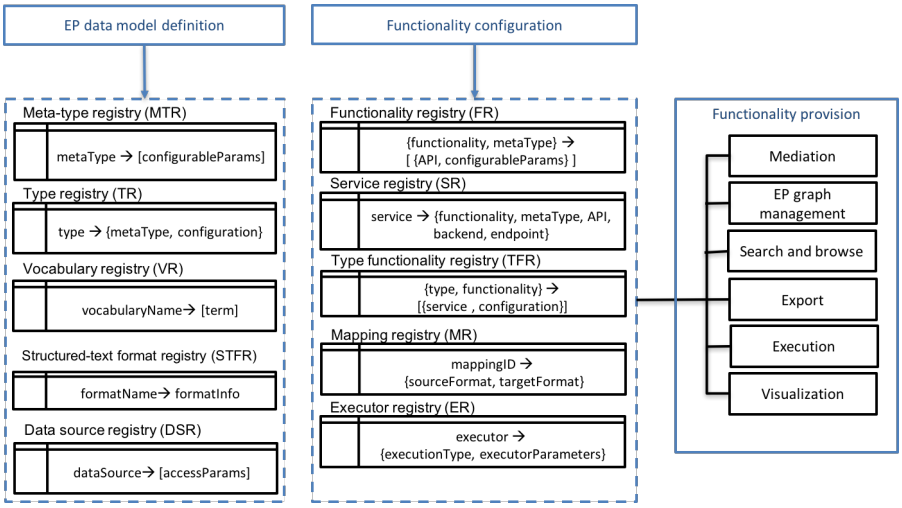


Fig. 4.5. EPMS registries

- Meta-type registry:
  $MTR : metaType \rightarrow [configurableParams]$
  It associates each meta-type to its configurable parameters. The latter are the parameters that the EPIS developers must configure when defining types of the EP data model. The meta-type registry is a read-only system registry that cannot be changed. Its content is derived from the specification of meta-types ad functionalities presented in Section 4.2.
- Type registry:
  $TR : type \rightarrow \{metaType, configuration\}$
  It associates types of the EP data model to their meta-types and the configuration of their configurable parameters. The type registry is fed by EPIS developers when defining types of the EP data model. For convenience, the EPMS can provide EPIS developers with a set of default types ready to be used and configured, such as types of metadata for standard formats.
- Vocabulary registry:
  $VR : vocabularyName \rightarrow [term]$
  It keeps track of the terms defined in a controlled vocabulary. The VR registry is useful for the selection of labels allowed by types of relationships (see Section 4.4.2).
- Structured-text format registry:
  $STFR : formatName \rightarrow formatInfo$
  It keeps track of the formats supported by the EPMS for structured-text parts. Examples of formats that could be natively supported by an EPMS are JATS, RASH, and Scholarly Markdown.
- Data source registry:
  $DSR : dataSource \rightarrow [accessParams]$
  It contains descriptive information about data sources (e. g. name, responsible organization) along with the parameters to access their endpoint for collecting research products.
- Functionality registry:
  $FR : \{functionality, metaType\} \rightarrow [\{API, configurableParams\}]$
  It specifies which functionality can be enabled on meta-types, which parameters can be configured by EPIS developers and which API must be implemented by micro-services willing to realize the functionality on the meta-type. The content of this registry is derived from the specification of meta-types ad functionalities presented in Section 4.2. As such, FR is typically static, as it must be updated only to introduce new functionalities or new APIs.
- Service registry:
  $SR : \mu service \rightarrow \{functionality, metaType, API, backend, endpoint, [discoveryParams]\}$
  It associates micro-services to information useful for service discovery:
  - the implemented functionality;
  - the supported meta-type;

  - the implemented API (as one functionality can define multiple API for the same meta-type);
  - the adopted storage back-end;
  - the micro-service endpoint;
  - a list of additional parameters useful for discovery purposes.

  The service registry SR is updated every time a micro-service enters (registers to) or exits (de-registers from) the EPMS. The service registration mechanism is not explicitly defined by the architecture so that implementers can decide to opt for a "self-registration" or a "third-party registration" approach.[9]
- Type functionality registry:
  $TFR : \{type, functionality\} \rightarrow [\{service, configuration\}]$
  For each functionality, this registry keeps the association between types of the EP data model, the functionality configuration specified by the EPIS developers and the micro-service that has been assigned to the system to serve the functionality for the given type. The type functionality registry TFR is updated every time EPIS developers configure functionalities and whenever an already assigned service de-registers from the EPMS. In that case the EPMS should be capable of looking for an equivalent micro-service to assign to the type for provisioning the functionality. If there is no equivalent micro-services available, the EPMS should notify the administrator and continue running with degraded functionalities.
- Mapping registry:
  $MR : mappingID \rightarrow \{sourceFormat, targetFormat\}$
  It keeps track of the available mapping services. A mapping service implements a transformation function to transform metadata records from a format to another.
- Executor registry:
  $ER : executor \rightarrow \{executionType, executorParameters\}$
  contains information about the executors that can be associated to executable part types $T_{exe} :: ExecutableType$.

4.4.2 EP data model definition

The modelling areas includes software components that provide EPIS developers with tools for:

- The definition of an EP data model in terms of the EP meta-model;
- The configuration of functionalities on the types of parts defined in the previous step.

The EP data model and the configuration of functionalities defined by the EPIS developer are shared among all software components of the EPMS via the EPMS registries (particularly relevant in this context are the type registry TR, and the type functionality registry TFR).

[9] Service registry pattern: `http://microservices.io/patterns/service-registry.html`. Last accessed: February 2015.

EPIS developers must be provided with a language – textual, graphical, or both – for the definition of the types and relationships that form the EP data model for their target community. Languages for the definition of EP data models are called EP Data Model Definition Languages (EP-DMDLs) and have their foundations on the types of the EP meta-model, i. e. the *meta-types* (see Fig. 4.6). A reference textual EP-DMDL based on the EP meta-model is here presented. The language provides primitives for the creation of types from meta-types (i. e. the types of the EP meta-model).

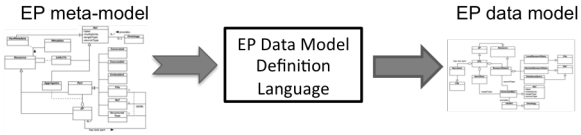


Fig. 4.6. The EP-DMDL is generated from the EP meta-model and allows the definition of personalised EP data models

In addition to the EP-DMDL for the definition of types of the EP data model, EPIS developers must be provided with a mechanism for enabling and configuring functionalities on the defined types. The reference mechanism here proposed is an extension language of the EP-DMDL, which allows EPIS developers to annotate the types they are defining with instructions for the configuration of functionalities. As discussed in Section 4.3, the EPMS must be aware of which functionalities can be enabled for a given type and how one selected functionality can be configured. All this information is kept in the functionality registry (FR) that:

- For each meta-type, keeps track of the functionalities that can be enabled;
- For each pair meta-type and functionality, keeps track of the configuration parameters that EPIS developers can set.

Extensions of the EP-DMDL for configuring specific functionalities are presented in the following subsections.

The EP Data Model Definition Language

The EP-DMDL is a reference textual language for the definition of EP data models. The language provides primitives for the creation of types from meta-types of the EP meta-model. Statements in EP-DMDL for the definition of types of an EP data model have the form:

```
1   CREATE TYPE T :: META-TYPE {
      [meta-type param = value],
3     [instance param = { paramName : string, paramType : simpleType }] ,
      described by = {
5       target = {
          type-ref = T_metadata
7       },
```

```
        allowedLabels = [l1,...lk],
        multiplicityAtSource = mult,
        multiplicityAtTarget = mult
      },
      [rel = {
        target = {
          type-ref = Ttarget
        },
        allowedLabels = [l1,...lk],
        multiplicityAtSource = mult,
        multiplicityAtTarget = mult
      }
      ]
    }
```

where:

- `T` is the name of the type under creation;
- `META-TYPE` is one of the meta-types of the meta-model: *ObjType*, *MetadataFormat*, *FileType*, *RefType*, *StructuredTextType*, *GeneratedType*, *ExecutableType*, *EPType*;
- `meta-type param` is a configurable parameter of the meta-type defined in the meta-type registry ($MTR(META-TYPE)$);
- `instance param` defines a parameter with name `paramName` and type `paramType` that must be set on instances of T. `simpleType` denotes a simple type like `string`, `boolean`, `int`, etc.;
- `described by` introduces the mandatory relationship to a type $T_{metadata} :: MetadataFormat$;
- `rel` is another type of relationship that can occur between instances of `T` and $\texttt{T}_{target}$.

For the creation of types of relationships, the following properties must be supplied:

- `allowedLabels` is the list of strings that can be used as semantic labels of relationships between objects. This allows EPIS developers to define one type of relationship that can be instantiated into relationships with different semantic labels. Labels can be simple strings or entries from a controlled vocabulary registered in the dedicated EPMS registry VocabularyRegistry (VR).
- `multiplicityAtSource` and `multiplicityAtTarget` specify how many relationship of this type can involve the source and the target object, respectively. Allowed values are:
  - $0..1$: at most one relationship can involve the source/target;
  - $0..N$: the relationship is optional and the cardinality unbounded;
  - 1: one and only one relationship must involve the source/target;
  - $1..N$: at least one relationship can involve the source/target

The following paragraphs show how the different meta-types can be instantiated into types.

Defining types with ObjType

ObjType is a meta-type introduced to model entities of the real world as objects that carry only metadata information. Examples are people, organisations and funding programmes. The creation of a type $T_{obj} :: ObjType$ requires the specification of the mandatory relationship `described by` to a type $T_{metadata} :: MetadataFormat$ and, optionally, relationships to other types:

```
CREATE TYPE T :: ObjType {
  described by = {
    target = {
      type-ref = T_metadata
    },
    allowedLabels = [l_1,...l_k],
    multiplicityAtSource = mult,
    multiplicityAtTarget = mult
  },
  [rel = {
    target = {
    type-ref = T_target
    },
    allowedLabels = [l_1,...l_k],
    multiplicityAtSource = mult,
    multiplicityAtTarget = mult
  }
  ]
}
```

The syntax above assumes that the type $T_{metadata} :: MetadataFormat$ has been already defined. A metadata format denotes a tree of properties (or fields). Typical examples of metadata formats defining a structure of a tree are XML schemata (XSD). Being XML the most used encoding for metadata in scientific communication and XSD the most used language for the definition of XML schemata, the definition of $T_{metadata} :: MetadataFormat$ requires to specify the URI pointing to an XSD in the parameter `schema`, as shown in the following template:

```
CREATE TYPE T_metadata :: MetadataFormat {
  schema = URI
}
```

Defining types with FileType

For the definition of a type with meta-type *FileType*, the parameter `format` must be provided. Allowed values are mime type strings like “application/pdf”, “ video/mp4”, “ image/png”.

```
CREATE TYPE T_file :: FileType {
  format = mimeType,
  described by = {
    target = {
```

```
5       type-ref = T_metadata
        },
7       allowedLabels = [l_1,...l_k],
        multiplicityAtSource = mult,
9       multiplicityAtTarget = mult
      },
11     [rel = {
        target = {
13      type-ref = T_target
        },
15      allowedLabels = [l_1,...l_k],
        multiplicityAtSource = mult,
17      multiplicityAtTarget = mult
       }
19     ]
    }
```

Defining types with RefType

Referenced parts represent remote resources accessible via a resolvable identifier such as URLs or PID (e. g. handle, DOI). Referenced parts can be used to represent a variety of digital entities, ranging from remote files to databases. Therefore, the creation of a type $T_R :: RefType$ would usually require the specification of the parameters needed to access the resource. Those parameters are expressed in the template below as list of `instance param`, where each `instance param` specifies the name and type of the parameter.

```
   CREATE TYPE T_R :: RefType {
2    [instance param = { paramName : string, paramType : simpleType }] ,
     described by = {
4      target = {
         type-ref = T_metadata
6      },
       allowedLabels = [l_1,...l_k],
8      multiplicityAtSource = mult,
       multiplicityAtTarget = mult
10   },
     [rel = {
12     target = {
         type-ref = T_target
14     },
       allowedLabels = [l_1,...l_k],
16     multiplicityAtSource = mult,
       multiplicityAtTarget = mult
18   }
     ]
20 }
```

Defining types with StructuredTextType

A structured text is a textual document with explicit "tags" that identify its document components such as title, chapters, figures, tables, and external references. Several formats for structured documents are adopted in scientific communication. Some of them are XML-based, such as the ANSI/NISO standard JATS [5], others are HTML-based, such as Scholarly Markdown [103] and RASH [128].

The template syntax for the creation of a type $T_{ST} :: StructuredType$ is:

```
CREATE TYPE T_ST :: StructuredTextType {
  format = structuredTextFormat,
  described by = {
    target = {
      type-ref = T_metadata
    },
    allowedLabels = [l_1,...l_k],
    multiplicityAtSource = mult,
    multiplicityAtTarget = mult
  },
  [rel = {
    target = {
    type-ref = T_target
    },
    allowedLabels = [l_1,...l_k],
    multiplicityAtSource = mult,
    multiplicityAtTarget = mult
  }
  ]
}
```

where `structuredTextFormat` is a format registered into the dedicated EPMS registry Structured-text format registry (SFR).

Defining types with GeneratedType

Generated parts are parts whose content is materialised on request for visualisation by combining other parts. Examples are dynamic tables that are updated when the underlying data set changes; or a molecule 3D rendering generated by running a 3D molecule viewer with appropriate parameters.

The template for the definition of a type $T_{GEN} :: GeneratedType$ is simple as the template for *ObjType*, as the only mandatory information is about the metadata format that will be used to describe the parts. Additional information is required instead for the configuration of the visualization functionality and it will be discussed in Section 4.4.6.

```
CREATE TYPE T_GEN :: GeneratedType {
  described by = {
    target = {
      type-ref = T_metadata
    },
    allowedLabels = [l_1,...l_k],
```

```
      multiplicityAtSource = mult,
      multiplicityAtTarget = mult
    }
    [rel = {
      target = {
        type-ref = T_target
      },
      allowedLabels = [l_1,...l_k],
      multiplicityAtSource = mult,
      multiplicityAtTarget = mult
    }
    ]
  }
```

Defining types with ExecutableType

Executable parts are parts that can be executed, such as workflows, web service instances, software code. The definition of a type $T_{EXE} :: ExecutableType$ requires the definition of the properties needed to ensure the executability of its parts (`instance param`).

```
CREATE TYPE T_EXE :: ExecutableType{
  [instance param = { paramName : string, paramType : simpleType }]
  described by = {
    target = {
      type-ref = T_metadata
    },
    allowedLabels = [l_1,...l_k],
    multiplicityAtSource = mult,
    multiplicityAtTarget = mult
  }
  [rel = {
    target = {
      type-ref = T_target
    },
    allowedLabels = [l_1,...l_k],
    multiplicityAtSource = mult,
    multiplicityAtTarget = mult
  }
  ]
}
```

Defining types with EPType

Types of EP can be defined as instances of *EPType*. A type of EP $T_{EP} :: EPType$ defines the structure and the semantics of the graphs of EP of the target EPIS. For the creation of an *EPType* T_{EP} it is mandatory to specify:

- which metadata format describes T_{EP} ($T_{metadata} :: MetadataFormat$);
- which type of part is the narration part of T_{EP} ($T_{text} :: NarrationType$);

- which types of parts (T_i) T_{EP} can aggregate.

```
CREATE TYPE T_EP :: EPType {
  described by = {
    target = {
      type-ref = T_metadata
    },
    allowedLabels = [l_1,...l_k],
    multiplicityAtSource = mult,
    multiplicityAtTarget = mult
  },
  has text part = {
    target = {
      type-ref = T_text
    },
    allowedLabels = [l_1,...l_k],
    multiplicityAtSource = 1,
    multiplicityAtTarget = 1
  },
  [aggregates = {
    target = {
      type-ref = T_1
    },
    allowedLabels = [l_1,...l_k],
    multiplicityAtSource = mult,
    multiplicityAtTarget = mult
  }]
}
```

The data model of Example 3, whose visual representation is copied here for convenience in Figure 4.7, can be defined in EP-DMDL as in Listing 4.1.

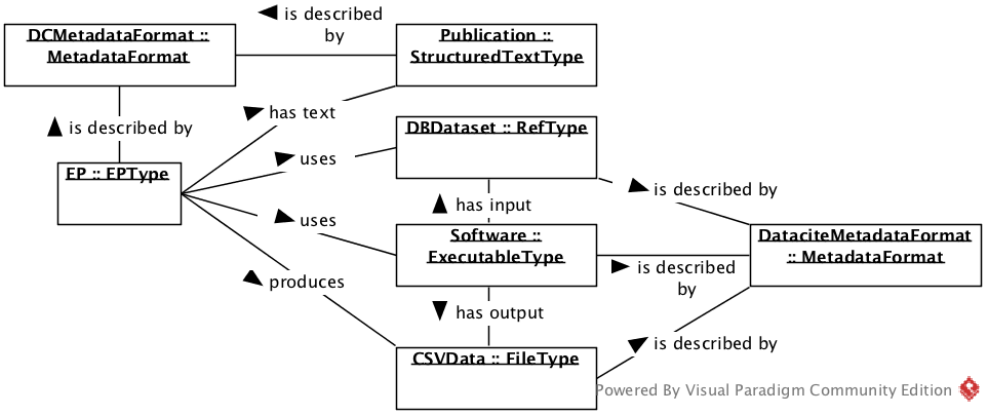


Fig. 4.7. The EP data model of Example 3

```
CREATE TYPE DCMetadataFormat :: MetadataFormat {
  schema = "http://dublincore.org/schemas/xmls/qdc/dc.xsd"
}

CREATE TYPE DataciteMetadataFormat :: MetadataFormat {
  schema = "https://schema.datacite.org/meta/kernel-3/metadata.xsd"
}

CREATE TYPE Publication :: StructuredTextType {
  format = JATS/XML,
  described by = {
    target= {
      type-ref = DCMetadataFormat
    },
    allowedLabels = [is described by],
    multiplicityAtSource = 1,
    multiplicityAtTarget = 1
  }
}

CREATE TYPE DBDataset :: RefType {
  instance param = {
    paramName = dbConnectionString,
    paramType = string
  }] ,
  described by = {
    target = {
      type-ref = DataciteMetadataFormat
    },
     allowedLabels = [is described by],
    multiplicityAtSource = 1,
    multiplicityAtTarget = 1
  },
}

CREATE TYPE CSVData :: FileType {
  format = text/csv
  described by = {
    target = {
      type-ref = DataciteMetadataFormat
    },
    allowedLabels = [is described by],
    multiplicityAtSource = 1,
    multiplicityAtTarget = 1
  },
}

CREATE TYPE Software :: ExecutableType {
  instance param = {
    paramName = downloadURL,
    paramType = URL
```

```
  },
  described by = {
    target = {
      type-ref = DataciteMetadataFormat
    },
     allowedLabels = [is described by],
    multiplicityAtSource = 1,
    multiplicityAtTarget = 1
  },
  hasInput = {
    target = {
      type-ref = DBDataset
    }
    allowedLabels = [has input],
    multiplicityAtSource: 1,
    multiplicityAtTarget: 0..N
  }
  hasOutput = {
    target = {
      type-ref = CSVData
    }
    allowedLabels = [has output],
    multiplicityAtSource: 0..N,
    multiplicityAtTarget: 0..1
  }
}

CREATE TYPE EP :: EPType {
  described by = {
    target = {
      type-ref = DCMetadataFormat
    },
    allowedLabels = [is described by],
    multiplicityAtSource = 1,
    multiplicityAtTarget = 1
  },
  has text part = {
    target = {
      type-ref = Publication
    },
    allowedLabels = [has text],
    multiplicityAtSource = 1,
    multiplicityAtTarget = 1
  },
  aggregates = {
    target = {
      type-ref = DBDataset
    },
    allowedLabels = [uses],
    multiplicityAtSource = 1..N,
    multiplicityAtTarget = 1..N
```

```
  },
  aggregates = {
    target = {
      type-ref = CSVData
    },
    allowedLabels = [produces],
    multiplicityAtSource = 1..N,
    multiplicityAtTarget = 1..N
  },
  aggregates = {
    target = {
      type-ref = Software
    },
    allowedLabels = [uses],
    multiplicityAtSource = 1..N,
    multiplicityAtTarget = 1..N
  }
}
```

Listing 4.1. EP-DMDL for the EP data model in Example 3

4.4.3 Data source mediation

Research products to use for the creation of EPs are often already available from external data sources, such as existing digital libraries, institutional and data repositories. Data sources expose information objects representing research products of three different typologies:

- *Metadata* records are "data about data". They provide a description of one primary (digital) research product and possibly of other products. For example a metadata record about a scientific dataset may contain descriptive information about the dataset (e. g. title, description, keywords, geographical information) and the publications that use that dataset (e. g. titles, DOIs);
- *Payloads* are the actual "data". Examples are the full-text of a scientific publication, a dataset in CSV format.
- *Relationships* are "statements about data" that better qualify or describe one or more (digital) research product with respect to another digital entity. Relationships typically have a label that expresses the semantics of the association, though in some cases the label may not exist because the semantics is implicit or not known. Relationships are typically expressed in the form of triples <source, label, target>. For example <doi:123, - , doi:456> represents a relationships between two research products with stable identifiers (DOIs) with unknown semantics. <doi:123, isCitedBy , doi:456> represents instead a relationship with known semantics ("is cited by") between the two research products. Typically, relationships are encoded in dedicated fields of the metadata of the related products.

  Data sources differ for the export interfaces they offer, in terms of:

- Exchange protocols (e. g. Java Database Connectivity (JDBC), OAI-PMH, Web Services);
- Format of data (e. g. PDF, XML, CSV) and metadata (e. g. Json, XML);
- Data model (ranging from standards such as Dublin Core (DC), Encoded Archival Description (EAD), and Common European Research Information Format (CERIF), to idiosyncratic data models).

Solving data interoperability issues is a complex problem [75, 108, 18, 19, 106] that is out of the scope of this thesis, hence it is assumed that data sources can offer an HTTP API that allows to:

- search for research products;
- get metadata of a research product given its identifier;
- get the payload of a research product given its identifier;

As discussed in Section 3.2.1, the data sources to be integrated depend on the application domain of the EPIS target community: in Life Science interesting data sources are databases of biological entities (e. g. UniProtKB [16], European Nucleotide Archive [101]), Life Science literature repositories (e. g. Europe PMC [112], PubMed Central [141]) and biological ontologies (e. g. Gene Ontology [13], NCBI Taxonomy [58]). In social science interesting data sources are instead archives of social science studies (e. g. DANS[10], national and international surveys (e. g. International Social Survey Programme [148], European Social Survey[11]) and literature repositories in the field (e. g. Social Science Open Access Repository[12], Social Science Research Network[13]).

The EPMS allows EPIS developers to specify the data sources from which parts of a given type can be collected via a dedicated annotation on the type definition:

```
@CollectFrom( [ {dataSource, objectType} ] )
```

where:

- `dataSource` is one of data sources selected by the EPIS developer from the full list of supported data sources, which is kept in the data source registry ($DSR : dataSource \rightarrow [accessParams]$);
- `objectType` is the type of objects (metadata or payload) to collect from `dataSource`.

Listing 4.2 shows how the type $Publication :: StructuredTextType$ of Example 3 (see also Figure 4.7 and Listing 4.1) can be configured to instruct the collection of metadata and payloads from one specific data source (europePMC) registered in the EPMS data source registry DSR.

```
1  CREATE TYPE Publication :: StructuredTextType {
    @CollectFrom(europePMC, payload)
3   format = JATS/XML,
```

[10] Data Archiving and Networked Services (DANS): `https://easy.dans.knaw.nl`
[11] European Social Survey (ESS): `http://www.europeansocialsurvey.org`
[12] Social Science Open Access Repository (SSOAR): `http://www.ssoar.info`
[13] Social Science Research Network (SSRN): `http://www.ssrn.com`

```
    described by = {
5    target= {
      @CollectFrom(europePMC, metadata)
7     type-ref = DCMetadataFormat,
     },
9    multiplicityAtSource = 1,
     multiplicityAtTarget = 1
11  }
   }
```

Listing 4.2. Configuring the type Publication of Example 3 for the collection of metadata and payloads from europePMC

When using the `@CollectFrom` annotation, the type functionality registry TFR is affected. The TFR keeps the association between types of the EP data model, the functionality configuration specified by the EPIS developers and the micro-service that has been assigned to the system to serve the functionality for the given type. Let's assume there is a service $service_{epmc}$ registered in the EPMS service registry SR for the collection of objects with meta-type *StructuredTextType* from the data source europePMC:

$$SR : service_{epmc} \rightarrow$$
$$\{collection, StructuredTextType, API_{collect}, europePMC, http://endpointURL\}$$

Based on the annotations in Listing 4.2, the TFR is updated to contain two additional entries for the collection functionality on the type `Publication`:

$$TFR : \{type, functionality\} \rightarrow [service, configuration]$$
$$TFR : \{Publication, collection\} \rightarrow$$
$$[\{service_{epmc}, \{dataSource = europePMC, objectType = payload\}\}]$$
$$TFR : \{Publication, collection\} \rightarrow$$
$$[\{service_{epmc}, \{dataSource = europePMC,$$
$$objectType = metadata, targetType = DCMetadataFormat\}\}]$$

Those two entries assert that:

1. parts of type `Publication` ($p : Publication$) can be created by collecting objects from the data source europePMC;
2. the service to collect payloads and metadata is $service_{epmc}$;
3. metadata collected by $service_{epmc}$ for $p : Publication$ must be used to create instances of `DCMetadataFormat` ($md : DCMetadataFormat$).

4.4.4 EP graph management

One of the requirements of EPMSs described in Section 4.3 is that of being able to manage graphs of EPs. The EPMS must therefore provide components for CRUD operations on EPs and their parts and for the navigation of the graph. As shown in Figure 4.8, the reference

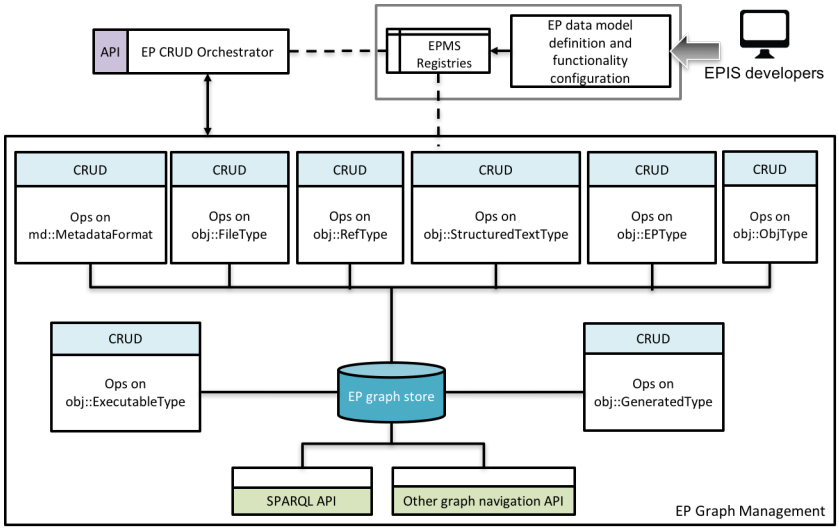


Fig. 4.8. Software reference architecture for EP graph management

architecture features a set of micro-services that implement CRUD operations on parts with different meta-types. Those operations affect the *EP graph store*, which is the core back-end for the materialisation of the EP graphs. The same back-end can be also used to offer navigation functionality over the graphs of EPs, thanks to services that expose dedicated API, for example SPARQL.

CRUD operations on EP graphs

The case in Figure 4.9 describes the message flows for the addition of a new part to an existing EP. From the GUI, the user selects a product *<p>* from the data source *<d>* and ask to import it as part of type *<type>*. In order to show the details of *<p>*, the GUI must communicate with the data source *<d>* in order to import the metadata and, possibly, the payload of the selected product. The discovery of the endpoint of the data source *<d>* is performed via lookup on the EPMS data source registry DSR. The user then decides to add the part to an existing EP *<ep>*. He also specifies the label for the aggregation relationship that must link *<ep>* to the part). The GUI communicates the request to the *EP CRUD orchestrator*, which is responsible to coordinate calls to services for the realisation of CRUD operations. In fact, CRUD operations on EP are actions that do not affect a single functionality: several services belonging to different functional areas must be notified based on the EP data model configuration (e. g. trigger the update of a full-text index, include/exclude parts or EPs from exports). This can be done either via *choreography* or *orchestration*. With a choreography approach, the request for a CRUD operation is sent to all interested services (e. g. via a shared message queue). Services re-act upon the reception of the message and perform the needed tasks. Choreography has the benefit of simplifying the integration of new services and functionalities, as the new component can simply join the infrastructure and re-act when receiving messages it is interested in. The

main drawback of this approach is that it is hard to track the control flow that realizes the operation at hand. The orchestration approach, instead, introduces a central point that manages the control flow. If a new service must re-act to a message, the orchestrator must be aware of it so that the service can be properly introduced in the control flow. In our specific case of EPMS, the orchestration approach seems to be a more sustainable choice. In fact, the control flow for the realisation of a CRUD operation is defined by the EPIS developers when configuring functionalities on the EP data model. For this reason, the architecture features an orchestrator component, the *EP CRUD orchestrator*.

The identification of the services to involve for the realization of an action is performed by looking up into the EPMS registries. In particular, the type functionality registry TFR contains all information that the EP CRUD orchestrator needs: for each type, the registry can tell which functionality has been enabled, how they have been configured and which is the responsible service to call. The orchestrator can then call the services and aggregate their responses to provide a comprehensive feedback to the caller.

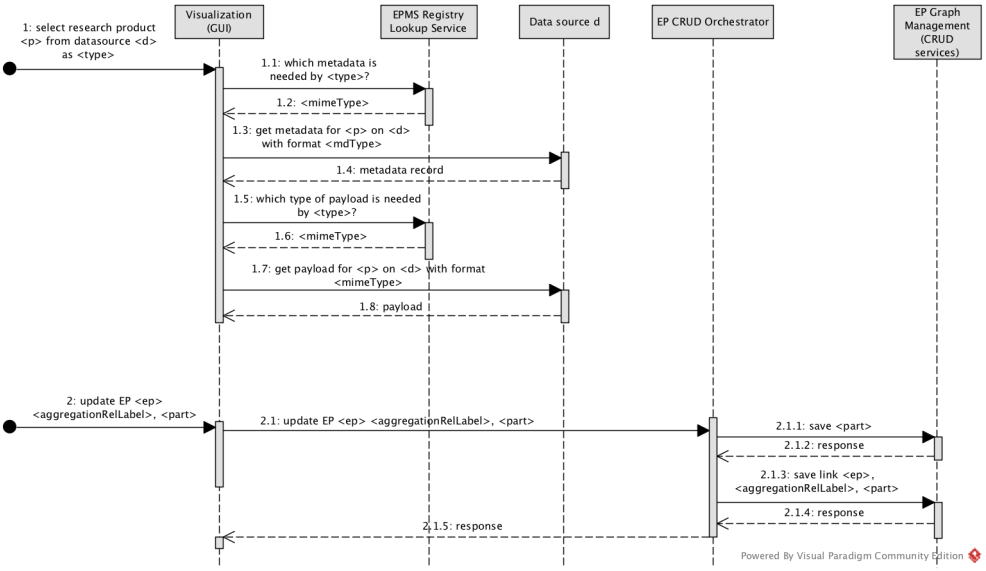


Fig. 4.9. Updating an EP

4.4.5 Search and browse

EPs should be searchable and browsable (via faceted search) based on their metadata, the metadata of their parts, and the payload of their parts (when applicable, for example for parts with meta-type *FileType* and *StructuredTextType*).

Figure 4.10 describes the interactions between components of the EPMS for replying to a search request made by a user from the GUI. The GUI receives the request for searching EPs of a given type that satisfy some criteria that can be expressed as a query. The

request must be forwarded to the search service responsible for that type. The service lookup on the EPMS registries can be performed via the *EPMS Registry Lookup Service*. A server-side component of the visualisation area can then call the search API of the selected service (`service1`). The service is responsible for rewriting the query to send to the *Search back-end*, in order to match the expected query language, and returns the response to the caller. Finally, the response can be processed and formatted for its visualization in the GUI.

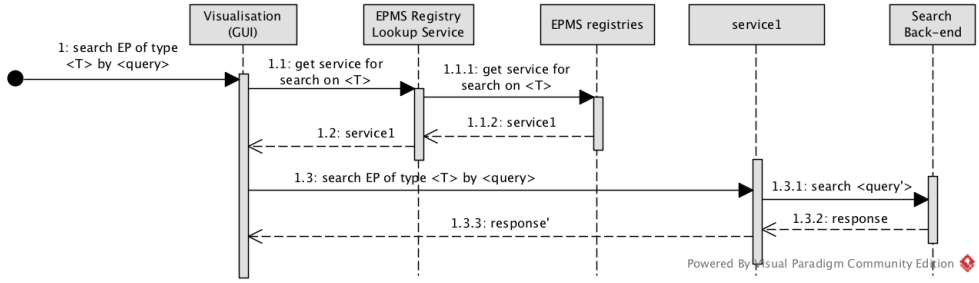


Fig. 4.10. Searching an EP

User queries can be formulated on the fields that EPIS developers have specified in the configuration of the EP data model. Regarding metadata records of EPs and parts, they contain descriptive information that is very useful for discovery purposes. Their structured and semantic nature enables the creation of one index per metadata field in order to support the formulation of specific queries. The EPMS should support the creation of such indices and allow EPIS developers to specify the mapping between paths of the tree structure of the metadata format and index field names, which can be used to formulate queries.

The annotation for the configuration of the search functionality on metadata records of EPs and parts is:

```
@Search(backend = backEndName,
  indexFields = [{indexField=fieldName, sourceField=sourceFieldPath}],
  browseFields = [{browseField=fieldName, sourceField=sourceFieldPath}])
```

where:

- `backend` identifies the search back-end that the EPIS developers want to use. This will be used for the discovery of the search service to associate to the annotated type in the type functionality registry TFR;
- `indexFields` specifies the configuration of the service for the creation of indices on the fields of the metadata format. The configuration to be provided is the mapping between paths of the tree structure of the metadata format (`sourceFieldPath`) and index field names (`fieldName`), which can be used to formulate queries;
- `browseFields` specifies the configuration of the service for the creation of indices for faceted search. The configuration to be provided is the mapping between paths of

the tree structure of the metadata format (`sourceFieldPath`) and index field names (`fieldName`), which can be used to formulate browse queries.

Listing 4.3 shows a possible configuration for the search and browse functionality on the metadata of the type $EP :: EPType$ that has been defined in Listing 4.1. The configuration specifies that instances of `EP` ($ep : EP :: EPType$) should be searchable via the associated metadata records ($md : DCMetadataFormat :: MetadataFormat$). The selected back-end is `solr` and should be configured with two index fields and one browse fields:

- the `author` index field must index the values of the metadata path `//creator`;
- the `title` index field must index the values of the metadata path `//title`;
- the `subject` browse field must index the values in the metadata path `//subject`.

```
CREATE TYPE EP :: EPType {
  described by = {
  target = {
    @Search(backend="solr",
    indexFields = [
      {indexField="author", sourceField="//creator"},
      {indexField="title", sourceField="//title"}
      ],
      browseFields = [{browseField="subject", sourceField="//subject"}])
      type-ref = DCMetadataFormat
    },
    multiplicityAtSource = 1,
    multiplicityAtTarget = 1
  }
  . . .
}
```

Listing 4.3. Annotating $EP :: EPType$ for the configuration of search and browse functionality on its associated metadata format

Likely, metadata formats associated to parts can also be annotated. Listing 4.4 shows a possible annotation on the metadata format $DataciteMetadataFormat :: MetadataFormat$ associated to type $DBDataset :: RefType$. The configuration specifies that instances of `DBDataset` ($d : DBDataset :: RefType$) should be searchable via the associated metadata records ($md : DataciteMetadataFormat :: MetadataFormat$). The selected back-end is `solr` and should be configured with three index fields and one browse fields:

- the `author` index field must index the values of the metadata path `//creator/creatorName`;
- the `title` index field must index the values of the metadata path `//titles/title`;
- the `abstract` index field must index the values of the metadata path `//descriptions/description[./@descriptionType="Abstract"]`;
- the `subject` browse field must index the values in the metadata path `//subjects/subject`.

```
CREATE TYPE DBDataset :: RefType {
  . . .
  described by = {
    target = {
      @Search(backend="solr",
      indexFields = [
        {indexField = "author", sourceField = "//creator/creatorName"},
        {indexField = "title", sourceField = "//titles/title"},
        {indexField = "abstract", sourceField =
          "//descriptions/description[./@descriptionType='Abstract']"}
      ],
      browseFields = [{browseField = "subject",
        sourceField = "//subjects/subject"}
      ])
      type-ref = DataciteMetadataFormat
    },
    multiplicityAtSource = 1,
    multiplicityAtTarget = 1
  }
}
```

Listing 4.4. Annotating $DBDataset :: RefType$ for the configuration of search and browse functionality on its associated metadata format

Parts that include a payload such as those with meta-types *StructuredTextType* and *FileType* can also be annotated to instruct the EPMS to index the payload content. Listing 4.5 shows how the type $Publication :: StructuredText$ can be annotated to set-up a full-text index on the payloads of its instances. Also, additional index fields are configured for specific document editorial components: the abstract and the methods section.

```
CREATE TYPE Publication :: StructuredTextType {
  format = JATS/XML,
  described by = {
    target= { type-ref = DCMetadataFormat},
    multiplicityAtSource = 1,
    multiplicityAtTarget = 1
  }
  @Search(indexFields = [
    {indexField = all, sourceField = "//article"},
    {indexField = abstract,
      sourceField = "//article/front/article-meta/abstract"},
    {indexField = methods,
      sourceField = "//article/body/sec[@sec-type='methods']"}
    ])
}
```

Listing 4.5. Annotating $Publication :: StructuredText$ for the configuration of search and browse functionality on its payloads

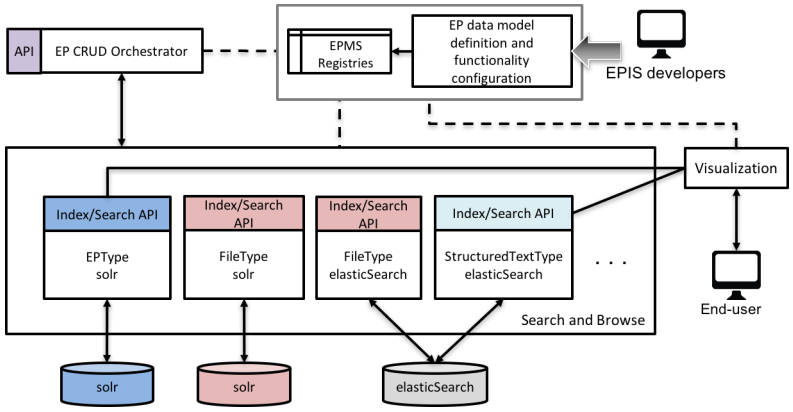


Fig. 4.11. Software reference architecture for the search and browse functional area

4.4.6 Visualization

The EPMS should offer ready-to-use GUI for creating, searching, visualizing and navigating EPs compliant to the EP data model defined by the EPIS developers. Such a customised GUI can be made available out-of-the-box by exploiting the structure of types and the configuration of functionalities defined in the EP data model.

Metadata records are structured data whose form is defined by a metadata format. By exploiting the information provided by a metadata format, the EPMS can automatically generate GUI components for the manual creation and visualization of metadata records. An advanced configuration of the GUI can be made available in order to allow EPIS developers to select what to visualise and what to hide to end-users. For example, some metadata formats define fields with information useful to humans together with fields that ease machine interpretation. In this case, the EPIS developers may decide to hide to the end-user the fields that are not "human-oriented". In other cases, the metadata records contain information that should not be publicly available for privacy reasons, such as e-mails and phone numbers of people. EPIS developers can instruct the GUI to not show the values on those metadata fields with annotation in the form:

```
1  @Visualize(hideFields = [fieldPath])
```

where `hideFields` contains the list of metadata field paths that should be hidden to the end-user.

Visualization options can also be configured on types with meta-type *FileType*. In this case the mime type of the payload is an important property to consider for the proper visualization of the parts. For example, EPIS developers may want to enable the visualization of a video in a player and images to be visualised as thumbnails. By annotating a type $T :: FileType$ with `@Visualize`, EPIS developers can instruct the system to find a registered service that is able to visualize file parts with the mime type defined in T. Let's suppose that a service for video streaming $service_V$ is registered in the EPMS and that a type $Video :: FileType$ has been defined as in Listing 4.6, so that the EPMS registries

- functionality registry:
  $FR : \{functionality, metaType\} \rightarrow [\{API, configurableParams\}]$
- service registry:
  $SR : \mu service \rightarrow$
  $\{functionality, metaType, API, backend, endpoint, [discoveryParams]\}$
- type registry: $TR : type \rightarrow \{metaType, configuration\}$

are in the following status:

$$FR : \{visualization, FileType\} \rightarrow \{API_V, -\}$$
$$SR : service_V \rightarrow$$
$$\{visualization, FileType, API_V, backend_V, endpoint_V,$$
$$[format = \{video/mp4, video/avi\}]\}$$
$$TR : Video \rightarrow \{FileType, format : video/mp4\}$$

```
CREATE TYPE Video :: FileType {
  @Visualize
  format = video/mp4
  described by = {
    target = {
      type-ref = DataciteMetadataFormat
    },
  multiplicityAtSource = 1,
  multiplicityAtTarget = 1
 }
}
```

Listing 4.6. Example of *FileType* with annotation to configure the visualization of its instances

The `@Visualize` annotation is processed by the EPMS, $service_V$ is identified as the proper service to handle the visualization of $video : Video$ with "video/mp4" mime type and a new entry is added to the type functionality registry TFR:

$$TFR : \{Video, visualization\} \rightarrow \{service_V, \{mimetype = video/mp4\}\}$$

Generated parts also deserve some considerations. Generated parts are parts whose content is materialised on request for visualisation by combining other parts. Examples are dynamic tables that are updated when the underlying data set changes; or a molecule 3D rendering generated by running a 3D molecule viewer. The visualization functionality on generated parts must be configured in terms of the types of parts that are input of the generation process (`input`) and the visualization service that implements the generation process (`generator`):

$$FR : \{visualization, GeneratedType\} \rightarrow$$
$$\{API_G, \{generator : serviceID, input : [Type]\}$$
$$SR : service_G \rightarrow \{visualization, GeneratedType, API_G, backend_G, endpoint_G, []\}$$

The visualization functionality can then be configured with
`@Visualize(generator=serviceID, input = [Type])`

Listing 4.7 shows how the type $Table :: GeneratedType$ is annotated so that its instances are visualized via the service $\texttt{service}_G$ using instances of `DBDataset`.

```
CREATE TYPE Table :: GeneratedType {
  described by = {
    target = {
      type-ref = DataciteMetadataFormat
    },
    multiplicityAtSource = 1,
    multiplicityAtTarget = 1
  }
  relData = {
    target = {
      type-ref= DBDataset
    },
    allowedLabels = [shows content from]
    multiplicityAtSource = 1,
    multiplicityAtTarget = 0..N
  }
  @Visualize(generator=service_G, input = [relData.target.type-ref])
}
```

Listing 4.7. Example of *GeneratedType* with annotation to configure the visualization of its instances

4.4.7 Export

Section 4.3 listed the availability of APIs for EP sharing among the requirements of an EPMS and identified the most relevant metadata and data exchange protocols for the scientific communication domain that should be natively supported by an EPMS:

- OAI-PMH for the export of metadata records about research products;
- OAI-ORE for the export of EPs as aggregations of resources;
- Linked Data for the export and navigation of the EP graphs.

In addition, it should be possible to support the export of payloads of parts via standard file exchange protocols such as FTP and HTTP.

OAI-PMH

The Open Archive Initiative - Protocol for Metadata Harvesting (OAI-PMH) protocol [97, 45] is particularly relevant to enable metadata interoperability among digital libraries, institutional and thematic repositories. OAI-PMH provides an application-independent interoperability framework based on metadata harvesting. Two classes of actors participate in the OAI-PMH framework: data providers, which expose metadata records via an OAI-PMH publisher service over HTTP; and service providers, or aggregators, which harvest metadata

records from data providers. Metadata records exported by providers can be grouped into OAI sets, so that harvesters can collect only a portion of the exported metadata records. The protocol also mandates that metadata records must be available at least in DC XML format. The EPMS should allow EPIS developers to configure:

1. the types whose metadata must be exported;
2. an optional mapping rule to transform metadata records of the selected type into the DC format;
3. an optional OAI set to group metadata records;
4. an optional list of metadata formats and mapping rules for the export of metadata records in formats different from DC.

This can be realised with the `@OAIPMH` annotation

```
@OAIPMH(DCMapping=mappingID,
  set={setSpec: string, setName: string, setDescription: string},
  mdFormats=[{prefix: string, schema: URL,
    namespace: URL, mapping: mappingID}])
```

where:

- `DCMapping` is an optional parameter that identifies the mapping to be used to transform metadata records into the DC format. Mappings are services registered in the EPMS mapping registry (MR) that process metadata records compliant to a format and produce XML records in a target format (DC, in this case). If the parameter is omitted the EPMS assumes that the metadata records are already in DC format.
- `set` is an optional parameter that specifies in which OAI set the metadata records must be grouped. According to the OAI-PMH specifications, an Open Archive Initiative (OAI) set must have an identifier (`setSpec`), a name (`setName`) and an optional description (`setDescription`);
- `mdFormats` is an optional list of additional formats available for the export. The EPIS developer must specify for each export metadata format:
  - the metadata `prefix`, which is a sort of identifier of the metadata format, in OAI-PMH jargon;
  - the metadata `schema`, i. e. the URL to the XML schema;
  - the metadata `namespace`, i. e. the URL to the XML namespace associated to the format;
  - an optional `mapping` to use for transforming the metadata records. If omitted the EPMS can assume that the metadata records are already in the export format.

Let's suppose a mapping service $datacite2dc$ is registered into the EPMS and that we want to export via OAI-PMH the metadata about $Publication$ and $DBDataset$ of the EP data model of Example 3 and Listing 4.1:

```
CREATE TYPE Publication :: StructuredTextType {
  format = JATS/XML,
  described by = {
    target= {
```

```
      @OAIPMH(set={setSpec = 'pubs', setName = 'publications',
6       setDescription = 'Metadata␣about␣publications'})
        type-ref = DCMetadataFormat
8   },
      multiplicityAtSource = 1,
10    multiplicityAtTarget = 1
    }
12 }

14 CREATE TYPE DBDataset :: RefType {
    instance param = {
16    paramName = dbConnectionString,
      paramType = string
18  }] ,
    described by = {
20    target = {
      @OAIPMH(DCMapping = datacite2dc,
22      set = {setSpec = 'datasets', setName = 'remoteDatasets',
        setDescription = 'Metadata␣about␣remote␣datasets'},
        mdFormats=[{prefix = 'oai_datacite',
24      schema = 'https://schema.datacite.org/meta/kernel-3/metadata.xsd',
        namespace ='http://datacite.org/schema/kernel-3'}])
26      type-ref = DataciteMetadataFormat
      },
28    multiplicityAtSource = 1,
      multiplicityAtTarget = 1
30  }
  }
```

Listing 4.8. Example of configuration for OAI-PMH export of metadata records

The configuration in Listing 4.8 instructs the OAI-PMH publisher service registered in the EPMS to export the metadata records associated to instances of $Publication :: StructuredTextType$ and $DBDataset :: RefType$ in two different sets (`pubs` and `datasets`, respectively). Records in the `pubs` set must be available in DC format, while records in the `datasets` set must be available both in DC and Datacite formats. The mapping from $DataciteMetadataFormat$ to DC is performed by the `datacite2dc` mapping service. Since the availability of metadata records in DC format is mandatory to comply with the OAI-PMH specification, the DC format is supported natively and the EPIS developer does not need to specify the DC metadata format details. On the other hand, to enable the export in Datacite metadata format, the EPIS developer must specify the details of the metadata format.

OAI-ORE

The Open Archive Initiative - Object Reuse and Exchange (OAI-ORE) protocol [98, 95, 96] defines standards for the description and exchange of aggregations of web resources. EPs are aggregations of resources by definition, so the OAI-ORE protocol and its abstract model fits naturally in this scenario. The OAI-ORE protocol defines the notion *ORE re-*

sources. ORE resources are items of interest identified by a URI and can be linked with each others. URI of ORE resources can be dereferenced to obtain their *ORE representation*s, i. e. their content. *ORE aggregation*s are collections of ORE resources. Aggregations are ORE resources themselves, so they are identified by a URI. The description of the structure of an ORE aggregation is called *ORE resource map*. A resource map is a web document in RDF format that contains the list of aggregated resources, their metadata, and the metadata about the aggregation itself.

In order to set-up an OAI-ORE export of EPs ($ep :: EPType$) the EPIS developer must specify:

- Which types of parts should be made available via OAI-ORE;
- A base URL to use for the construction of the URIs for ORE resources.

Listing 4.9 shows a possible configuration of $EP :: EPType$ of Example 3 and Listing 4.1. The type $EP :: EPType$ is annotated with

```
@OAIORE(baseURL='http://example.ep.eu')
```

to specify the base URL that should be used for aggregations, resource maps and the aggregated resources.[14] The types of parts to be included in the aggregation are annotated with `@OAIORE`. In Listing 4.9, for example, the type $CSVData :: FileType$ is not annotated, so its instances won't be included in the resource map of the aggregation (Line 28). Metadata records associated to an EP and aggregated parts will be used to complete the resource map with metadata information.

```
CREATE TYPE EP :: EPType {
  @OAIORE(baseURL='http://example.ep.eu')
  described by = {
    target = {
      type-ref = DCMetadataFormat,
    },
    multiplicityAtSource = 1,
    multiplicityAtTarget = 1
  },
  has text part = {
    target = {
      @OAIORE
      type-ref = Publication
    },
    multiplicityAtSource = 1,
    multiplicityAtTarget = 1
  },
  aggregates = {
    target = {
```

[14] Services realizing the functionality must therefore implement a strategy for the creation of URLs, for example by appending a constant string that identify the type of ORE resource (e. g. "aggregation", "remap", "resource") and a stable identifier of the resource (e. g. the identifier assigned by the EPMS). For example, the URL for the ORE aggregation of an $ep : EP$ with id 123 could be "http://example.ep.eu/aggregation/123", while the URL of its resource map could be "http://example.ep.eu/remap/123".

```
20        @OAIORE
          type-ref = DBDataset
22      },
        multiplicityAtSource = 1..N,
24      multiplicityAtTarget = 1..N
      },
26    aggregates = {
        target = {
28        type-ref = CSVData
        },
30      multiplicityAtSource = 1..N,
        multiplicityAtTarget = 1..N
32    },
      aggregates = {
34      target = {
          @OAIORE
36        type-ref = Software
        },
38      multiplicityAtSource = 1..N,
        multiplicityAtTarget = 1..N
40    }
    }
```

Listing 4.9. Example of configuration for OAI-ORE export of $EP :: EPType$

Linked Data

The term Linked Data refers to a set of best practices for publishing and interlinking structured data on the Web. The principles were first introduced by Tim Berners-Lee in [24] and consist of:

1. Using URIs as names for things.
2. Using HTTP URIs, so that people can look up those names.
3. Providing useful information when someone looks up a URI, using the standards (RDF, SPARQL).
4. Including links to other URIs, so that clients can discover more things.

The OAI-ORE protocol described in the previous paragraph covers the above principles, however it does not allow to customize the semantics of the aggregation relationships and does not directly address the provision of a SPARQL endpoint [68], as suggested by the third principle. Supporting SPARQL, in particular, enables clients to navigate the graphs of EPs materialised in the EP graph store of the EPMS. Typically, graph stores and triplestores that can be adopted as EP graph store natively offer a SPARQL interface.

FTP/HTTP exports

Exporting the payloads of instances with meta-type *StructuredTextType* and *FileType* may be configured via the `@FileExport(protocol : string)` annotation, as shown in the following listing:

```
CREATE TYPE Publication :: StructuredTextType {
  @FileExport(protocol:'http')
  format = JATS/XML,
  described by = {
    target= {
      type-ref = DCMetadataFormat,
    },
    multiplicityAtSource = 1,
    multiplicityAtTarget = 1
  }
}
CREATE TYPE CSVData :: FileType {
@FileExport(protocol:'ftp')
  format = text/csv
  described by = {
    target = {
      type-ref = DataciteMetadataFormat,
    },
    multiplicityAtSource = 1,
    multiplicityAtTarget = 1
  }
}
```

The types $Publication :: StructuredTextType$ $CSVData :: FileType$ are annotated to export the payload of their instances via HTTP and FTP, respectively.

Finally, an EP should also be made exportable as a "self-contained package" that includes all metadata and payloads of its parts together with information about its structure. Structural information may be encoded as a METS[15] document that specifies how the parts are related to each other.

4.4.8 Execution

Data-intensive e-Science brought in scientific communication the novel requirements of disseminating traditional digital publications with a "research experiment context", which would allow for better interpretation and validation by-repetition of the research conducted. Executable parts of EPs carry information required to execute a process, such as a reference to a web service used in an experiment, a workflow to be executed by a given engine, or, more generally, code which can be dynamically executed by a given run-time. The execution of an executable part can occur:

- in the EPIS, whenever the execution context and engine is integrated by the underlying EPMS;

[15] Metadata Encoding and Transmission Standard (METS) is a standard maintained in the Network Development and MARC Standards Office of the Library of Congress for encoding descriptive, administrative, and structural metadata regarding objects within a digital library `https://www.loc.gov/standards/mets/mets-home.html`. Last accessed: February 2016.

- in external (web) applications, where the user is redirected when requesting the execution (e. g. workflows in myexperiments.org [142] or a Taverna workbench[16]);
- in the computer of researchers, who must install and deploy software. In this case the executable part must include software (e. g. SVN references), how-to manuals, or configurations (e. g. virtual machine images [65]).

When creating a type $T_{EXE} :: ExecutableType$ EPIS developers must know which execution engines are supported by the EPMS as external and integrated engines in order to select the one that is appropriate for the instances of the type at hand. The information about the available execution engines is kept in the EPMS executor registry. The role of the executor services is that of creating a "bridge" between the EPMS and the execution engines, providing functionalities for the realization of the three execution modes listed above.

The configuration of a type $T_{EXE} :: ExecutableType$ for execution can be performed via the `@Execute` annotation:

```
CREATE TYPE T_EXE :: ExecutableType{
  @Execute(executor : executorID,
      executorParams: [{executorParamName : string,
                        value : instanceParamName }])
  [instance param = { paramName : string, paramType : simpleType }]
  . . .
}
```

where

- `executorID` is the identifier of a registered executor;
- texttttexecutorParams is a list that instruct the system about the instance parameters (`instanceParamName`) to pass to the parameters required by the executor (`executorParamName`);
- `instance param` defines a parameter that must be set when a part of type T_{EXE} is created.

[16] Taverna: powerful, scalable, open source and domain independent tools for designing and executing workflows. `http://www.taverna.org.uk`. Last accessed: February 2016.

5

Conclusion and future work

The attention on the paradigm of Enhanced Publication (EP) for scientific communication is growing in the research community, also thanks to mandates of funding agencies and organizations, which are advocating for publishing and citing any type of research product, not only the scientific publication, in order to measure their return of investment, improve their funding strategies, or gain visibility and scientific rewards. However, theory and practice of EPs is still not advanced and lacks of structured foundations and general-purpose technical solutions.

The analysis of the state of the art showed how existing Enhanced Publication Information System (EPIS) tend to be delivered adopting ad-hoc technical implementations. Their feature is that of thoroughly satisfying the needs of the final users for whom they are conceived, but in general they fail to be re-used in different contexts, where data model and functional requirements may slightly or heavily change. Though community-specific EPISs are effective for the community they target, they typically entail non-negligible realisation and maintenance costs. Because of the lack of EP-oriented tools, EPIS developers realise EPIS by integrating technologies that are general purpose (e. g. databases, file stores) and Digital Library (DL)-oriented (e. g. repository software, cataloguing systems), re-implementing every time community-independent functionalities.

This study contributes to the field of scientific communication in several ways. First, it provides the necessary terminology to describe and classify EP data models in terms of their structural and semantic features. The analysis of the state of the art resulted in the identification of the typologies of parts that can form EPs. Those have been used as building blocks for a a reference data model for Enhanced Publication Management Systems (EPMSs), called *EP meta-model*. Second, it proposes a classification of existing EP data models and EPISs in terms of their functional goals. Such classification proved to be useful for the comparison and discussion of technical solutions for the management of EPs. Third, a systemic approach, based on the novel notion of EPMS, is proposed as a more cost effective solution to the realization of EPISs, compared to the "from scratch" strategy. EPMS are information systems devised for offering EP-oriented tools for the realisation of customised EPISs. By using an EPMS, EPIS developers must not imple-

ment functionality for EP access, manipulation, retrieval, and export, because those are natively provided by the EPMS. Developers can instead focus on the configuration of the EP data model and on the functionalities that must be provided to the end-users of the EPIS. Last, a reference software architecture for EPMSs is presented, with the goal of setting up the foundations for the design of concrete EPMSs implementations that support the realisation, maintenance and operation of customised EPISs.

The software reference architecture can be seen as a guide for bringing into the scientific communication and the EP world the benefits introduced by the adoption of Data Base Management Systems (DBMSs) in the database scenario. Among all benefits, the following have been highlighted: general-purposness; technology transparency; availability of Application Programming Interface (API) for data sharing; availability of languages for data definition, manipulation and access; extensibility. The architecture adopts the generic data model derived from the analysis of the state of the art (the "EP meta-model") and identifies all functionalities that should be available on each type of the model. It also offers the EP Data Model Definition Language (EP-DMDL) for the definition of customised EP data models and the configuration of functionalities on types of the defined EP model.

At the current stage, the reference architecture does not support natively the versioning of EPs and parts: developers must explicitly define versioning relationships between instances. Those are seen by the EPMS and its services as normal relationships between parts so that no specific support can be provided. In addition, the search and browse functionality is not publicly exposed but it serves only the other component of the EPMS, in particular those of the visualization area through which end-users search and browse for EPs. This decision has been initially taken considering that selective access via search and browse would usually be performed by humans, via the provided Graphical User Interface (GUI), while machines would rather opt for one of the protocols included in the export functional area (Open Archive Initiative - Protocol for Metadata Harvesting (OAI-PMH), Open Archive Initiative - Object Reuse and Exchange (OAI-ORE), SPARQL Protocol and RDF Query Language (SPARQL)). Both the versioning support and the exposure of a public search API are to be considered in the next version of the reference architecture, to be delivered in the near future.

Future work also includes the implementation of an EPMS and the definition of quality metrics for measuring its compliance and correctness with regard to the reference architecture.

It is expected that the introduction of the reference software architecture and its implementation in the scientific communication scenario will help in widening the adoption and usage of EPs and EPISs, today hindered by their high costs of realisation, operation, and maintenance.

References

1. CrossRef. http://crossref.org/. Last visited: 11 March 2014.
2. Data archiving and networked services. http://www.dans.knaw.nl/en. Last visited: 2 March 2014.
3. Datacite. http://datacite.org/. Last visited:2013-02-13.
4. Force11: the future of research communication and e-scholarship. http://www.force11.org.
5. JATS: Journal Article Tag Suite. http://jats.niso.org/. Last visited: 2016-01-20.
6. Joint Data Archiving Policy (JDAP). http://datadryad.org/pages/jdap. Last visited: 2014-03-10.
7. RDFa. https://rdfa.info. Last visited: February 2016.
8. The British Atmospheric Data Centre. http://badc.nerc.ac.uk/home/index.html, December 2013. Last visited: 2 March 2014.
9. IJsbrand Jan Aalbersberg, Frans Heeman, Hylke Koers, and Elena Zudilova-Seinstra. Elsevier's article of the future enhancing the user experience and integrating data through applications. *Insights: the UKSG journal*, 25(1):33–43, 03 2012.
10. Sophia Krzys Acord and Diane Harley. Credit, time, and personality: The human challenges to sharing scholarly work using web 2.0. *New Media and Society*, 15(3):379–397, 2013.
11. Suzie Allard. Dataone: Facilitating escience through collaboration. *Journal of eScience Librarianship*, 1, 2012.
12. P Arzberger, P Schroeder, A Beaulieu, G Bowker, K Casey, L Laaksonen, D Moorman, P Uhlir, and P Wouters. Promoting access to public research data for scientific, economic, and social development. *Data Science Journal*, 3:135–152, 2004.
13. Michael Ashburner, Catherine A Ball, Judith A Blake, David Botstein, Heather Butler, J Michael Cherry, Allan P Davis, Kara Dolinski, Selina S Dwight, Janan T Eppig, et al. Gene ontology: tool for the unification of biology. *Nature genetics*, 25(1):25–29, 2000.
14. TK Attwood, DB Kell, P McDermott, J Marsh, SR Pettifer, and D Thorne. Utopia documents: linking scholarly literature with research data. *Bioinformatics (Oxford, England)*, 26(18):i568–74, 09 2010.
15. SA Azer. Evaluation of gastroenterology and hepatology articles on wikipedia: are they suitable as learning resources for medical students? *European journal of gastroenterology & hepatology*, 26(2):155–163, 02 2014.
16. Amos Bairoch, Rolf Apweiler, Cathy H. Wu, Winona C. Barker, Brigitte Boeckmann, Serenella Ferro, Elisabeth Gasteiger, Hongzhan Huang, Rodrigo Lopez, Michele Magrane, Maria J. Martin, Darren A. Natale, Claire O'Donovan, Nicole Redaschi, and Lai-Su L. Yeh. The universal protein resource (uniprot). *Nucleic Acids Research*, 33(suppl 1):D154–D159, 2005.
17. Alex Ball and Monica Duke. How to cite datasets and link to publications. Dcc how-to guides, Edinburgh: Digital Curation Centre, 2012.

18. Alessia Bardi, Paolo Manghi, and Franco Zoppi. Aggregative data infrastructures for the cultural heritage. In JuanManuel Dodero, Manuel Palomo-Duarte, and Pythagoras Karampiperis, editors, *Metadata and Semantics Research*, Communications in Computer and Information Science, pages 239–251. Springer Berlin Heidelberg, 2012.
19. Alessia Bardi, Paolo Manghi, and Franco Zoppi. Coping with interoperability and sustainability in cultural heritage aggregative data infrastructures. *Int. J. Metadata Semant. Ontologies*, 9(2):138–154, April 2014.
20. Franz Barjak. The role of the internet in informal scholarly communication. *Journal of the American Society for Information Science and Technology*, 57(10):1350–1367, 2006.
21. Sean Bechhofer, Iain Buchan, David De Roure, Paolo Missier, John Ainsworth, Jiten Bhagat, Philip Couch, Don Cruickshank, Mark Delderfield, Ian Dunlop, Matthew Gamble, Danius Michaelides, Stuart Owen, David Newman, Shoaib Sufi, and Carole Goble. Why linked data is not enough for scientists. *Future Generation Computer Systems*, 29(2):599 – 611, 2013. <ce:title>Special section: Recent advances in e-Science</ce:title>.
22. Sean Bechhofer, David De Roure, Matthew Gamble, Carole Goble, and Iain Buchan. Research objects: Towards exchange and reuse of digital knowledge. *The Future of the Web for Collaborative Science*, 2010.
23. David Beckett. Rdf 1.1 n-triples a line-based syntax for an rdf graph. Technical report, W3C Recommendation, February 2014.
24. Tim Berners-Lee. Linked data. `https://www.w3.org/DesignIssues/LinkedData.html`, 07 2006. Last visited: February 2016.
25. Chris Bizer and Richard Cyganiak. Rdf 1.1 trig. Technical report, W3C Recommendation, February 2014.
26. Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, MarMar 2009.
27. Theo Bloom. Data Access for the Open Access Literature: PLOS's Data Policy. `http://www.plos.org/data-access-for-the-open-access-literature-ploss-data-policy/`, 12 2013. Last visited: 2014-03-10.
28. Christine L. Borgman. What are digital librires? competing visions. *Information Processing and Management*, 35(3):227 – 243, 1999.
29. Christine L. Borgman. The conundrum of sharing research data. *Journal of the American Society for Information Science and Technology*, 63(6):1059–1078, 2012.
30. Christine Borowski. Enough is enough. *The Journal of Experimental Medicine*, 208(7):1337, 2011.
31. Philip E. Bourne, Tim Clark, Robert Dale, Anita De Waard, Ivan Herman, Eduard H Hovy, and David Shotton. Force11 White Paper: Improving The Future of Research Communications and e-Scholarship. Technical report, Force11, 2012.
32. Grant R Brammer, Ralph W Crosby, Suzanne J Matthews, and Tiffani L Williams. Paper mâché: Creating dynamic reproducible science. *Procedia Computer Science*, 4:658–667, 2011.
33. J. Brase. Datacite - a global registration agency for research data. In *Cooperation and Promotion of Information Resources in Science and Technology, 2009. COINFO '09. Fourth International Conference on*, pages 257–261, Nov 2009.
34. Alvis Brazma, Helen Parkinson, Ugis Sarkans, Mohammadreza Shojatalab, Jaak Vilo, Niran Abeygunawardena, Ele Holloway, Misha Kapushesky, Patrick Kemmeren, Gonzalo Garcia Lara, et al. Arrayexpress—a public repository for microarray gene expression data at the ebi. *Nucleic acids research*, 31(1):68–71, 2003.
35. Leen Breure, Hans Voorbij, and Maarten Hoogerwerf. Rich internet publications: Show what you tell. *Journal of Digital Information*, 12(1), 2011.
36. E. Brewer. Cap twelve years later: How the "rules" have changed. *Computer*, 45(2):23–29, Feb 2012.

37. Budapest Open Access Initiative. Budapest Open Access declaration. `http://www.budapestopenaccessinitiative.org`, February 2002. Last visited: October 2015.
38. Sarah Callaghan, Steve Donegan, Sam Pepler, Mark Thorley, Nathan Cunningham, Peter Kirsch, Linda Ault, Patrick Bell, Rod Bowie, Adam Leadbetter, et al. Making data a first class scientific output: Data citation and publication by nercâ s environmental data centres. *International Journal of Digital Curation*, 7(1):107–113, 2012.
39. Leonardo Candela, Donatella Castelli, Nicola Ferro, G Koutrika, C Meghini, P Pagano, Seamus Ross, D Soergel, M Agosti, and M Dobreva. *The DELOS Digital Library Reference model. Foundations for digital Libraries (Version 0.98)*. ISTI-CNR at Gruppo ALI, 2008.
40. Leonardo Candela, Donatella Castelli, Paolo Manghi, and Alice Tani. Data journals: A survey. *Journal of the Association for Information Science and Technology*, 66(9):1747–1762, 2015.
41. Leonardo Candela, Donatella Castelli, and Pasquale Pagano. On-demand virtual research environments and the changing roles of librarians. *Library Hi Tech*, 27(2):239–251, 2009.
42. Leonardo Candela, Donatella Castelli, Pasquale Pagano, and Manuele Simi. From heterogeneous information spaces to virtual documents. In *Digital Libraries: Implementing Strategies and Sharing Experiences*, pages 11–22. Springer, 2005.
43. Leonardo Candela, Donatella Castelli, Pasquale Pagano, and Manuele Simi. Opendlib: A digital library service system. *Handbook of Research on Digital Libraries: Design, Development, and Impact*, 2009.
44. Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys (CSUR)*, 17(4):471–523, 1985.
45. Carl Lagoze and Herbert Van de Sompel. The making of the open archives initiative protocol for metadata harvesting. *Library Hi Tech*, 21(2):118 – 128, 2003.
46. Annamaria Carusi and Torsten Reimer. Virtual Research Environment Collaborative Landscape Study: A JISC funded project. Technical report, JISC, January 2010.
47. Donatella Castelli, Paolo Manghi, and Costantino Thanos. A vision towards scientific communication infrastructures. *International Journal on Digital Libraries*, pages 1–15, 2013.
48. Donald D. Chamberlin. Relational data-base management systems. *ACM Comput. Surv.*, 8(1):43–66, March 1976.
49. Vishwas Chavan and Lyubomir Penev. The data paper: a mechanism to incentivize data publishing in biodiversity science. *BMC Bioinformatics*, 12(Suppl 15):S2, 2011.
50. Kwok Cheung, Jane Hunter, Anna Lashtabeg, and John Drennan. Scope: a scientific compound object publishing and editing system. *International Journal of Digital Curation*, 3(2):4–18, 2008.
51. E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.
52. UniProt Consortium et al. The universal protein resource (uniprot). *Nucleic acids research*, 36(suppl 1):D190–D195, 2008.
53. R. Cook, W. Michener, D. Vieglais, A. Budden, and R. Koskela. Dataone: A distributed environmental and earth science data network supporting the full data life cycle. In A. Abbasi and N. Giesen, editors, *EGU General Assembly Conference Abstracts*, EGU General Assembly Conference Abstracts, 2012.
54. Richard Cyganiak, David Wood, and Markus Lanthaler. Rdf 1.1 concepts and abstract syntax. Technical report, W3C Recommendation, February 2014.
55. Michael Diepenbroek, Hannes Grobe, Manfred Reinke, Uwe Schindler, Reiner Schlitzer, Rainer Sieger, and Gerold Wefer. Pangaea—an information system for environmental sciences. *Computers & Geosciences*, 28(10):1201–1210, 2002.
56. Dublin Core Metadata Initiative. Dublin Core. `http://dublincore.org/`.
57. Dominic Farace, Jerry Frantzen, Christiane Stock, Laurents Sesink, and Debbie Rabina. Linking full-text grey literature to underlying research and post-publication data: An enhanced publications project 2011-2012. *Grey Journal (TGJ)*, 8(3), 2012.

58. Scott Federhen. The ncbi taxonomy database. *Nucleic acids research*, 40(D1):D136–D143, 2012.
59. J Lynn Fink and Philip E Bourne. Reinventing scholarly communication for the electronic age. *CTWatch Quarterly*, 3(3):26–31, 2007.
60. A. Fox and E.A. Brewer. Harvest, yield, and scalable tolerant systems. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, pages 174–178, 1999.
61. L Garcia Castro, C McLaughlin, and A Garcia. Biotea: Rdfizing pubmed central in support for the paper as an interface to the web of data. *Journal of Biomedical Semantics*, 4(Suppl 1):S5, 2013.
62. Eugene Garfield. What is the primordial reference for the phrase 'publish or perish'? *The Scientist*, 10(12):11, 1996.
63. Anna Gerber and Jane Hunter. Authoring, editing and visualizing compound objects for literary scholarship. *Journal of Digital Information*, 11(1), 2010.
64. Jim Giles. Internet encyclopaedias go head to head. *Nature*, 438(7070):900–901, 2005.
65. Pieter Van Gorp and Steffen Mazanek. Share: a web portal for creating and sharing executable research papers. *Procedia CS*, 4:589–597, 2011.
66. Burton Grad and Thomas J. Bergin. Guest editors' introduction: History of database management systems. *IEEE Annals of the History of Computing*, 31(4):3–5, 2009.
67. Jim Gray. Jim gray on escience: a transformed scientific method. *The fourth paradigm: Data-intensive scientific discovery*, pages 19–31, 2009.
68. W3C SPARQL Working Group et al. Sparql 1.1 overview. w3c recommendation 21 march 2013. Technical report, W3C - SPARQL Working Group, March 2013.
69. W3C Working Group. Rdfa 1.1 primer - third edition. rich structured data markup for web documents. Technical report, W3C, March 2015.
70. Jean-Claude Guédon. *In Oldenburg's long shadow: Librarians, research scientists, publishers, and the control of scientific publishing*. Association of Research Libraries, 2001.
71. Jean-Claude Guédon. Locating the information society within civil society: The case of scientific and scholarly publications. *Communicating in the information society*, pages 165–194, 2003.
72. CA Haigh. Wikipedia as an evidence source for nursing and healthcare students. *Nurse education today*, 31(2):135–139, 02 2011.
73. T. Haigh. How data got its base: Information storage software in the 1950s and 1960s. *Annals of the History of Computing, IEEE*, 31(4):6–25, Oct 2009.
74. Diane Harley. Scholarly communication: Cultural contexts, evolving models. *Science*, 342(6154):80–82, 2013.
75. Bernhard Haslhofer and Wolfgang Klas. A survey of techniques for achieving metadata interoperability. *ACM Comput. Surv.*, 42(2):7:1–7:37, March 2010.
76. Tom Heath and Christian Bizer. *Linked data: Evolving the web into a global data space (1st edition)*, volume 1 of *Synthesis lectures on the semantic web: theory and technology*. Morgan & Claypool, 2011.
77. Steve M Hitchcock, Les A Carr, and Wendy Hall. A survey of stm online journals 1990-95: the calm before the storm. *Directory of Electronic Journals, Newsletters and Academic Discussion Lists*, pages 7–32, 1996.
78. Marilu A. Hoeppner. Ncbi bookshelf: books and documents in life sciences and health care. *Nucleic Acids Research*, 41(D1):D1251–D1260, 2013.
79. Maarten Hoogerwerf, Mathias Lösch, Jochen Schirrwagen, Sarah Callaghan, Paolo Manghi, Katerina Iatropoulou, Dimitra Keramida, and Najla Rettberg. Linking data and publications: Towards a cross-disciplinary approach. *International Journal of Digital Curation*, 8(1):244–254, 2013.
80. Xing Huang, Xianghua Ding, Charlotte P. Lee, Tun Lu, and Ning Gu. Meanings and boundaries of scientific software sharing. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, pages 423–434, New York, NY, USA, 2013. ACM.

81. Jane Hunter. Scientific models: a user-oriented approach to the integration of scientific data and digital libraries. In *VALA2006*, pages 1–16, 2006.
82. Jane Hunter. Scientific publication packages–a selective approach to the communication and archival of scientific output. *International Journal of Digital Curation*, 1(1):33–52, 2008.
83. Jane Hunter and Kwok Cheung. Provenance explorer-a graphical interface for constructing scientific publication packages from provenance trails. *International Journal on Digital Libraries*, 7(1-2):99–107, 2007.
84. Scott Jackson. *Architecting resilient systems: Accident avoidance and survival and recovery from disruptions*, volume 66. John Wiley & Sons, 2009.
85. Nicholas W Jankowski, Andrea Scharnhorst, Z Tatum, and C Tatum. Enhancing scholarly publications: Developing hybrid monographs in the humanities and social sciences. *Scholarly and Research Communication*, 4, 2013.
86. Arif E. Jinha. Article 50 million: an estimate of the number of scholarly articles in existence. *Learned Publishing*, 23(3):258–263, 2010-07-01T00:00:00.
87. Åke Johansson and Mats Ola Ottosson. A national current research information system for sweden. In *11th International Conference on Current Research Information Systems (CRIS 2012: Prague, Czech Republic, June 6th-9th, 2012)*, pages 67–71. Agentura Action M, euroCRIS, 2012.
88. Lucas N. Joppa, Greg McInerny, Richard Harper, Lara Salido, Kenji Takeda, Kenton O'Hara, David Gavaghan, and Stephen Emmott. Troubling trends in scientific software use. *Science*, 340(6134):814–815, 2013.
89. Brigitte Jörg. Cerif: The common european research information format model. *Data Science Journal*, 9:CRIS24–CRIS31, 2010.
90. Patricia Kahn and David Hazledine. Nar's new requirement for data submission to the embl data library: information for authors. *Nucleic Acids Research*, 16(10):I, 1988.
91. Gary King. An introduction to the dataverse network as an infrastructure for data sharing. *Sociological Methods & Research*, 36(2):173–199, 2007.
92. Kenneth M King. The future of scholarly communication. In Eugene P. Trani, editor, *The Future of Academic Library*, number 188-189 in Occasional Papers, page 63. University of Illinois, Graduate School of Library and Information Science, January 1991.
93. Joost G Kircz. New practices for electronic publishing 2: New forms of the scientific paper. *Learned Publishing*, 15(1):27–32, 2002.
94. John A Kunze, Patricia Cruse, Rachael Hu, Stephen Abrams, Kirk Hastings, Catherine Mitchell, and Lisa R Schiff. Practices, trends, and recommendations in technical appendix usage for selected data-intensive disciplines. `http://www.escholarship.org/uc/item/9jw4964t`, 2011.
95. Carl Lagoze, Herbert Van de Sompel, Michael L. Nelson, Simeon Warner, Robert Sanderson, and Pete Johnston. Object re-use & exchange: A resource-centric approach. *CoRR*, abs/0804.2273, 2008.
96. Carl Lagoze and Herbert Van de Sompel. Ore specifications and user guides. `http://www.openarchives.org/ore/1.0`, 08 2014. Last visited: February 2016.
97. Carl Lagoze and Herbert Van de Sompel. The Open Archives Initiative Protocol for Metadata Harvesting. Protocol Version 2.0 of 2002-06-14. `http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm`, 01 2015. Last visited: February 2016.
98. Carl Lagoze, Herbert Van de Sompel, Michael Nelson, Simeon Warner, Robert Sanderson, and Pete Johnston. A web-based resource model for scholarship 2.0: object reuse & exchange. *Concurrency and Computation: Practice and Experience*, 24(18):2221–2240, 2012.
99. Frederick W Lancaster. The evolution of electronic publishing. *Library trends*, 43(4):518–527, 1995.
100. Jung-Won Lee, Ki Ho Lee, and Won Kim. Preparations for semantics-based xml mining. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 345–352. IEEE, 2001.

101. Rasko Leinonen, Ruth Akhtar, Ewan Birney, Lawrence Bower, Ana Cerdeno-Tárraga, Ying Cheng, Iain Cleland, Nadeem Faruque, Neil Goodgame, Richard Gibson, Gemma Hoad, Mikyung Jang, Nima Pakseresht, Sheila Plaister, Rajesh Radhakrishnan, Kethi Reddy, Siamak Sobhany, Petra Ten Hoopen, Robert Vaughan, Vadim Zalunin, and Guy Cochrane. The european nucleotide archive. *Nucleic Acids Research*, 39(suppl 1):D28–D31, 2011.
102. James Lewis and Martin Fowler. Microservices. `http://martinfowler.com/articles/microservices.html`, March 2014. Last visited: February 2016.
103. Tim T.Y. Lin. Scholarly Markdown. `http://scholarlymarkdown.com`, 2015. Last visited: December 2015.
104. Philip Lord and Alison Macdonald. e-science curation report. data curation for e-science in the uk: an audit to establish requirements for future curation and provision. Technical report, The JISC Committee for the Support of Research (JCSR), 2003.
105. Norbert Lossau and Dale Peters. Driver: Building a sustainable infrastructure of european scientific repositories. *LIBER Quarterly*, 18(3/4), 2008.
106. Paolo Manghi, Michele Artini, Claudio Atzori, Alessia Bardi, Andrea Mannocci, Sandro La Bruzzo, Leonardo Candela, Donatella Castelli, Pasquale Pagano, Miguel-Angel Sicilia, and Miguel-Angel Sicilia. The D-NET software toolkit: A framework for the realization, maintenance, and operation of aggregative infrastructures. *Program*, 48(4), 2014.
107. Paolo Manghi, Lukasz Bolikowski, Natalia Manold, Jochen Schirrwagen, and Tim Smith. Openaireplus: the european scholarly communication data infrastructure. *D-Lib Magazine*, 18(9):1, 2012.
108. Paolo Manghi, Marko Mikulicic, Leonardo Candela, Michele Artini, and Alessia Bardi. General-purpose digital library content laboratory systems. In Mounia Lalmas, Joemon M. Jose, Andreas Rauber, Fabrizio Sebastiani, and Ingo Frommholz, editors, *Research and Advanced Technology for Digital Libraries. 14th European Conference, ECDL 2010, Glasgow, UK, September 6-10, 2010. Proceedings*, volume 6273 of *Lecture Notes in Computer Science*, pages 14–21, Glasgow, UK, September 2010. Springer.
109. Robert Cecil Martin. The Principles of OOD. `http://www.butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod`. Last visited: February 2016.
110. Robert Cecil Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.
111. John Maunsell. Announcement regarding supplemental material. *The Journal of Neuroscience*, 30(32):10599–10600, 2010.
112. Johanna R McEntyre, Sophia Ananiadou, Stephen Andrews, William J Black, Richard Boulderstone, Paula Buttery, David Chaplin, Sandeepreddy Chevuru, Norman Cobley, Lee-Ann Coleman, et al. Ukpmc: a full text article resource for the life sciences. *Nucleic acids research*, 39(suppl 1):D58–D65, 2011.
113. Gerry McKiernan. E is for everything. *The Serials Librarian*, 41(3-4):293–321, 2002.
114. Barend Mons. Which gene did you mean? *BMC Bioinformatics*, 6(1):142, 2005.
115. Hailey Mooney and Mark P Newton. The anatomy of a data citation: Discovery, reuse, and credit. *Journal of Librarianship & Scholarly Communication*, 1(1):6, 2012.
116. D. Namiot and M. Sneps-Sneppe. On micro-services architecture. *International Journal of Open Information Technologies*, 9, 2014.
117. Richi Nayak, Rebecca Witt, and Anton Tonev. Data mining and xml documents. In *International Conference on Internet Computing, IC'2002*, pages 660–666, Las Vegas, Nevada, 2002. CSREA Press.
118. Paul Newman and Peter Corke. Editorial: Data papers — peer reviewed publication of high quality data sets. *The International Journal of Robotics Research*, 28(5):587, 2009.
119. Sam Newman. *Building Microservices*. O'Reilly Media, Inc., 2015.
120. David Nicholas, Paul Huntington, Hamid R. Jamali, and Tom Dobrowolski. Characterising and evaluating information seeking behaviour in a digital environment: Spotlight on the 'bouncer'. *Information Processing and Management*, 43(4):1085 – 1102, 2007.

121. David Nicholas, Paul Huntington, Peter Williams, and Tom Dobrowolski. Re-appraising information seeking behaviour in a digital environment: bouncers, checkers, returnees and the like. *Journal of Documentation*, 60(1):24–43, 2004.
122. Piotr Nowakowski, Eryk Ciepiela, Daniel Harężlak, Joanna Kocot, Marek Kasztelnik, Tomasz Bartyński, Jan Meizner, Grzegorz Dyk, and Maciej Malawski. The collage authoring environment. *Procedia Computer Science*, 4(0):608 – 617, 2011. <ce:title>Proceedings of the International Conference on Computational Science, {ICCS} 2011</ce:title>.
123. Stephanie Oeben and Sarah Huggett. The black eagle soars: Germany's bibliometric trends 2004-2013. *Research Trends*, 38, September 2014.
124. Ann Okerson. The electronic journal: What, whence, and when? *Public Access-Computer Systems Review*, 2(1), 1991.
125. Gloria Origgi and Giovanni B. Ramello. Current dynamics of scholarly publishing. *Evaluation Review*, 2015.
126. J.S. Mackenzie Owen. *The scientific article in the age of digitization*. Ph.d. dissertation, University of Amsterdam, Faculty of Humanities, 2005.
127. MA Parsons and PA Fox. Is data publication the right metaphor? *Data Science Journal*, 12:WDS32–WDS46, 2013.
128. Silvio Peroni. Research Articles in Simplified HTML (RASH). `https://github.com/essepuntato/rash`. Last visited: January 2016.
129. Quan Pham, Tanu Malik, Ian Foster, Roberto Di Lauro, and Raffaele Montella. Sole: linking research papers with science objects. In *Provenance and Annotation of Data and Processes*, pages 203–208. Springer, 2012.
130. Benjamin C Pierce. *Types and programming languages*. MIT press, 2002.
131. Heather Piwowar. Supplementary materials is a stopgap for data archiving. `http://researchremix.wordpress.com/2010/08/13/supplementary-materials-is-a-stopgap-for-data-archiving`, August 2010. Last visited: 2014-02-23.
132. Heather A Piwowar, Todd J. Vision, and Michael C. Whitlock. Data archiving is a good investment. *Nature*, 473:285 – 285, 5/2011 2011.
133. Andrew Plume and Daphne van Weijen. Publish or perish? the rise of the fractional author.... *Research Trends*, 38, September 2014.
134. Jason Priem and Bradely Hemminger. Scientometrics 2.0: New metrics of scholarly impact on the social web. *First Monday*, 15(7), 2010.
135. Nicola J Reavley, Andrew J Mackinnon, Amy J Morgan, Mario Alvarez-Jimenez, Sarah E Hetrick, Eoin Killackey, Barnaby Nelson, Rosemary Purcell, Marie BH Yap, and Anthony F Jorm. Quality of information sources about mental disorders: a comparison of wikipedia with centrally controlled web and printed sources. *Psychological medicine*, 42(08):1753–1762, 2012.
136. Dietrich Rebholz-Schuhmann, Miguel Arregui, Sylvain Gaudan, Harald Kirsch, and Antonio Jimeno. Text processing through web services: calling whatizit. *Bioinformatics*, 24(2):296–298, 2008.
137. Susan Reilly, Wouter Schallier, Sabine Schrimpf, Eefke Smit, and Max Wilkinson. Report on integration of data and publications. Technical report, Opportunities for Data Exchange (ODE), 2011.
138. Allen H. Renear and Carole L. Palmer. Strategic reading, ontologies, and the future of scientific publishing. *Science*, 325(5942):828–832, 2009.
139. Julian Richards, Stuart Jeffrey, Stewart Waller, Fabio Ciravegna, Sam Chapman, and Ziqi Zhang. The archaeology data service and the archaeotools project: faceted classification and natural language processing. *Archaeology 2.0: New Approaches to Communication and Collaboration*, pages 31–56, 2011.
140. Chris Richardson. Microservices: Decomposing applications for deployability and scalability. `http://www.infoq.com/articles/microservices-intro`, May 2014. Last visited: February 2016.

141. Richard J Roberts. Pubmed central: The genbank of the published literature. *Proceedings of the National Academy of Sciences*, 98(2):381–382, 2001.
142. David De Roure, Carole Goble, and Robert Stevens. The design and realisation of the virtual research environment for social sharing of workflows. *Future Generation Computer Systems*, 25(5):561 – 567, 2009.
143. David Shotton, Katie Portwin, Graham Klyne, and Alistair Miles. Adventures in semantic publishing: Exemplar semantic enhancements of a research article. *PLoS Comput Biol*, 5(4):e1000361, 04 2009.
144. J. Siciarek and Bogdan Wiszniewski. Ioda - an interactive open document architecture. *Procedia CS*, pages 668–677, 2011.
145. Manu Sporny, Dave Longley, gregg Kellogg, Markus Lanthaler, and Niklas Lindstrom. Json-ld 1.0 a json-based serialization for linked data. Technical report, W3C Recommendation, January 2014.
146. Joan Starr and Angela Gastl. iscitedby: A metadata scheme for datacite. *D-Lib Magazine*, 17(1):9, 2011.
147. Niels Stern, Jean-Claude Guédon, and Thomas Jensen. Crystals of knowledge production. an intercontinental conversation about open science and the humanities. *Nordic Perspectives on Open Science*, 1(0):1–24, 2015.
148. Stefan Svallfors. National differences in national identities? an introduction to the international social survey programme. *Journal of Ethnic and Migration Studies*, 22(1):127–134, 1996.
149. Carol Tenopir, Suzie Allard, Kimberly Douglass, Arsev Umur Aydinoglu, Lei Wu, Eleanor Read, Maribeth Manoff, and Mike Frame. Data sharing by scientists: Practices and perceptions. *PLoS ONE*, 6:e21101, 6/2011 2011.
150. Carol Tenopir, Donald W. King, Sheri Edwards, and Lei Wu. Electronic journals and changes in scholarly article seeking and reading patterns. *Aslib Proceedings*, 61(1):5–32, 2009.
151. Costantino Thanos. The future of digital scholarship. *Procedia Computer Science*, 38:22 – 27, 2014. 10th Italian Research Conference on Digital Libraries, {IRCDL} 2014.
152. The Library of Congress. The Contextual Query Language. `https://www.loc.gov/standards/sru/cql`, August 2013. Last visited: February 2016.
153. Johannes Thones. Microservices. *Software, IEEE*, 32(1):116–116, Jan 2015.
154. Andrew Treloar. Rethinking the library's role in publishing. *Learned Publishing*, 12(1):25–31, 1999-01-01.
155. Henk van den Heuvel, René van Horik, Stef Scagliola, Eric Sanders, and Paula Witkamp. The veterantapes: Research corpus, fragment processing tool, and enhanced publications for the e-humanities. In *LREC*, 2010.
156. Hollie C White, Sarah Carrier, Abbey Thompson, Jane Greenberg, and Ryan Scherle. The dryad data repository: A singapore framework metadata architecture in a dspace environment. *Universitätsverlag Göttingen*, page 157, 2008.
157. Claus O. Wilke. Supplementary materials need the right format. *Nature*, 430(6997):291–291, 07 2004.
158. Benjamin Wootton. Microservices - Not A Free Lunch! `http://highscalability.com/blog/2014/4/8/microservices-not-a-free-lunch.html`, April 2014. Last visited: February 2016.
159. Renze Woutersen-Windhouwer Brandsma, Peter Verhaar, Arjan Hogenaar, Maarten Hoogerwerf, Paul Doorenbosch, EugËne D¸rr, Jens Ludwig, Birgit Schmidt, and Barbara Sierman. *Enhanced Publications: Linking Publications and Research Data in Digital Repositories*. SURF / E-Driver, 2009.
160. Karen Yook, Todd W Harris, Tamberlyn Bieri, Abigail Cabunoc, Juancarlos Chan, Wen J Chen, Paul Davis, Norie De La Cruz, Adrian Duong, Ruihua Fang, et al. Wormbase 2012: more genomes, more data, new website. *Nucleic acids research*, 40(D1):D735–D741, 2012.