D5.1v2
Design of the Liquid Publications Integrated Platform

Maintainer/Editor-in-chief | Aliaksandr Birukou
--- | ---
Core authors | Marcos Baez, Aliaksandr Birukou, Ismel Brito, Fabio Casati, Ronald Chenu, Muhammad Imran, Juan Jose Jara, Jordi Madrenas Ciurana, Maurizio Marchese, Alejandro Mussi, Cristhian Parra, Mario Passamani, Azzurra Ragone, Joseph Rushton Wakeling
Maintainers/Editors | Marcos Baez
Reviewers | Ronald Chenu, Nardine Osman, Judith Simon
LiquidPub research group leaders | Fabio Casati, Roberto Casati, Marlon Dumas, Ralf Gerstner, Fausto Giunchiglia, Maurizio Marchese, Gloria Origgi, Alessandro Rossi, Carles Sierra, Yi-Cheng Zhang
LiquidPub project leader | Fabio Casati

<table>
<thead>
<tr>
<th>Grant agreement no.</th>
<th>213360</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project acronym</td>
<td>LiquidPublication</td>
</tr>
<tr>
<td>Version</td>
<td>v2.0</td>
</tr>
<tr>
<td>Date</td>
<td>June 15, 2010</td>
</tr>
<tr>
<td>State</td>
<td>Solid</td>
</tr>
<tr>
<td>Distribution</td>
<td>Public</td>
</tr>
</tbody>
</table>
Disclaimer

The information in this document is subject to change without notice. Company or product names mentioned in this document may be trademarks or registered trademarks of their respective companies.

This document is part of a research project funded by the European Community as project number 213360 acronym LiquidPublication under THEME 3: FP7-ICT-2007-C FET OPEN. The full list of participants is available at http://project.liquidpub.org/partners-and-contributors/liquidpub-teams.

Abstract

This document presents the three use cases identified for the LiquidPub project and the design of the architecture of the Liquid Publication integrated platform. This new version not only introduces advances in the development of the platform but also a more mature understanding of the requirements and possibilities, explored in the three use cases covering the aspects of knowledge creation, dissemination and evaluation.

Keyword list: design, architecture, platform, liquid journals, liquid books, liquid conferences
Executive Summary

The goals of LiquidPub are realized through concepts and models which are supported by an IT solution (the LiquidPub platform) and an ecosystem of services. This document describes the overall platform focusing in particular on the three ways in which the LiquidPub dissemination concepts can be exposed to the scientific community: the liquid counterparts of journals, conferences and books. This document represents a major change with respect to the first version as the work (and this document) is now driven by these three main dissemination models.

In a nutshell, the way in which the LiquidPub platform (Figure 1) supports the LP vision is the following:

- Three pillars embody the LiquidPub principles in the three main ways scientific dissemination is achieved today: journals, conferences and books. These are the metaphors through which we achieve the LP objectives with different dissemination paradigms.

- A SKO knowledge repository and semantic knowledge bus integrates information across (and allows communication among) the different pillars, and also acts as integrated repository of liquid knowledge.

- Reputation modules (OpinioNet and ResEval) process the information in the SKO repository to compute metrics that (i) reflect the collective opinions of scientists over scientific contributions, (ii) go beyond metrics with well-known flaws such as citation counts, and (iii) expose metrics that encourage behaviors that are good for science.

- Tools for workflows and lifecycle management (LPMaSys and Gelee) support the modeling and definition of the various dissemination processes and lifecycles that can be associated with the liquid artifacts handled by the modules and the knowledge bus.

- Infrastructure services (Karaku) connect LP with existing sources of information and in general with the non-LP world. An example of this consists in exposing applications such as Google Scholar and Google docs via an API to the LP platform so that information from them can be integrated into the SKO repository or the pillars, or used by the reputation tools.

In this executive summary we briefly summarize the features of the use cases described in this deliverable, as well as the architectural philosophy we followed.

Liquid Journals. We have identified a model and developed a (prototype) tool for liquid journals, that embodies many of the principles discussed above and specifically i) it supports efficient dissemination of multi-faceted knowledge; ii) it facilitates knowledge search and consumption, that is, the navigation through knowledge (thanks to the linking done by the reader, transforming readers in providers of knowledge); iii) it supports an innovative form of knowledge evaluation, which can be complementary or alternative to peer review, that leverages the community by capturing what each of us does implicitly every day in terms of selecting and sharing papers and using this information as source of evaluation. We have also understood (and modeled) that contextual
knowledge about scientific artifacts (relationships among artifacts) can be subjective and we need to allow readers to specify this. We see this as a fundamental shift in the way knowledge is created and consumed. The model has already received great interest by other information providers and publishers/societies.

**Liquid Conferences.** We have developed a model and implemented a platform for LiquidConferences in complete concordance with the LP philosophy. As a SKO itself, a liquid conference is constituted by a set of contents, authors, panelists, reviewers, readers (registered users), as well as one or more administrators; each of the roles being associated with a set of rights. Each conference is organized as a standard conference on invitation only. Invited speakers are alerted automatically, as they are “created” as authors in the platform and invited to submit their SKO. Then, the SKO will remain open to discussion for a certain amount of time (usually two weeks) and then archived and available for reading only. The temporal dynamics allow for an efficacious interaction among participants, keeping both the lively aspect of the face-to-face events and the standards of written scientific papers. Further, we allow for a modular organization of papers as well as sub-atomic credit attribution to parts of papers. The first liquid conference on Predictions will run on our platform this summer.

**Liquid Books.** We have identified a model for publishing books that is evolutionary, collaborative, and multi-facet. The model goes towards LP objectives in that it allows to more easily build on knowledge of others and to more efficiently release new and updated content. It aims at reducing the barrier that discourage authors from publishing or updating their book and it allows to have books that are very tailored to the needs of the readers. We have identified the main issues we need to address to realize the model, which are related to contracts and to defining processes for credit attribution. The model has already gained the interest of societies and publishers, even if it is not sufficiently mature for being enacted.

**LiquidPub Ecosystem.** We have understood that the nature of the LP platform is that of supporting pillars that implement the LP vision (Liquid Journals, Conferences, Books) and a knowledge bus that integrates information across (and allows communication among) the different pillars, and that also act as integrated repository of liquid knowledge. In parallel to this initial architecture, we envision different kinds of extensions: some will be directly extending the pillars, but others can build directly on top of the knowledge bus and of other knowledge services. In terms of implementation strategy, we progressively understood that the various LP pillars that deliver (along with the other LiquidPub components) functionality to users and ultimately deliver the LP vision *will evolve over time* also during the life of the project, especially as they become adopted and we learn how people use them. We feel this evolution is natural as the concepts are novel. Therefore, initially each pillar should have its own model and API, and possibly initially even its own physical data repository, which is required to support their fast evolution. However each of them needs to be integrated, so that the pillars can exchange knowledge. Hence, i) concepts and API of the pillars need to be mapped to concepts and API of a shared model/bus, and ii) over time the shared model should evolve to accommodate new requirements from the pillars, at the appropriate level of abstraction. This is managed by integration via adapters with the knowledge bus.
## Contents

1 **Introduction** 1

2 **Use cases (three pillars)** 5

3 **SKO Platform and Knowledge Bus** 36

4 **Reputation services** 43

5 **Process lifecycle services** 49

6 **Infrastructure services** 56

---

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Use cases (three pillars)</td>
<td>5</td>
</tr>
<tr>
<td>Liquid Journals</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>5</td>
</tr>
<tr>
<td>2.1.2 Scientific Resources for Liquid Journals</td>
<td>7</td>
</tr>
<tr>
<td>2.1.3 Liquid Journal Model</td>
<td>10</td>
</tr>
<tr>
<td>2.1.4 Creating and Filling a Liquid Journal</td>
<td>12</td>
</tr>
<tr>
<td>2.1.5 Usage model and evaluation metrics</td>
<td>14</td>
</tr>
<tr>
<td>2.1.6 Liquid Journal platform architecture</td>
<td>15</td>
</tr>
<tr>
<td>2.1.7 Related Work</td>
<td>17</td>
</tr>
<tr>
<td>2.1.8 Liquid Journals and LiquidPub</td>
<td>18</td>
</tr>
<tr>
<td>Liquid Conferences</td>
<td>18</td>
</tr>
<tr>
<td>2.2 Introduction</td>
<td>18</td>
</tr>
<tr>
<td>2.2.2 What is a Liquid Conference?</td>
<td>19</td>
</tr>
<tr>
<td>2.2.3 Roles</td>
<td>20</td>
</tr>
<tr>
<td>2.2.4 Modularity and multiple authorship</td>
<td>22</td>
</tr>
<tr>
<td>2.2.5 Interdisciplines: an implementation of Liquid Conferences, Beta-version</td>
<td>27</td>
</tr>
<tr>
<td>Liquid Books</td>
<td>29</td>
</tr>
<tr>
<td>2.3 Introduction to Liquid Books</td>
<td>29</td>
</tr>
<tr>
<td>2.3.2 Liquid Books for Teaching</td>
<td>30</td>
</tr>
<tr>
<td>2.3.3 Conceptual Model</td>
<td>31</td>
</tr>
<tr>
<td>2.3.4 Modeling, Business, and IT decisions</td>
<td>33</td>
</tr>
<tr>
<td>2.3.5 Related work, novelty and difference with other book models</td>
<td>34</td>
</tr>
<tr>
<td>SKO Conceptual Model</td>
<td>37</td>
</tr>
<tr>
<td>Liquid Journals Matching</td>
<td>39</td>
</tr>
<tr>
<td>Liquid Conferences Matching</td>
<td>40</td>
</tr>
<tr>
<td>Liquid Books Matching</td>
<td>41</td>
</tr>
<tr>
<td>Enabling Other Services through the SKO Knowledge Repository/Bus</td>
<td>42</td>
</tr>
<tr>
<td>OpinioNet</td>
<td>43</td>
</tr>
<tr>
<td>4.1 Implementation</td>
<td>45</td>
</tr>
<tr>
<td>ResEval</td>
<td>47</td>
</tr>
<tr>
<td>4.2 Implementation</td>
<td>49</td>
</tr>
<tr>
<td>LPMaSys</td>
<td>49</td>
</tr>
<tr>
<td>5.1 Implementation</td>
<td>51</td>
</tr>
<tr>
<td>Gelee</td>
<td>52</td>
</tr>
<tr>
<td>5.2 Implementation</td>
<td>55</td>
</tr>
</tbody>
</table>

---

June 15, 2010 213360
6.1 Updater: Crawling the Web for Scientific Resources ........................................... 57
6.2 Resman: Accessing a Web of Heterogeneous Resources ................................. 58
   6.2.1 Resource Space Management .................................................................... 59
   6.2.2 Access Management ............................................................................... 59
6.3 The Role of Adapters ...................................................................................... 60
   6.3.1 Example ................................................................................................... 61
6.4 Implementation ................................................................................................. 62

7 Testing ................................................................................................................. 62
1 Introduction

The goal of LiquidPub is to capture the lessons learned and opportunities provided by the Web and open source, agile software development to develop concepts, models, metrics, and tools for an efficient (for people), effective (for science), and sustainable (for publishers and the community) way of creating, disseminating, evaluating, and consuming scientific knowledge. This is realized through concepts and models which are supported by an IT solution (the LiquidPub platform) and an ecosystem of services. The way in which the LiquidPub platform (Figure 1) supports the LP vision is the following:

- Three *pillars* embody the LiquidPub principles in the three main ways scientific dissemination is achieved today: conferences, journals, and books. These are the metaphors through which we achieve the LP objectives with different dissemination paradigms. Within the LP platform, these are the vertical rectangles in Figure 1.

- A SKO knowledge repository and semantic *knowledge bus* integrates information across (and allows communication among) the different pillars, and also acts as an integrated repository of liquid knowledge.

- Reputation modules (OpinioNet and ResEval) process the information in the SKO repos-
ility to compute metrics that (i) reflect the collective opinions of scientists over scientific contributions, (ii) go beyond metrics with well-known flaws such as citation counts, and (iii) expose metrics that encourage behaviors that are good for science.

- Tools for workflows and lifecycle management (LPMaSys and Gelee) support the modeling and definition of the various dissemination processes and lifecycles that can be associated with the liquid artifacts handled by the modules and the knowledge bus.

- Infrastructure services (Karaku) connect LP with existing sources of information and in general with the non-LP world. An example of this consists in exposing applications such as Google Scholar and Google docs via an API to the LP platform so that information from them can be integrated into the SKO repository or the pillars, or used by the reputation tools.

In addition to these architectural components, LP develops licensing and business models for the pillars, though these aspects do not result in software modules and therefore do not appear in Figure 1 and in this deliverable. Preliminary considerations on licensing models of Liquid Books are given in [9], while business models are described in [14].

Each of the above software modules is described later in this document. We include first a broader description for the three pillars as these: (i) were not part of the original initial description of work; (ii) they are now key elements of LiquidPub and embody many of the requirements we envision for the functionality of the LiquidPub platform. Furthermore, they are only presented and discussed in details in this deliverable. We also go in some details for the infrastructure services (Karaku and its sub-component Resman), also because they are only discussed here. For the remaining components we briefly introduce them to make this document self-contained, but then discuss them in other deliverables. For instance, details of SKO model, process and lifecycle modules are described in D1.3v1 [8], D2.1v1 [9] and in D2.3v1 [10], while details of reputation modules are described in D4.1 [11] and D4.3v1 [12].

Before going into the presentation of the three pillars and the other components, in this introduction we put them into the context of the LiquidPub platform and, as we will see, of the LiquidPub ecosystem. Specifically, what we want to discuss here is the design approach we took to determine the degree of coupling among the components, that is, how loosely vs tightly the component should be integrated, and the trade-offs of each decision.

In terms of implementation strategy, while advancing with the design and implementation of the pillars, we progressively understood that:

1. The various LP pillars that deliver (along with the other LiquidPub components) functionality to users and ultimately deliver the LP vision will evolve over time also during the life of the project, especially as they become adopted and we learn how people use them. We feel this evolution is natural as the concepts are novel. As we dive deep into development and as we have early prototypes available, we better learn and understand requirements as well as how to refine, extend, and in general evolve the basic abstractions and underlying conceptual models. We cannot imagine to get things right the first time around.

2. Researchers in this area may develop other dissemination paradigms, or paradigms targeted to specific artifacts (or, we may find over time that many concepts from books, conferences and journals will merge - as we indeed already started to experience). For example,
some researchers outside LiquidPub (from UNSW Sydney) are developing a Liquid Benchmark module, while others are interested in using Liquid Journals as a way to annotate, tag, and link resources in a digital library they own, for example a library of demos (as in the Share system from TU Eindhoven). Incidentally, these two modules developed by other researchers can already interface with LiquidPub.

These observations carry two important implications. The first implication is that it would be a mistake to tightly couple among them the abstractions of each of the pillar from the very start. In fact, both from a conceptual perspective and in terms of implementation, in LP we proceeded by developing the three dissemination paradigms for the pillars. Each paradigm and supporting tool (including in particular the UI and the design of the interaction with the user) has its own requirements and abstractions. It is important that, at least initially, the abstractions and the conceptual model are tailored for each of them, because this makes it easier to provide a coherent and understandable set of concepts at the appropriate abstraction level. It also makes easier to write code and design the user interaction. For example, Liquid Conferences will have the notions of conference, of paper to be discussed, of comments, etc. Liquid Journals will have the notion of scientific resources that are the items in a journal, the notion of journal itself, the notion of issues, editors, etc. Liquid Books will have the books and editions as first class objects. Furthermore, as the tools are developed and used, and even as our thought process proceeds and new ideas and opportunities come to mind, these conceptual model and, more in general, the functionality offered by each of the three pillars will evolve. Because of the research nature of the project, binding the models and implementations tightly from the start would reduce the flexibility and make it more difficult to capture the lessons learned during development and early usage.

However, we do want to have an integrated end result because we do want each LP pillar to benefit from what the others can provide. While we believe that a certain degree of autonomy is necessary as it facilitates flexibility and evolution of ideas (and tools), it is also important to connect the pillars so that they can share knowledge and put the functionality of one at the service of the other. For example, a scientific resource in a liquid journal can become a paper in a liquid conference that is the basis for a conference/discussion session, or vice versa. Or it can be taken as part of a liquid book.

The second implication is that we need to provide the hooks for other researchers to build over our results and easily integrate their idea with LiquidPub - as we certainly do not claim to have the copyright on all ideas in the area. We envision different kind of extensions, some will be directly extending the pillars, but others can build directly on top of the knowledge bus and of other “knowledge services”, as discussed below. This is what we mean by the “ecosystem”: providing a way for other R&D effort to integrate with LiquidPub (this is graphically represented by the xyz pillar in Figure 1), leveraging what we can offer in terms of liquid knowledge but also providing information that can be collected as SKOs, measured by the LiquidPub reputation metrics, and provided as information to the three pillars.

These problems (integration and extensibility) are the LiquidPub counterpart of the well-known enterprise application integration (EAI) research and development area, where the goal is to integrate a company’s IT systems and services. EAI is needed in an enterprise because systems are developed independently (and it is important that it is so) but then they also need to be

\[http://is.tm.tue.nl/staff/pvgorp/share/\]
integrated, otherwise it is impossible to execute business processes efficiently as these invariably span many IT systems. The response to this was the development of a multi-billion dollar industry around such concepts as enterprise middleware, service bus, and the like. LiquidPub needs the same kind of bus, but for knowledge and for supporting knowledge dissemination. In the knowledge bus, abstractions related to scientific knowledge are integrated and can be transferred across pillars. Having a knowledge bus also implies having a model for liquid knowledge at a lower level of abstraction and more general in that it needs to cover the concepts of the pillars (and of the ones that will come along in the future, to the possible extent). So the challenge in the knowledge bus and its model is to be generic enough to enable integration and extension but also specific enough to facilitate the development of services for the LiquidPub ecosystem.

The SKO knowledge bus and repository also provide a hook for defining and developing one of the most important aspect of LiquidPub, which is reputation. Indeed, while a reputation module could be developed for each pillar - more tailored but with i) more development effort as it needs to be developed for each of them, and ii) restricted in visibility to what is managed by that pillar - one option LiquidPub is pursuing is to achieve integration at the knowledge bus/shared model level. This means that reputation in LP entails mapping reputation-relevant information from the pillars into the shared model and then operating on this to derive reputation metrics for people and scientific artifacts.

What this all means in terms of architecture and in terms of LP development process (development of both concepts and software) is:

- The pillars initially have their own abstraction and conceptual model. These will also be the initial requirements for a knowledge bus/shared model.
- Because these abstractions and models will evolve over time (possibly in an agile and rapid fashion, especially in an initial period) we need to have a way to, on the one hand, let these models evolve freely (this is important to be able to capture and model the concepts that prove to be important), but on the other hand to consider that these changes do impact in some way the shared SKO model, that should indeed reflect them.
- Therefore, initially each pillar should have its own model and API, and possibly initially even its own physical data repository, which is required to support the fast evolution of each pillar.
- However each of them needs to be integrated, for the reason stated above. This observation has two sides: i) concepts and API of the pillars need to be mapped to concepts and API of a shared model/bus, and ii) over time the shared model should evolve (though, realistically, more slowly than the pillars) to accommodate new requirements at the appropriate level of abstraction. In other words, in some cases new concepts in the pillar may simply correspond to new mappings, while in other cases may result in new abstractions to be pushed down to the level of the shared model so that they are made available at the platform level.

According to the same philosophy, the infrastructure services (Karaku and its sub-component Resman) can be queried from - and feed information to - the SKO repository as well as directly into the pillars.
This strong emphasis on “use cases” (which we see not only as use cases, but rather as key elements of what of LiquidPub is exposed to users) was not part of the initial description of work, and so the pillars are not described in other deliverables. We also observe that the evolution of our thinking in terms of architecture means that what we have is not really a main platform with plugins to be added (the initial thinking), but rather full-fledged software applications to be integrated, many of which even have their own UI to interact with the end users. In the following sections, we will describe the three use cases (three pillars) in Section 2 and then components of the architecture: SKO platform and knowledge bus (Section 3), reputation services (Section 4), process lifecycle services (Section 5), and infrastructure services (Section 6). We conclude the document with the description of the approach we adopt for testing (Section 7).

2 Use cases (three pillars)

We now discuss the three LP applications of journals, conferences and books. We call them pillars also to emphasize that they are not just use cases.

2.1 Liquid Journals

2.1.1 Introduction

The Web has opened a whole world of possibilities for how scientific knowledge can be created, evaluated and disseminated. We can now publish preprints in online archives (e.g., arXiv) or simply post our papers on Web pages. Furthermore, ”papers” are not the only unit of scientific dissemination. Data, comments, scientific experiments, and even blogs can now be shared and they can be considered a form of scientific contribution that can help other scientists in their work. This means that today we have a large scientific community who can make available a large, and rapidly evolving, set of scientific contributions of different kinds. This is a big opportunity for making communication and knowledge transfer more efficient and almost real-time, but results in an information overload in terms of scientific content. Although we have a sea of information at our fingertips, it is not easy to find for interesting contributions.

An additional and somewhat puzzling problem is that with so much information available we would at least hope to be able to broaden our horizons. For example, we would hope to be able to search for contributions on ”the effectiveness on peer review” in many different fields (as this problem is indeed studied in different areas). However, having this much information results in narrowing down what we read as opposed to broadening it. We experience this in everyday life: having a TiVo® or analogous digital video recorder makes a wide set of TV programs available to us so that we tend to watch what we know we like, and are less encouraged to look for new programs. The same effect has also been observed in science [22], as we tend to keep looking into the sources we are familiar with, thereby missing a plethora of potentially interesting and relevant contributions.

Today we have very few tools at our disposal to leverage the richness of information while handling the overload. When we search for contributions, we still tend to look for papers, and

2 www.tivo.com
one option is to do so by looking at collection of papers indexed by services such as DBLP or CiteSeer. This is useful but it does not come near to solving the problem: we are limited to what is indexed, we are limited to papers (and to published papers), we are narrow in the selection (e.g., these services are in computer science for the most part), and despite this narrowness we are still likely to be overloaded with the result. An alternative approach is to use a google search, but that is not tailored to finding scientific contributions. Or we can use Google Scholar, the specific search for scientific contributions offered by Google, but the result is not often as helpful as when we search the Web for other purposes.

The analogy of Web pages points out to another limitation of the current dissemination model: while on the Web we have fairly established and accepted ways to \textit{rank} resources and to \textit{navigate} them, in science this is not always the case. There are many debates – and many papers - on which scientific metrics are “good”, along with evidences that commonly adopted metrics (e.g., citation count) have many flaws. Furthermore, contributions other than papers are not typically evaluated, and the only “link” between scientific resources used in practice is that of citations.

\textit{Liquid Journals} (LJ) are a way to overcome the information overload issue in scientific publications while also allowing to connect and collaboratively evaluate all kinds of scientific resources. Their underlying principles consist i) in leveraging the very same (large) community of scientist that creates the overload problem/opportunity to collaborate in filtering and prioritizing the information, ii) in enabling a dissemination and consumption model that naturally reduces the noise portion of the information overload right at the source, iii) in having a set of metrics that mitigate the overload and encourage ”good behaviors” for science, such as early sharing and providing feedback, and iv) in computing scientific diversity and “enforcing” it when providing information.

LJs put this principles at work through concepts, methods, and ultimately tools. In a nutshell, the method is based on facilitating – and making it convenient – for scientists to share the implicit choices and observations they make every day in terms of selecting and linking knowledge. We all read papers and collect them, group them, share them via email with our colleagues or team (and therefore implicitly stating that it’s worth to spend time reading them), “mentally” annotate them or observe that they build over this other paper or experiment or datasets, etc. If we could exploit this knowledge we would have a way to assess scientific resources of all kinds, and to link them, thereby facilitating evaluation, search and navigation. The “problem” is that today we don’t bother to communicate this knowledge – or if we do, we do it verbally while talking at the coffee machine, or perhaps via email. We do not have time (or sufficient motivation) to go put this information somewhere to let the community benefit from this and probably we would not even know how to communicate all this. Hence, one of the main intuition – and challenges – we try to exploit is how to facilitate the above and making it convenient for scientists to implicitly or explicitly share this information and for a system to process it for providing the benefits as described above.

As discussed in the previous section, in the following we present the LJ model without referring to SKOs and SKO terminology. Later in this document we show how LJ concepts can be mapped into SKOs (and an adapter has also been built for this purpose). We now present the usage models and derived metrics along with a diversity-aware information prioritization model. We also present the architecture, implementation, and usage of the platform, which is available at liquidjournal.org
2.1.2 Scientific Resources for Liquid Journals

The liquid journals model builds on a model for scientific contributions, which is designed to facilitate the search for - and navigation of scientific information of interest, and is therefore essential to support the features discussed above.

LJs see scientific contributions as a structured, evolving, and multi-facet objects. Specifically, we see the space of scientific content we want to search, assess and disseminate as consisting of scientific resources, organized as set of nodes in a graph, that can be connected and annotated by authors or even readers. The reasons for connections, and hence for modeling resources as a graph, is to capture several kinds of dependencies or relationships among them (or between resources and people or other entities, as discussed next), which go beyond what is currently captured today (which is essentially limited to authorship and citation).

We see these relationships as subjective, in that each person can in principle have different opinions on how two resources are related or even on the authorship (that is, on how or how much a person contributed to the knowledge represented by a scientific resource). So, in the LJ scientific resource model, we do not assume any universal truth. Relationships (including authorship), like annotations, are defined by a person or institution that therefore claims that relation to be true. For example, I may state that I contributed to a paper. Relations can then be certified by certification agencies or endorsed by people. For example, Springer may certify that I, indeed, am author of a paper, and so that the relation is true.

In the current model we do not assume any method for ascertaining truth: we simply let users create and/or endorse relations, record them, and let the community manage and evolve them. This is analogous to what Wikipedia does and it works fine for that purpose. It is possible that this "democratic", honor approach over time does not prove to work and in that case a solution will have to be studied also depending on which specific aspect fails to be effective in the this community-managed approach.

The reason for allowing anybody to define relationships is because in this way we can leverage the power of the community to build contextual knowledge, that is, knowledge that can help annotate and relate resources above and beyond what authors would do. In other words, people generate knowledge that helps in organizing and finding scientific resources. This is sometimes called "second-order knowledge", which we believe is as important in supporting scientists' work (standing on the shoulders of giants) as first order one.

Figure 2 shows an example of resource graph, which illustrates relationships and their purpose. To illustrate these concepts, in Figure 2(a) we show the work of the LP research group on evaluation metrics and peer review. We started this line of research with in the context of a project deliverable (D3.1). This deliverable is composed of a review of the state of the art, experiments, analysis and presentation of the results. The results were delivered in two releases, D3.1v1 and D3.1v2, and we plan to produce in the near future a third version. These releases are captured by special relations that allow us to specify that a particular scientific resource is the evolution, or a new version, of a previous one. This is important as when we search for information, these

---

3 [https://dev.liquidpub.org/svn/liquidpub/final/Year1/LP_D3.1.pdf](https://dev.liquidpub.org/svn/liquidpub/final/Year1/LP_D3.1.pdf), accessible using your reviewers' credentials

213360 June 15, 2010 7
(a) Example of scientific resources connected by next-version-of relation

(b) Example of different representation for the same resource

(c) Example of other general relations

Figure 2: Relations between resources
links help us navigate to the latest version of a scientific contribution and it makes explicit the incremental nature of a result, thereby allowing reputation algorithms to factor this in when computing reputation of a scientific contribution. While working on this, we wanted to maintain the state of the art on peer review we had produced, and therefore, so we created a “spin-off” of the deliverable (D3.1v1) that we maintain in a distributed control version system.\footnote{https://launchpad.net/liquidpub/peer} Analogously, at some point, we reached some interesting result we wanted to communicate, so we took some of the work of the second version of our deliverable and produced a technical report: “Is peer review any good?”. These types of spin-off are captured by different branches in the graph of the line of research. Then, when expanding a particular scientific resource (is peer review any good?), we can see that the very same research result has many representations (Figure 2(b)). These alternative representations are the different views of the same resource, as in the example: slidesets, a technical report and a conference paper. We can also see how this scientific resource is semantically related to other entities. In Figure 2(c) we illustrate the use of particular dataset and experiments. The information described in the example captures the semantics of the relations, which today are hidden in the text and can be understood only after reading all the scientific resources.

Formally, we define the space of scientific resources $\Sigma = \langle SR, E, L, A \rangle$ where

- **SR** is a set of resources $r = \langle id, uri, ct, cf \rangle$ are the individual scientific resources. $id$ denotes the universal identifier for the resource. $uri$ points to the resource as available on the Web. $ct$ is the content type of the resource and can take values such as paper, video, slideset, dataset, experiment, and others. $cf$ is the content format which can for example be pdf, pptx, and so on. Because we consider journals as a way to create or at least to disseminate knowledge, they are also resources. Notice that scientific resources do not contain the actual scientific contribution (e.g., the text of a paper) but only a link to it.

- **E** is the set of entities that create, access, relate, annotate, or certify resources. These can be people or institutions (including certification agencies).

- **L** denotes a set of links $l = \langle es, et, lt, u \rangle$ representing relations among resources or between resources and entities (from source $es$ to target $et$). Besides the objects they relate, they are essentially characterized by a type $lt$ (e.g., “next version of”), by the users $u \in E$ that created it, and by the set $EN$ of users or agencies that endorse it, if any.

- **A** denotes a set of annotations $a = \langle e, at, v \rangle$ that can be attached to a resources or entity $e$. Annotations can be of a certain type $at$ (e.g., tags, flags, comment), and carry a value $v$ (e.g., “good example of state of the art”).

While the model allows anybody to create any kind of relation, the liquid journal model assumes and leverages specific relation types, to which it assigns an agreed semantic (and also graphical interaction patterns in the Liquid Journal interface).

1. **Temporal relations** model the evolution of a contribution, be it a paper or dataset or anything else. This is a natural behavior of research dissemination where for example we write a preliminary version of a paper and then we extend or refine it. Or, we clean or add more data to a dataset. Figure 2(a) also shows that evolution can follow a line (as in multiple versions...
of our project deliverable 3.1) or branch (from one deliverable we then derive a paper or a technical report). In particular, LJ assumes resources to be organized in trees, where each tree essentially denotes a line of research and tracks the evolution in time of related documents. The tree is described by relationships of type \(<child next version of parent>\), and LJ assumes that each child has only one parent. This model for SR reduces overload because it clusters contributions into research lines (which are themselves SRs) and then allows users to navigate through contributions in the line of research. It also allows to more fairly attribute credit to contributions or authors by making explicit the incremental nature of a contribution.

2. **Representation relations** model the multi-faceted nature of scientific resources. For example, a paper can have associated slides and datasets, and so be deemed as a complex multi-faceted artifact, including artifacts that encode (part of) the same knowledge but have different representation. In particular, LJ assumes that the relation type \(<B alternative representation of A>\) denotes that resource B is a way to render/present the concepts/content of resource A (e.g., B can be a short version of a long paper A). To simplify the model and make visualization manageable, we assume that if B is a representation of another resource A, there can be no resource C who is representation of B. In other words, representations are arranged in a star where there is a resource at the center, and many alternative possible representations can be connected to it.

3. **Structural relations** represent arbitrary relationships between contributions, where the relationship may be further described by annotations. For example, a paper can be related to a dataset in that it describes results of experiments on that dataset. LJs exploit these structural relationships:

   (a) \(<Re performed on dataset Rd>\) denotes that Re is an experiment, Rd is a dataset, and Re is conducted on top of dataset Rd.

   (b) \(<R1 reporting on R2>\) denotes that R1 describes a dataset R2 or reports on the result of an experiment R2.

4. **Authorship relations** denote who contributed to the creation of the resource.

   (a) \(<P contributed to R>\) indicates that person P contributed to the creation of the scientific resource R the typical example being the authoring of a paper. A typical annotation for this relation would denote the kind of contribution. For example, it may state that P contributed the state of the art section, or that P simply contributed by hiring the people who did the actual work.

   (b) \(<P edited JI>\) indicates that person P has been the editor of a journal issue JI.

5. **Usage relations** denote usage by mean of the LJ model. For example they include appearance of a resource in a journal, subscription to a journal, and sharing of a resource.

### 2.1.3 Liquid Journal Model

A liquid journal is an evolving collection of interesting and relevant links to scientific contributions (whether freely or not) available on the web. Considering journals as collections of links means that

---

D5.1v2 Design of the Liquid Publications Integrated Platform
journals do not own the contributions. We assume contributions are posted elsewhere (web pages, traditional journals, etc). Many journals can then refer to the very same scientific contribution. This “appearance” of contributions in journals is an important information we exploit for capturing interest. In terms of the model described above, we represent journals as a collection of scientific resources (which are indeed links, as discussed earlier).

The resources that end up in a journal (which define its content) can be decided by the editor who picks them one by one, or can be defined by a web search through the liquid journal engine, where the results are dependent on the reputation metrics and of the diversity-aware module as discussed next. The result of the search can be refined and then “snapshotted” by the editor (resulting in an issue of a liquid journal), or the journal can adopt a continuous model where the journal is essentially the web search, and the result evolves naturally and continuously as new content becomes available or as the values of metrics for existing contributions makes them qualified for the journal we have defined.

The rationale behind this model is that we see journals as a mechanism for people to find and share interesting and diverse content, for themselves or for their research group. While doing this, while running LJ-enabled search for web content, and while refining the results and sharing the most interesting contributions with our colleagues, we do a service to our team but, as we will see, we are also acting as “filters” in that we implicitly rate contributions. Hence, we are also doing a service to the community. LJs essentially put the community itself to work as content selectors, while having people performing activities they need to do anyways, such as looking for content and sharing interesting findings with their team. It is like capturing the interestingness people perceive from the result of a web search, and using this as a way to rate content and therefore separate interesting contributions from the rest.

Formally, a liquid journal LJ is a specific type of scientific resource. Unlike a paper that proposes a new idea or a new interesting content, it is a resource whose scientific value is that of collecting, aggregating, and organizing content. As such it helps in disseminating scientific knowledge and it contributes to science. The essence of the journal is manifested via specific relationships with - and constraints on - other resources it is related to. A liquid journal LJ is characterized by (related to) a set of editors, one or more publishers, a default editorial process, and a set of journal issues. Issues are themselves resources, and can have editors and can be related to processes in case the selection and publishing process for a specific issue differs from the default one. Both the journal and the issues are then characterized by a set of scientific resources, that represent the content of the issue. In essence, the journal has content that evolves (possibly even continuously), while issues typically represent snapshots of a journal that the editor decides to “solidify”. All these properties are expressed via distinguished relationships (such as ”contains” or ”is edited by”) that are known to the LJ software. Notice that since resources are links, solidifying an issue means solidifying the links, not the content pointed by the links.

Because of the assumption on resources defined in the previous section, this means that a journal is now a “view” over resources available in the resource space, that can correspond to different kinds of contributions (or annotations over contributions), that can be navigated to find different versions or representations, and that can be more easily consumed and understood thanks to annotations and relations to comments, datasets, experiments, or blogs. The essence to make things work is that the relationships are maintained by the community and, as we will see, the members in the community do so based on their own selfish interest. We will also see that the liquid journal
approach facilitates credit attribution, supports diversity and broadness in search, and encourage positive behaviors in science, and makes the scientific dissemination process more efficient. We begin by discussing how the LJ application supports the creation of liquid journals and showing how, while searching for interesting content to fill our journals, we provide an alternative and we believe more efficient and effective way to assign reputation to science and scientists.

2.1.4 Creating and Filling a Liquid Journal

A liquid journal can be created in two ways: by selecting and adding content manually or via a web search. In both cases, the characteristics of the LJ model are exploited to facilitate the collection and identification of the appropriate content to share.

**Manual Content Selection.** Manual selection mimics the way we collect and share interesting content today. There are various ways in which we identify scientific artifacts we’d like to read, use or share. For example, colleagues send us links or attachments via mail, recommending us to read them, and possibly even adding why (e.g., "take a look at this paper, it has a nice idea although it is not really developed or validated"). Or, we can stumble upon interesting material while searching or browsing the web. As another example, we may be listening to a talk at a conference and be browsing the conference proceedings and decide that the paper being presented is interesting, and that we should take the time to read it at the earliest opportunity.

The LJ infrastructure (and ecosystem, as we will see) supports this by facilitating editors in

1. **Gathering** scientific resources of interest into a *cooking area*, which is where resources are "parked" and then explored before being possibly published.

2. **Exploring** the relationships of the resources in the cooking area with other resources (e.g., newer versions of a paper, updated releases of a scientific experiment, different representations of a concept).

3. **Selecting** the actual resources we want to place in the journal

4. Creating a new issue and **publishing** it

The gathering phase can be done over time in one of the following ways: i) by dragging and dropping PDF files we have on our PC or receive as email attachment; ii) via "add to my journal" links that web site in the LJ ecosystem place next to scientific resources, much like links to digg and delicious; iii) via browser plugins that detect scientific resources in the page, analogously to what CiteULike plugins do for some web sites and some papers; iv) via pictures of papers, e.g., via an iPhone application that people can use to take a picture of a printed paper. For both the mobile phone example or the PDF email attachment, the LJ infrastructure takes care of identifying the link to the scientific resource available online and adds this link to the cooking area.

The exploration phase is performed by accessing the cooking area and exploring resources much like readers of a journal can do. In this exploration, editors can decide to exclude items from publication, or to take additional / alternative versions and representations of a resource, or other related resources that can be explored by navigating the resource graph.
The selected contributions can then be published in a new issue of a journal and hence shared with our team, our colleagues, or the entire community. At this time, because resources have been included in a journal, reputation is also assigned to these resources, as discussed next.

**Content Selection based on Relevance, Interestingness and Diversity.** Search-based selection differs from manual selection in the way the cooking area is filled up. With search, the cooking area is filled with items that are related to certain keywords or related to a resource or set of resources. For example, we may search for papers and datasets on peer reviews, or that are related to a given paper, or that use a given dataset. LJ search results are computed based on three dimensions: relevance, interestingness, and diversity. At this stage of the work we have not done proper research work in search over LJ. However, we describe some initial ideas and initial implementations, we discuss the possibilities that the LJ model offers in terms of search, and how they can be exploited.

Relevance can be computed with traditional techniques (for example, searching for keywords included in titles or text of papers) but in LJ can also leverage the links among resources as well as keyword search in annotations. In LJ we do not have the bandwidth (and probably the competences) to do research on keyword extraction or keyword search but we will rather reuse methods already available. What can also be done in LJ is to use the relationships among resources to provide search results, and in general we can use a Page Rank approach to rank results.

LJ supports search by interestingness by computing reputation metrics looking at how resources are “used” in journals (included in journals, or shared, or annotated). The detail of how this is done are described in the deliverables of WP 4, but we provide hints below that emphasize how LJ usage can support novel reputation metrics that we believe to be more significant for search and ranking than traditional ones.

Besides ranking by relevance and interestingness, LJ supports broadness of search in terms of searching for contributions coming from different scientific communities. We experienced first-hand the importance of this when searching for prior art on peer review. In this work we collaborated in a multidisciplinary group that included computer scientists, physicist, and philosophers. When looking for papers on this topic, somehow each group ended up finding content in its particular community of reference. Thus, for each group there was a lack of diversity in the discovery process. To avoid this problem LJ aims at enforcing diversity when providing search results, both to achieve a broader perspective on a topic and to achieve cross-fertilization between communities. A simple way to do this is to tag each resource with community information and have a graph of communities with measures of distance among them. We have done initial work in this direction by discovering communities in a bottom-up fashion, by looking at publication data. We have also discovered mappings between people and papers with communities. We refer readers to the CET website[6] for further information on the ongoing work on community discovery. More sophisticated approaches can be developed by looking at the relationships among resources defined within the LJ application itself. This and other approaches (and tools) for enforcing diversity are part of our research agenda and are not described further here.

---

2.1.5 Usage model and evaluation metrics

A value proposition of LJ is that editors and readers provide knowledge that can help connect and assess scientific contributions. This happens in three ways, all supported by the LJ interface:

1. Editors implicitly evaluate resources by “publishing” them in their journal.

2. Readers implicitly evaluate resources by sharing them with their team. For example a professor or a PhD student may share papers they think interesting within their team.

3. Readers provide knowledge by linking and annotating resources. For example, a reader can state that paper P1 reports results of experiment E over dataset D, and extends the initial results of P2. They can also state that paper P3 performs a nice literature review. The latter action provides information that is useful for navigating from a resource to related resources and therefore to find “related” information.

With the first two actions, the scientific community collectively establish what is worth reading. Feedback in this form is not intrusive but gets benefitted by actions that are anyways useful for editors or readers. This work of “selecting” and sharing knowledge is what we do every day. What LJ tries to do is to capture this information by making it easy and convenient for each of us to select and share resources and by then implicitly using the collective (implicit) opinions expressed by people by selection and sharing content. In other words, by giving scientists a tool to collect, organize, and share interesting scientific resources we aim at having a way to assess the “interestingness” of such resources, and consequently a way to filter interesting knowledge and help manage the information overload. Furthermore, expanding the reach of metrics to other types of content and other activities will allow us to look into other aspects of researcher’s productivity. For example, we explore how to reward people sharing good ideas (e.g., who do so by posting them in a blog), selecting and creating good collections of contributions and also giving constructive feedback. Traditional metrics not only are unable to provide such insights but they are still based on citations, which have shown to have flaws.

The conceptual model of LJ also provides the information to capture these aspects in the dimensions of the scientific resources, in the subscription links, in the structural links that make contributions appear in journals, in the usage information (tags, forwarding, sharing).

From an evaluation perspective, we see the main contribution of this work in providing the basic information for evaluating all sorts of resources based on community opinions implicitly provided. Out of these, many new metrics can be developed, just like many citation-based metrics popped up once it has been possible to compute citations automatically. A trivial approach consists in counting the number of journals in which a resource appears, or the number of people that shares it, or tag it, etc. However, as it is unfeasible to provide a unique (and accepted) magic formula that captures all these aspects, in LJ we focus on providing the guidelines that will govern the instantiation of particular derived metrics. Indeed, we believe it is the community the one to decide what counts on the community. We are developing this concept with the metric uCount that, as the name suggests, captures both the fact that everyone in the community counts and that everyone is involved in the process of defined what counts in the specific community. The idea

[1]In collaboration with ICST, icst.org
is that anybody can then decide which metric formula to use to filter out the resources of interest when searching for content on the Web. In LiquidPub we are also proposing a way to compute reputation out of opinions, which is embodied as algorithms and a tool, OpinioNet, which is already integrated to get information from LJ and to also provide reputation information back to LJ so that it can be used to rank search results.

2.1.6 Liquid Journal platform architecture

Designing and implementing an infrastructure for supporting the LJ model requires solutions and strategies for the different aspects of the model: i) managing the lifecycle of the journals, ii) journal creation, evolution, consumption and sharing; iii) access to scientific content in the Web, iv) computing the reputation of contribution (for ranking), and the v) projection of these features to a user interface. The liquid journals architecture relies on specialized components designed for each of the aspects mentioned. In Figure 3 we illustrate these components. Liquid journals provide a view of the scientific content available of the Web. As scientific contributions, in the broad meaning of the paper, fall outside traditional sources (e.g., digital libraries) where standards can be applied, the infrastructure requires an access layer that provides us the abstractions for easily accessing and searching content residing in non conventional, dispersed and heterogeneous sources on the Web. In order to address this requirement, we rely on the abstraction of Resource Space Management Systems (RSMS) applied to the scientific domain.

The Resman system, a prototype implementation of a RSMS which is part of Karaku, provides a uniform access layer to resources available on the Web. It abstracts applications on top of the heterogeneity of the underlying services on the Web. The approach followed by the system is to rely on adapters, components that map the specifics of different and non compatible services to a common and uniform protocol. Resman then allows upper layers to interact with a registry of adapters and operate on the resources using different levels of abstractions. On top of Resman, the abstraction of Scientific RSMS provides a common and extensible conceptual model for scientific resources, and a set of basic services for searching and operating on these resources. On the foundation provided by the components at the bottom, the liquid journal core component builds the services that support the model introduced in this paper. These services are organized in groups of APIs as shown in Figure 3.

As we need to provide LJ editors the possibility of defining their own journals’ lifecycle, the architecture includes as lifecycle management component, the Gelee system. Then, OpinioNet computes reputation for resources based on the LJ data, as discussed later in this deliverable.

Services are very important in our architecture but to fully exploiting them it is necessary to provide a Web interface that facilitates the journal definition, search, content consumption and sharing. Without an appropriate interface, services are worthless. In this approach, we pay special attention to this issue and develop a rich Web application on top. We are also integrating our application with the Facebook social network with the goal of facilitating the sharing, and making it easier for people to use and connect with the system. This is possible due to the Facebook Connect service. At the time of writing, process and lifecycle management tools are not integrated with

---

5 http://project.liquidpub.org/resman
6 http://project.liquidpub.org/karaku
7 http://developers.facebook.com/connect.php
Figure 3: Liquid journals architecture
2.1.7 Related Work

In spite of the progress in dissemination models, the current model of publishing and evaluating scientific contributions remains almost the same. Novel models such as the deconstructed model [24] and the overlay journal [21] introduce interesting ideas yet to be explored and taken beyond structural changes to meet the Web. These models (and the traditional model) are still constrained to the traditional notion of paper, and so other types of contribution remain hidden. The social part, the study of behaviors that are good for science, such as early feedback, sharing and collaboration remain also unexplored. More importantly, none of the models tackles, and offers mechanism to face, the problem of attention. All these issues affect also the evaluation, which continues to be based on papers (citation-based, e.g., [17,18]) and so leaving out other aspects of research productivity.

The Social Web has made possible new forms of collaboration. Prominent examples are the social bookmarking services that allow users to share interest within communities. CiteULike [11], Mendeley [12], Zotero [13] and Connotea [14] are examples of social bookmarking services with the focus on sharing and organizing academic references. These tools come with social tagging features that allow people to collaboratively tag content so it can be easily found later. Thus, these tools provides storing, sharing and tagging of references to publications via shared collections and groups. In the case of public collections, readers can consume the content by browsing or via RSS feeds.

Tools for sharing and collaboration stand as a promising direction. These systems provide some foundations and results for further studies in the scientific domain regarding collaboration. However, they are only the “mean” for collaboration without a formal and complete knowledge dissemination model established. Moreover, taking technical aspects apart, one disadvantage of these services is that they rely on active users, that is, users who inject content into the system. Thus, the discovery is limited to what is already there. Our model builds on some social features of these systems but provide a complete model of dissemination designed specially to overcome the dissemination overload in the scientific domain.

Search is a common service on the Web and so search engine technology has been explored and applied to scientific content [19]. Specialized search engines, such as Google Scholar [15] and CiteSeer [16] have been developed for searching papers/books across multiple repositories using crawling techniques and protocols. Using another approach, the academic search engine, BASE [17] indexes the metadata from repositories that implement the OAI-PMH protocol. In addition to what the user can provide as input to the search (e.g. keywords), implicit preferences and collaborative filtering has also been used for bringing users content they might like [23]. This has led to general relevance and diversity algorithms proposals trying to balance user preferences and diversity (e.g. [16]). In the academic domain, recommendation of papers have also been explored in many

11 http://www.citeulike.org
12 http://www.mendeley.com
13 http://www.zotero.org
14 http://www.concretea.org
15 http://scholar.google.com
16 http://citeseerx.ist.psu.edu
17 http://www.base-search.net
D5.1v2 Design of the Liquid Publications Integrated Platform

Thus, academic search engines provide only a partial view of the scientific contributions dispersed over several sources on the web. They do not capture the user preferences and lack of proactive behavior. Users need to know what to search and how to search in order to get content. Search by similarity is not provided (or limited) and diversity in the results is not enforced. General approaches provide the foundation but their use in the scientific domain need to be modeled for the broader notion of scientific contribution, and other special issues of the scientific domain (e.g., ranking). In our approach we define the notions of interestingness, of diversity, of quality applied to scientific content, and combine them to rank and group scientific contributions.

2.1.8 Liquid Journals and LiquidPub

In addition to the benefits discussed in this section, with respect to the specific objectives of LiquidPub, LJ if successful and adopted as a model and system helps in achieving the following:

- It allows to model evolution and it facilitates people in stating evolution relations among resources by enabling authors, editors, and readers to define the relationships (and also annotate them to detail what the changes are about);

- it provides the basis for reputation metrics that are based on opinion of each of us in the scientific community by attempting to capture our implicit opinions we express by selecting and sharing content with colleagues. Furthermore, because incremental contributions are marked as such, it enables a proper evaluation of such increments (or of an entire line of research);

- it supports multi-faceting by considering resources of all types as first class objects, not just papers. It also makes it easy to navigate to them in searching for content or in reading a journal, as they are linked from other resources;

- it supports an efficient dissemination of knowledge, since posted contributions are spread by the community via a Web 2.0 version of the word of mouth. Contributions can be disseminated immediately when the authors think they are ready to do so and it can be done in the format and representation the authors think it is best for the specific content (or in multiple forms);

- because readers are likely to subscribe to content from journals and authors that post innovative ideas, early release of knowledge is more likely to be rewarded, and as such encouraged.

2.2 Liquid Conferences

2.2.1 Introduction

Research papers are ways of keeping active a conversation among researchers, whose roles and accesses can be different according to the position they hold towards the SKO that is commun-

Note that an alternative version of this section also appears in Section 3.1 of LiquidPub Deliverable D2.1v1. We consider this to be an example of Liquid Publication in action, with different deliverable authors making collaborative use of the same shared SKO resources.
The dynamic of standard academic exchanges in the “solid” world of classic publications is that of a conversation in slow motion: too long messages communicated at a too slow rate of transmission. The success of the email among the community of researchers shows how needed was a technology to quicken the exchanges. Nevertheless, this acceleration of learned conversations among researchers faces many institutional, social and technical obstacles due to the weight of copyright policies, research practices, like peer review, and old technologies of publication. One of the goals for the second year of the LiquidPub project was to rethink the dynamics of learned conversations by reconceptualizing roles, accesses, licensing models and providing adapted tools able to encompass the different needs of the accelerated conversation. We provide a description of roles and licensing models in [9], and this section presents an example of dynamic tool for research exchanges: Interdisciplines, implementing main aspects of the Liquid Conferences use case. In the following we will outline the structure of a liquid conference, which aims of solving research and communications problems in science related to standard models of publication. We show how single issues have been tackled and partially solved by the beta-version of the Interdisciplines platform we put online.

2.2.2 What is a Liquid Conference?

Following the terminology previously introduced, a liquid conference (or virtual conference) is a Scientific Knowledge Object (SKO) constituted by:

- A set of **contents** (structured in a “modular” form that will be detailed later in the presentation)
- A set of **authors** (defined by a profile and an institution)
- A set of **panelists**, or invited discussants (defined by a profile and an institution)
- A set of **reviewers** (profiles can be anonymized, but known by the moderators)
- A set of **moderators** (who can edit the content, edit the profiles of authors, panelists, and reviewers)
- A set of **readers** (registered users) (by registering by name and location, they can leave tags and evaluations on papers)
- One or more **administrators** (who can open/close conferences, export them, give/change permissions to all the roles)

Each conference is organized as a standard conference on invitation only. Invited speakers are alerted automatically, as they are “created” as authors in the Interdisciplines platform and invited to submit their SKO. Then, the SKO will remain open to discussion for a certain amount of time (usually two weeks) and then archived and available for reading only. The temporal dynamics allows for an extremely efficacious interaction among participants, keeping both the lively aspect of the face-to-face events and the standards of written scientific papers.
2.2.3 Roles

Here below we detail the different actions and states for each role:

**Administrators.** As an administrator of a liquid conference you can perform the following actions:

- Open a new conference
- Set profiles of people
- Appoint authors
- Set permissions for authors
- Invite authors
- Appoint discussants
- Set permissions for discussants
- Invite discussants
- Appoint moderators for a conference
- Appoint reviewers for papers/Set reviews as public or private
- Close the conference.

**Authors.** As an author of a liquid conference you can perform the following actions:

- Once appointed by administrator/moderators, authors can launch a new paper (in a date defined by the administrator/moderators)
- Appoint co-authors from the list of appointed authors. Suggest new co-authors who must be approved by administrator/moderators
- Assign different parts of the “granular” paper to different co-authors
- Assign collective co-authorship to the whole paper
- Reply to comments
- Suggest reviewers. See privately reviews
- Chat with co-authors and discussants
- Once the discussion has ended, publish on the website a “solid” version of the paper (.pdf)
- Export the paper in a .pdf format or “embed” it with the comments into another service.
Invited panelists. As a panelist of a liquid conference you can perform the following actions:

- Modify their own profile pages
- Comment on papers/Edit their comments (within the time frame the conference is open: after that, they will need permission from the administrators/moderators)
- Comments on other comments
- Export their comments into other Internet services
- Chat with authors (not implemented yet).

Reviewers. As a reviewer of a liquid conference you can perform the following actions:

- Write reviews of the papers
- Write reviews of the whole conference

Reviews are a special class of comments that are visible to the administrators/moderators/authors only. The name of the reviewers on the review is visible only to administrators/moderators, not to authors. The set of reviewers can overlap with the set of commentators and their profiles can be private or public.

Moderators. As a moderator of a liquid conference you can perform the following actions:

- Act as partial administrators, organizers, animators, and editors of each conference
- Access the administrator’s back-office with restricted permissions
- Set time parameters for a new conference (launched by the administrator)
- Have a profile on the website
- Edit the content
- Create new profiles of people
- Set/change permissions for authorship
- Moderate messages coming from readers
- Access reviews.
Readers. As a reader of a liquid conference you can perform the following actions:

- Read the whole content without registering
- Register to leave a comment which will be moderated by moderators
- Registration needs name, email, academic affiliation, if any, and country
- Each time the name of a new academic affiliation is entered, it is added to a database of academic institutions. Further readers can choose from a menu that is updated automatically
- Leave evaluations on papers and comments (just “read” or “liked”), tag papers and comments and have a list of “favorite” papers or comments
- Subscribe to RSS feeds, add papers or comments to their own blogs, facebook profiles and other services, like Delicious, Mendeley, Connotea
- Be easily contacted through mailing list for announcing new conferences and for possible participation in surveys.

2.2.4 Modularity and multiple authorship

One of the main themes of discussion among the LiquidPub team revolved around the modular structure of the SKO and the possibility to assign a more granular authorship to subpart of the SKO according to different levels of collaboration (see D1.1 and [6]). An initiator of a SKO, that is, the person who has the idea of creating a new starting point of a scientific conversation, may decide to assign specific areas of the SKO to collaborators for co-writing the paper. We implemented this need into the new Interdisciplines platform the CNRS team has developed in partnership with the start-up ColDev (Collective Developments). The platform implements most of the ideas behind the Liquid Conferences use case, and the technical details of the platform are explained in Section 2.2.5. The new platform allows for collaborative writing and multiple authorships of modular papers. We will outline the structure of the modular paper, will show some screenshots of the demo, and indicate the URLs of the online platform.

The modular paper. We have conceived a template for modular SKOs in which each section is pre-defined and autonomous. Pre-defined sections are:

- Title
- Keywords
- Abstract
- Introduction
- State of the Art
- Central Claim
Figure 4: Structure of the modular paper in Interdisciplines

- Results
- Methods
- Discussion/Conclusions
- Acknowledgments
- Bibliography

Each of these different units can be edited autonomously and can have a specific title. A possible improvement in liquidity that is not implemented yet would be the option to add as many “modules” as needed for each different SKOs, thus leaving even more freedom to the authors. Another improvement we envisage is to have a more structured template for each module, or a series of templates (CSS sheets) that the authors can choose in order to fit their data at best.

The structure of the modular paper is visible at http://www.collectivedevelopments.org/interdis/main.php?actionType=writePaper or in Figure 4.

Multiple authorship. Another innovation of the Interdisciplines platform is the option of managing multiple authorships for each module of the paper. The initiator of the paper (the first author) can add for each module of his/her paper a number of authors, if they are already in the website database, or suggest to moderators/administrators to invite authors. The order of authors’ names is established in the end by the initiator of the paper. His/her name is the default first name, but he/she can decide to change the order by just dragging names one below another.

For this and following links to Interdisciplines you must register at Interdisciplines and login. Also, the URLs are likely to change to http://www.interdisciplines.org/XXX in the future.
Figure 5: Managing authors and licenses on Interdisciplines
Managing bibliographies on Interdisciplines

Managing licenses. For each new paper, authors can decide the type of licensing model they want to endorse, if it is compatible with the overall licensing model of the website. The list of licenses appears in a menu and authors can easily select the appropriate one. Once the conference is closed and the destiny of the papers change, the authors still keep the privilege of updating the license of the paper. You can see the box for managing licenses in Figure 5.

Bibliography. Authors can enter their bibliographical references into the EndNote compatible template available on the website. They can assign the reference just to one paper or to others. They can also add an external link to an online bibliographical tool, such as Mendeley, to point to their online bibliographies. See http://www.collectivedevelopments.org/interdis/main.php?actionType=manageBibliography or Figure 6.
Figure 7: Submitting review on Interdisciplines
Reviews. Assigned reviewers can write an anonymous review through a template that has been condensed by looking at the practices of the main journals and main evaluation tools for research. See [http://www.collectivedevelopments.org/interdis/main.php?actionType=readReview](http://www.collectivedevelopments.org/interdis/main.php?actionType=readReview) or Figure 7.

2.2.5 Interdisciplines: an implementation of Liquid Conferences. Beta-version

In this section we briefly present Interdisciplines, an implementation of the Liquid Conferences use case. Deliverable D5.2v1 [13] and the presentation at [http://www.slideshare.net/ColDev/interdisciplines-20](http://www.slideshare.net/ColDev/interdisciplines-20) provide more information.

Conferences, authors, papers, discussions and reviews. Each conference on Interdisciplines gathers a selection of papers written or co-written by registered authors. An active conference is one in which all registered users can discuss, comment, evaluate and review each paper. After the termination of a conference - whose duration is determined by Administrator(s) and Moderator(s) - all discussion and commenting features will be disabled. Nonetheless, the full conference will remain fully accessible to the public as an archive (conference abstracts, papers, bibliographies, author biographies, discussions and comments). Finally, papers and discussions can be exported in PDF format at any point in time.

Interdisciplines manages five user types with graduated levels of permissions and access: Administrator, Moderator, Author, Panelist, Registered Reader. All contributors to conferences and to co-written papers are approved by either the Administrator(s) or the Moderator(s): prior to the release of any paper, an author has to be invited and granted permission to participate in a conference. Finally, Interdisciplines comprises an evaluation template for papers to be reviewed by all user types. Each review is revised by the Administrator before being transmitted to the reviewed author(s). Reviews are not public and can only be viewed by the reviewer, the reviewed author(s) and the Administrator(s).


You can also open the authors’, readers’ or participants’ page at [http://www.collectivedevelopments.org/interdis/index.php](http://www.collectivedevelopments.org/interdis/index.php) or in Figure 8.
Figure 8: Main page of Interdisciplines
2.3 Liquid Books

2.3.1 Introduction to Liquid Books

Liquid books (LB) are collaborative, evolutionary, multi-facet versions of the traditional books. In a nutshell, a liquid book is characterized by the following entities (in italics):

- **content** added by **contributors**

- A content is published to the public via **editions**, prepared by **editors** (editors are a subset of contributors).

- a **contract** defines who can add or consume content, who can create editions, what is the frequency of editions. A contract defines also the licensing/visibility policies for content among contributors.

- A contract also includes **processes** for attributing royalties to editors and contributors and for publishing editions

- a **publisher** publishes and distributes the editions of the book, typically holds the copyright, and possibly provides additional services, including IT services supporting LBs.

A traditional book is one instance of the model above. In a traditional book (think of authored books, not at edited books for now) we typically have one or a very small number of editions, typically not in parallel. All contributors (the authors) are listed as authors of the book, and get royalties based on a pre-agreed contract. Variations of the book are typically subject to new contracts. In particular, an author cannot just take the content and publish their own variation without contractual agreements with the other authors and with the publisher. Also, in a typical publishing process, there are rigorous quality checks. The “time to market” is overall very high, both because it takes a long time for authors to prepare the manuscript and because of the publishing process after the initial manuscript has been delivered, that also takes several months. Overall this resembles very much a waterfall software development process, with all its benefits and limitations.

The opposite extreme of a liquid book resembles more agile and open source software development, with some twists due to the nature of book publishing and copyright management. So, at the “liquid” extreme, we have a liquid book where contributors add content to the LB, possibly in a continuous fashion. Contributors can freely decide at any point to prepare new editions by collecting and editing content put by other contributors to the same LB, without asking for permission to the other contributors and without new contractual agreements. Each edition will typically be tailored for a set of expected readers. The royalties attribution process, defined in the contract associated to the LB, then kicks in. These processes can range from very simple rules (e.g., editors always take 20% of the royalties and the rest is shared equally with all contributors) to complex procedures (for example, a voting mechanisms that approves royalty distributions proposed by the editor). The key here is to have this process simple and well defined, and not to have to involve the publisher in it, otherwise the complexity is excessive and very likely this will not scale. Furthermore, at the liquid extreme we expect that:

1. The ”time to market” is low: because content is added frequently and editions are also prepared relatively frequently, it is important that updated are rapidly reflected into new
published editions. If we consider textbooks, for example (our main scenario as discussed below) this is important as frequently people only start preparing their books in the summer so that they can be used for classes in the fall.

2. The barrier for new authors to get to the first edition is low. Quality or completeness will improve over time. Keeping this barrier low is important to get people who have interesting things to say but that are scared by the effort of writing a book to actually start writing one.

3. The pricing should reflect the fact that books may be of improving quality and completeness. If we again take the example of textbooks, we can expect that students may not want to pay much for a book that is a bit more than organized lecture notes.

In the end the benefit that we hope to get to with the more liquid extremes is to:

- Have more people write books, including in particular busy but competent people who would not normally put the very considerable effort it takes to come up with a first edition of a traditional book.

- Ensure that the book can evolve over time in an agile fashion (also with limited effort required from edition to edition) and in this way have up to date editions

- Thanks to concurrent editions, ensure that it is possible or even easy for contributors to prepare editions that are targeted to a specific group of readers

- Facilitate collaboration and reuse, as editors can take, edit, and aggregate content from multiple contributors

### 2.3.2 Liquid Books for Teaching

To motivate the need for LBs, we discuss the case of textbooks even if the concepts are generally applicable. Textbooks are a case where it is very easy to see the value of liquid books, and they are also an important segment (in terms of market size and also of impact on society).

When teaching, professors and students need a book which is tailored for the class, which is up to date, which includes (or points to web-based) exercises, which has a companion web site with additional readings, etc.

From the perspective of professors, it is very hard to find a book tailored for their class, meaning that they have to resort to a collection of books, with the related problems (cost, inconsistent description, difficulty in exposing a coherent set of topics in a coherent way, difficulties in linking topics and classes and exercises, etc.). Furthermore we as professors do not always agree with what is written in a book, or do not like the way it is presented, or it makes assumption on prior knowledge of the readers that does not hold for our class. Even when we find books we like, they become outdated over time and sometimes the new releases are done in a rather sloppy way.

The main idea behind LB in textbooks is to allow professors to compose and create a customized, evolving book for their class by putting together modular (but “refinable”) content from other colleagues. The scenario we envision is the one in which a few professors share their teaching materials (slides, sections on given topics, exercises) in a shared pot. Then, each of these
professors, or a group of them, collects material from this pot that they think interesting for their class, and organizes it in a flow that is consistent with what they want to teach. This is the edition. Realistically, creating an edition will involve much more than composing existing content. Whenever we take pieces of content written by others, we will need to adapt it to the context in which it will be inserted. We may also want to change or smooth some sentences, or remove them, add references, and so on. Changes can go to minor modifications to heavy editing. Again, the key to enable this is to have a proper contractual framework that allows professors to do this easily, and a publishing process that also facilitates this, as discussed above. Realistically we see this happening between a group of professors that trust each others. At any time of course a professor can stop adding content, can join other LBs	extsuperscript{20} etc.

In the case of textbooks, applying the model above brings the following benefits:

**Benefits for readers (mainly students):** LBs are up to date and tailored to their needs. At one extreme, there could be a different book for each class and each year. Hence, students would get a text that is easy to follow. Because the text (as well as companion exercises) is prepared with content from multiple experts who also have experience in teaching, it is likely that the edition can be of high quality and with interesting insights and opinions. Finally, we see a benefit in that students have a chance to read content that otherwise would remain buried in slides of specific professors. Because the entry barrier to writing the first LB edition can be low, we can get people scared by the effort to write a traditional book to actually collect, polish, and publish their lecture notes. This can be an untapped gold mine of interesting knowledge that we can now access and reuse and make available to students, possibly integrated and refined by other different contributors.

**Benefits for contributors (typically professors, in this case):** The main benefit for professors is that they have a high quality book for their students, and access to a variety of exercises possibly developed by colleagues. This way preparing classes and teaching becomes simplified. Furthermore, professors can more easily become authors, and this gives credit, personal satisfaction, and possibly a little bit of money.

**Benefits for publishers:** publishers benefit because i) they can get to publish (good) books from people that ordinarily would not write books, ii) they have up to date editions (which also makes it important for people to get the last edition as opposed to reusing previous ones), iii) they are likely to have a publishing process that has lower cost, and iv) they can experiment entirely new business models, more similar to how software is sold (e.g., they can sell individual editions, updates, all editions of a LB, etc).

2.3.3 Conceptual Model

We now describe in some more detail the conceptual model for liquid books. The version presented here already includes concepts inspired from liquid journals as we realized over time that we could reuse many of the concepts developed there, and in fact ideas we were not thinking of for books came out after developing the journal model.

The conceptual model is shown in Figure 9. The figure shows that a liquid book has a set of
contributors, essentially those that sign the contract associated to the book). These contributors can add content to a book, and can also create editions. The contract includes a publishing process (procedures for publishing but also indication of time to market and expected quality levels) as well as a process or policy for determining how royalties are distributed. Currently we do not distinguish between people that can only add content and people that can only create editions. If we will see the need for this distinction at the modeling level, we will add it (this distinction can also be explicitly written inside the contract). In this model we assume that a book always has a publisher. Cases where people publishes their own book by simply putting it on a web page are modeled with default publishers (e.g., “self”).

For modeling the content, we rely on the liquid journal model, with the use of specific types of relationships. This is not shown in the figure as the base model here is the same as LJ and we emphasize this by writing in the figure that the content is a resource, as per LJ model. The reason for reusing LJ resources is that we want to have the same ability to link, annotate, tag, evaluate resources as this can help contributors navigate through the set of resources of a book, search them, etc. Again since LJ does not manage access control, the assumption is that if LB contributors do not want resources to be visible, they can do so. We also assume that a resource belongs to one and only one book, from an LB perspective. This means that if a contributor has more than one LB and wants the same resource to belong to both (assuming that the publisher allows this and that this is in compliance with the contract), then this means duplicating the resource (remember that resources in LJ are pointers, so contributors have the option of simply duplicating the pointer, not necessarily the actual text). Treating content as LJ resources also allows us to naturally manage branching and evolution. Contributors may take and modify content done by other contributors, and a relationship can be defined between the original and the modified resource to reflect this. Relationships can be
annotated with the degree of changes if this is considered helpful by contributors (or required by the contract).

Another benefit of modeling content as LJ resources is that we have a natural support for multi-facet representations. While in the discussion above we always assume that a book is a pdf, a LB can have companion web sites or additional material, and the LJ model allows to link resources of various kinds to state relationships that can be used to populate a companion web site for example.

The figure shows that an edition is modeled by a link to an entry point, a specific content (resource). This resource will be linked to other resources to define a serialization of resources that will be included in the edition. Some of these resources will represent chapters, others will be table of contents, and the like. In other words, we assume that to create an edition a contributor, besides taking and refining the content, they aggregate it by defining a distinguished kind of relationships between resources called LB follows. A resource can have only one source and one target connected via this relationship, as it is supposed to define a serial order.

### 2.3.4 Modeling, Business, and IT decisions

The above conceptual model is quite general and can model traditional books as well as liquid books. Part of the essence of LB however cannot be grasped by looking at the model, and they are not related to IT. Many of the issues relate to the details of the credit attribution process and to the publishing process. We list below some of the issues we have identified:

- what should be the expected quality of the first edition? (and correspondingly what is the publishing process in terms of review and quality checks?). This can range from a printout of slides used to teach, for example, to a traditional book.

- If time to market is important, what can be the (best, quickest) publishing process? How can publishers create specific processes for LBs, and should they do so?

- Related to the point above, which processes and initial quality levels are likely to attract interesting authors while allowing for an acceptable quality level?

- What can be the royalty attributions options, or decision processes?

- In the discussion above we assumed that author names for individual chapters are not listed. The contributor that prepares an edition simply takes content and his/her name appears on the cover (possibly names of all LB participants are listed somewhere). An alternative approach is the one in which for each chapter or section, the contributors discuss about who should be listed as contributors, which is not always easy to determine (it depends on who provided the original content, who modified it and to which extent, etc.). For this, we expect processes to be in place analogous to the ones for attribution of royalties.

- What should be the pricing model, and what is sold?

- What are the sales channels and strategy?

- How freely can a subset of authors take the material and start a separate LB? Which are good contract templates that can cover this issue?
What are, in general, the options for the contractual frameworks? Which are reasonable contract templates that can be proposed to contributors that want to start an LB?

Is it possible to sell multiple concurrent editions of the same book? How frequent should editions be? Is having too many editions a problem, and if so what is the appropriate number and frequency of editions?

We are currently working to find answers to all these questions and beginning to experiment this and variations of the above model. The description and the preliminary results of this experiment are included as an appendix in the D1.2v2. The model has also attracted interest from communities as shown at [http://icst.org/liquid-publications/](http://icst.org/liquid-publications/). We assume that many of the answers can only be found over time and as people begin to experiment on LB. We will now begin to advertise the model and search for authors interested in collaborating with this modality.

As the reader will have notice, we do not discuss IT issues here. There are two reasons for this: the first is that we think the core problems lie in defining the model and reasonable templates for contracts and publishing processes as per the issues above. With this done, a publisher such as Springer can begin to launch a LB series even without any kind of specific IT support for LB. The second is that we do not have IT for liquid books as in scope for the project (which we prefer to use for the other parts of the project where IT is important to prove the concepts and demonstrate the value). This being said, we are noticing more and more that the LJ model and infrastructure (and the SKO model and infrastructure, as LJ concepts are integrated into the SKOs) can be used as a base for supporting an IT platform for LB, as already discussed above when detailing the model. We will further explore this option during Year 3 of the project.

### 2.3.5 Related work, novelty and difference with other book models

A question that often comes up when we present LB is about the difference with respect to Wiki-books [WB](http://en.wikibooks.org/).

WBs are open-content and freely distributed, as they adopt a GFDL license [which ensures](http://it.wikipedia.org/wiki/GFDL) that content remains copyrighted to the authors, while at the same time the copyleft licensing allows to freely distribute and reproduce such a content. WBs are “public” in that everybody can see everything. Also, in WB anyone can contribute by adding new content, editing or deleting existing one. Everybody can participate, so a WB can have thousands of authors.

LB are very different from this model in that:

1. LB are not necessarily open-content. The content is exposed through the editions. This does not exclude (but also does not require) an approach where the content is also made public or freely available (based on certain licensing policies defined by the contributors or the publishers).

2. The collaboration in LB is more on the contractual side: contributors may collaboratively edit a text, as done in wikis, but this may or may not happen and it is not the essence and...
value of LBs. The essence is in allowing reuse and modifications based on contractual agreements that define attribution of royalties.

3. Each piece of content in LB can exist in multiple versions, both because the original contributor can evolve it and because other contributors may take it and change it for putting it in their on edition. Around WB there is instead a social interest in having a reference point, a unique version which can be taken as reference.

4. In LB, the responsibility for the content ultimately rest on the LB editors of an edition. While in WB the focus is on the content creation process, in LB it is on the edition creation process by reusing, modifying, and organizing content. A distinction has to be done between “functional work”, (e.g., manuals, computer programs, recipes) where it is more easy to establish rules about reuse and modification of contents and “opinion-type work” (e.g. scientific reports) where a modification can lead to misrepresent the authors ideas or thoughts [25].

5. Editions are static collections (an edition is analogous to the edition of a book today), the LB itself remains in its evolving state and contributors can keep adding or editing content. We can see this as a branch in software development.

6. In LB the we still have the notions of publishers and editions with ISBN, DOI, etc.

With respect to other related efforts, a first attempt to create evolving textbooks is in place by MacMillan, one of the five largest publishers of trade books and textbooks. There are several points in common between our vision of LB and MacMillan’s one of DynamicBooks [23] as well as some differences. DynamicBooks will allow instructors to edit digital editions of textbooks. In this way, professors will be able to customize the book for the needs of their classes, deleting chapters, uploading different kind of materials (notes, videos, pictures, graphs). However, the big breakthrough is the fact that professors will be completely free to rewrite or delete paragraphs, equations, illustrations without consulting the original authors or publisher, they have just to log into the system.

For what concern the business aspects, MacMillan will sell online access to the students, and the modifiable e-book editions will be much cheaper than the traditional printed version; even if the publisher will also offer print-on-demand versions of the customized books for an higher price. This can be seen as a particular case of liquid book with specific and very simple predefined rules and processes. They do not envision any customizable publishing process or legal framework to constrain possible changes to books, publishing of editions, attribution of credits and royalties, and the like. On the contrary they rely completely on the instructors and on students or other instructors to help monitor changes, even if the publisher reserves the right to “remove anything that is considered offensive or plagiarism”.

For what concern the collaborative aspect, there are several examples of collaboratively-created books, even if they are more similar to classical e-books, without the multi-facet aspect or the evolving one. A first example is “How to Think Like a Computer Scientist” [24] series of publications by Green Tea Press, where the same core programming text has been adapted to several

---

different programming languages (Java, Python, C++, etc.). On the same line, *97 Things Every Programmer Should Know*\(^{25}\) another example of collaborative written book, with hundreds of contributors, where multiple and varied perspectives are collected about things a programmer should know. Contributors have become involved through announcements, specific invitations, recommendations and word of mouth. Another example of collaborative written book is *Business Model Generation*\(^{26}\) with 470 co-authors and distributed without a publisher. The last three initiatives do not have a specific platform for collaboration, they usually rely on call for contributions and then on manual integration of contributions. A first attempt to facilitate the search of contributors and the collection of contributions is the one by Fast pencil\(^{27}\) a platform to write, publish and sell collaboratively written e-books. The above mainly focus on collaboration on content creation, which again is not the focus of LB.

### 3 SKO Platform and Knowledge Bus

The originally planned function for the SKO model was to be the base format of a central repository that would contain all the data, metadata and basic services to enable all the features of an integrated LiquidPub platform. Nevertheless (and as explained in Section 1) during the second year, three main use cases (Liquid Journals, Liquid Books and Liquid Conferences) were introduced as the project’s main driving forces. Eventually, each use case evolved its particular set of needs and terminologies that would adapt and change through their various iterations. The SKO model accompanied these evolutions with numerous self-revisions and sometimes even suggestions, but this very fact hinted toward the shift of focus from the top-down (from SKO model to use cases) to the bottom-up (from use cases to SKO model) approach.

As a result, the role of the SKO model and the LiquidPub Core Platform gradually shifted to become the project’s knowledge bus and repository that would connect the three use cases and manage the transference of information not only between them but also with other platforms and formats external to the project (through infrastructure services). This back-end (i.e. data and low-level services management) exclusive role was further consolidated by the decision of allowing each use case to design and implement their own user interfaces. In the end however, this shift of focus, did not impact the concepts behind the SKO theory described in D1.2v2 \(^7\) significantly (if anything, it made it become more flexible and able to cope with the evolving requirements of the use cases).

The main objective of this section is to provide a conceptual matching between the the SKO model and the models and functionalities required by each of the use cases. The implementation-level matching of the the services and data has also started and a work-in-progress version of this work is included in D1.3v1 \(^8\). Most of the SKO model features discussed in D1.2v2 \(^7\) (e.g., layered structure, version control, annotations) are implemented or prototyped in D1.3v1 \(^8\), using the general model shown in Figure 10 as a base.

26 http://www.businessmodelgeneration.com/
27 http://www.fastpencil.com/
3.1 SKO Conceptual Model

It is worth noting that the conceptual model from Figure 10 has a slight focus towards the actual implementation details of the SKO model, as it contains several entities that are mostly internal low to mid level representations of information in the system and that are never directly altered or shown to the potential users (human or service-using application). Such an implementation-oriented diagram is useful to compare the SKO model with the main use case concepts presented for Liquid Journals (Section 2.1.3) and Liquid Books (Section 2.3).

More specifically, Figure 10 contains the following elements:

- **Low-level entities**: the boxes with the dark background represent entities that are internal (i.e. users or external services do not interact directly with them) to the system. These types of entities are only used internally in the core platform but they were included in the representation of the model to highlight special qualities of the high-level entities.
  - **Basic entity**: contains several attributes of entities at a higher level. The high-level entities SKO annotations, SKO and SKOnode (through Content entity) are all special cases of Basic entity. It was mainly included in the diagram to convey the common aspects and functionalities of other higher-level entities.
  - **Content entity**: this entity is generalized into Basic entity. Content entities contain attributes and services that are common only for both the SKO and the SKOnode entities. This conveys that among high-level entities only the SKO and SKOnode entities have actual content, while the SKO annotation entity only has meta-content.
- **Selection and Fragment definition**: these are intermediate entities that are used to point a SKO annotation to a complete or a part of a particular Basic entity. These are displayed to remark the fact that SKO annotations may refer to specific subsections of content (and not only to the whole content).

- **Annotation type**: this defines the general properties and restrictions of each particular type of annotation. For example, this could be used to create a “single comment” annotation type that is used to store text comments for other entities.

- **Scientific resource**: this entity is the one that actually contains the content. This entity is, in fact, outside of the model and the core platform and is accessed via SKOnode that wraps Scientific resource.

- **High-level entities**: the regular boxes represent entities that are accessible and modifiable from the exterior of the LiquidPub Core Platform. These entities are in charge of implementing all the services that the LiquidPub use cases need and the interaction with them is mainly managed through the API interface defined in D1.3v2 [8].

  - **SKO Annotation**: this entity enables the creation of custom attributes and relations between all the other entities in the model. SKO annotations have one (mandatory) source that represents the main entity to which the SKO annotation refers to and an optional destination set that represents other related entities (mainly used to represent relations). For example, SKO annotations may be used to store the comments that refer to a specific part of a document, to declare that a paper is related to another paper, or even to capture the version history between a set of texts.

  - **SKOnode**: this entity is the only one that directly interacts with the external scientific resource. As such, its main objective is to wrap the scientific resources and enable the system’s operations and services on it (e.g. defining attributes, relations, comments, etc.).

  - **SKO**: this entity not only aggregates one or more SKOnodes but it also defines attributes and relations that are specific to this aggregation (and maybe not present in the individual parts).

- **Use case entities**: the double-bordered boxes represent entities that contain the data and implement the services required by each of the LiquidPub use cases.

It is worth noting that the use case entities are external to the SKO model and the LiquidPub Core Platform, as each use case defines its own model, requirements and even services. Because of this, the SKO theory has no direct direct control over them but instead it uses the high-level entities to generalize the use case entities and implement their required data and services. This means not only that a concrete matching (presented in the following subsections) between each of the use case entities and the SKO model can be established but that also that, in the future, the SKO model could potentially accommodate new use cases with their unique requirements (such as xyz pillar in Figure 1).

To further facilitate this conceptual matching, Figure contains a custom simplified diagram in which only the most important entities from Figure are kept and the internal details of the SKO-based platform are hidden.
This simplified model evidences that (through the SKO) all of the three LiquidPub use cases are focused on enriching, organizing and offering additional services on existing scientific resources. Furthermore, this simplified model will be used as the base to the matching being done on the following subsections.

3.2 Liquid Journals Matching

This section covers the matching of the Liquid Journal conceptual model to the previously presented SKO model. The description of the Liquid Journal concepts may be found in Section 2.1 but Figure 12 presents a simplified overview.

From Figure 12 its worth further discussing the following items:
- **Scientific resources**: as explained before, this entity is external to the platform. For Liquid Journals and Liquid Books, scientific resources represent not only the papers, wikis and blogs but also figures, datasets, charts and videos that can be found in the Internet.

- **Persons**: much like scientific resources, persons is also external to the SKO model (this is managed externally in the use case pillars for the moment). In Liquid Journals, the persons entity represents the authoring source of all the content/meta-content and also represents the editors (i.e. who determine the content selected) and subscribers for the Liquid Journals.

- **Annotation and Relations**: in general, this entity is used to represent all the relations, annotations, and other metadata needed by the Liquid Journals. All the semantic relations between the entities in Liquid Journals can be also encoded as SKO annotations.

- **Liquid Journals**: are basically sets of links to scientific resources that are selected by users in two ways: query-based (intentional) and manually cherry-picked (extensional). The SKO entity from the SKO model supports both of these ways of selecting sources. Furthermore, as shown before all the custom and additional information or relations needed by Liquid Journals can be directly encoded as SKO annotations.

- **Liquid Journal issues**: these basically are snapshots of Liquid Journals that only contain data selected extensionally (direct enumeration of items to include to an issue). As in the previous case, this can be implemented directly by a SKO entity.

### 3.3 Liquid Conferences Matching

This section covers the matching of the Liquid Conferences conceptual model to the previously presented SKO model. The description of Liquid Conferences concepts may be found at Section 2.2, but Figure 13 presents a simplified overview.

![Simplified Liquid Conferences model and its mapping to the SKO model](image)

Figure 13: Simplified Liquid Conferences model and its mapping to the SKO model
From Figure 13 its worth further discussing the following:

- **Persons**: similarly to previous iterations, persons create all content and meta-content. However, in this case, the persons can also take roles like “panelist” or “moderators” that could influence the evolution of the papers and the conference.

- **Content Parts**: are a special case of Scientific resources and, as such, they are considered external from the system by the SKO model (despite the fact that they may be indeed stored in the same server). Content parts represent the high-level parts (e.g. introduction, related work, claims, experimental data) that compose a Paper.

- **Paper parts**: these wrap content parts with metadata (allowing Content parts to receive comments, reviews, etc) and also allow for linking external media to them. In the SKO model, this role (i.e containing meta information about the individual parts) is taken by the SKOnode entity. Note that the division between content parts and paper parts is not explicit in the current Liquid Conferences use case but it was made here to help with the explanation of the matching.

- **Papers**: the current specification of Liquid Conferences takes scientific papers as the basic unit of presenting work. These papers are implemented as an aggregation of particular paper parts, which are aggregated (for the moment) in a fixed way in the platform. In any case, the SKO entity from the SKO model is able to represent the papers as defined by the needs of the Liquid Conferences use case.

- **Comments and Reviews**: these entities can refer to papers, paper parts and even conferences and they add meta-information (in the form of text comments/discussions or marks) to them. The current implementation of comments and reviews from the Liquid Conferences use cases could be implemented mainly by the use of SKO annotations.

- **Liquid Conferences**: these contain the set of links to the papers, persons and conference rules that belong to a specific instance of a conference. As such, a SKO entity can be used to represent this entity in the SKO model.

### 3.4 Liquid Books Matching

This section covers the matching of the Liquid Book conceptual model to the previously presented SKO model. The description of the Liquid Book concepts may be found at Section 2.3 but note that the model displayed in Figure 9 refers more to the life cycle and copyright/licensing aspect of Liquid Books, while pointing back to the Liquid Journal content representing model from Section 2.1. Because of the previous, the simplified model shown in Figure 14 presents a content-representation model similar to the one from Liquid Journals but focusing more on the aspects of version and representation tracking that are more important for Liquid Books.

From Figure 14 its worth further discussing the following:

- **Representations and Versions**: Different versions, renderings/representations of the same scientific resource, relations between resources, and annotations may all be encoded as SKO annotations. For example, an annotation encoding that a scientific resource A is the previous...
version of the scientific resources B and C would be represented as a SKO annotation in which its source=A, destination={A,B} and name=is-version-of.

- **Liquid Book repository**: similarly to Liquid Journals, the Liquid Book Repository is a set of links to scientific resources, representations and versions that represent all the content that it is deemed relevant by the creator of that particular Liquid Book. A Liquid Book Repository can, in the same way, be represented by a SKO entity.

- **Liquid Book edition**: is a sequential ordering of content from the Liquid Book Repository and whose objective is to prepare the content for consumption by other persons. Another SKO, in addition to the one representing the repository, can be used to represent this ordering.

- **Persons** and **Scientific resources** are mapped as in Liquid Journals

Note that most of the additional entities from Figure 9 (e.g., Contract, Edition, etc.) can be represented by using specially tailored SKO annotations but the entity Publisher will need to be defined in the Liquid Book use case pillar as it does not have an internal representation in the SKO model (in the same way as the Person entity is maintained in the pillars).

### 3.5 Enabling Other Services through the SKO Knowledge Repository/Bus

As explained in Section 1 adding new services that can operate on the data and meta-data of a given use case could be done by either integrating the functionality directly with that use case or by integrating it with the SKO-based core platform. While the former will result on a better tailored implementation of the desired functionality the latter guarantees that the particular functionality would be available to all of the pillars that are built on top of the SKO-based core platform. Thus, services such as reputation will most likely be initially implemented and tested on a particular use
case pillar. Over time, according to the generality of such services and the needs from the other pillars, they may migrated to the SKO knowledge repository/bus.

4 Reputation services

This section presents models and architecture of the OpinioNet reputation module and of the Research Impact Evaluation (ResEval) module, developed in LiquidPub. OpinioNet computes the reputation of both researchers and research work (or SKOs) based on the innovative algorithms developed in WP4 and getting input from the LiquidPub applications and repositories. ResEval is instead a web application that allows people to view a variety of reputation metrics and compare them. It also computes “traditional” reputation metrics such as citation counts and h-index. ResEval provides this information as an additional input to OpinioNet, and in the future it will be able to display the reputation results computed by OpinioNet. The integration of the two tools is described in details in D5.2v1.

4.1 OpinioNet

The OpinioNet reputation module provides a set of algorithms for computing the reputation of two of the resources defined in the LiquidPub Platform: contribution (or a SKO) and person (or a researcher). In both cases, reputation is computed depending on which information sources contribute to this calculation. The table in Figure shows the types of reputation and the different information sources considered by OpinioNet. The reputation of additional roles (e.g. collaborators, invited speakers, etc.) as well as additional knowledge related entities (e.g. libraries, institutions, etc.) may also be investigated in a similar manner; although we leave these issues for future work. We note that other reputation metrics such as h-index and g-index are provided by ResEval.

OpinioNet is wrapped into a RESTful web service and integrated in ResEval as a new pluggable metric (see Figure).

The reputation of a scientific contribution is computed in terms of the opinions the contribution has received. The reputation of researchers, however, depends on the role that researchers play. We currently distinguish between two roles that can be played by researchers inside the LP world: author and reviewer. The reputation of an author is based on the reputation of its contributions. However, the reputation of a reviewer may also consider an analysis of his/her reviews or his/her social network.

OpinioNet is wrapped into a RESTful web service and integrated in ResEval as a new pluggable metric (see Figure).

The Representational State Transfer (REST) Interface for computing the reputation metric is the following:


The text of this section is partially reused from to keep this deliverable self-contained.
where the input parameters represent:

- **Object**: Id or name of the object to be evaluated; it could be the name of a contribution or a person.

- **ReputationEntity**: The name of the entity to be evaluated. For the time being only four options are possible: liquid journal, contribution, author, and reviewer (although the reputation of authors and reviewers are equivalent in the current implementation).

- **ReputationMetric**: In this case we want to compute the reputation based on a given metric. For the time being the options are the following: contopinions (to make use of the opinions about contributions), authorcont (to make use of a researcher’s contributions), revreviews (to make use of a researcher’s reviews), or revnetwork (to make use of social relations between researchers).

- **MetricOption**: an option of how reputation will be calculated (for instance, based on citations, opinions, etc.).
Following, an example of a computation request for a contribution and its result is given. This link computes the reputation of a contribution with id = 10: \[ \text{http://eru.iiia.csic.es:9090/ReputationService/resources/compute_reputation/10/contribution/contopinions/none} \]

The result of the reputation service invocation will be an XML with a number representing the reputation of the object evaluated. The data to compute the reputation is retrieved from different sources such as ResEval and Karaku. A complete scheme of the reputation module can be seen in Figure 17.

### 4.1.1 Implementation

A detailed description of the algorithm is presented in Chapter 4 of Deliverable 4.1 [11].

The API specification of the OpinioNet is available in Google Docs at: \[ \text{http://docs.google.com/Doc?docid=0AVLiEhF9ZGeIFZGhoM3FyeD1fMGpqbXd0NGho&hl=en&invite=CJH7-dYF} \]

The code of the OpinioNet reputation module can be found at \[ \text{http://project.liquidpub.org/reputation/docs/propagation_demo.zip} \]. In the future, as the code becomes stable, it will be available on the LiquidPub svn.
Figure 18: The architecture of ResEval
4.2 ResEval

The Research Impact Evaluation (ResEval) module[^1] is in charge of managing and computing the metrics of scientific entities, such as authors and contributions. The architecture of ResEval is defined in three well-differentiated layers in Figure 18. Those layers are the typical ones that are defined in a three-tier architectural pattern, namely the user interface, the business logic (core) and the data layer. From the client point of view we have implemented two types of interfaces, to grant both web access through “classic” web pages, and automated access using the REST interface, which is the most commonly adopted solution for web services. The choice of developing multiple interfaces is motivated by the need to have, eventually, the possibility to embed the application in more complex platforms and applications. Since during the design phase the flexibility requirements have been taken into account, the system is ready for changes or extensions required for its inclusion in other systems. The caching system on the data level is clearly motivated by performance reasons, since we are dealing with a web application that often queries external services, such as Google Scholar and, therefore, loses much time due to network’s overhead.

**Interface layer:** this layer provides the main functionalities of the tool through several user interfaces. The metric management interface offers all the functionalities needed to create new metrics or update existing ones. The metric computation interface allows one to compute existing metrics defined using the available options and information source (e.g., Google Scholar).

**Core layer:** this layer stores and manages all the definitions and logic of the metrics. Metric manager is the core component of the tool, it that allows one to create, update and delete metric definitions. Computation engine computes the metric using the specified algorithm and its parameters that indicate metric definition and information source. The modules that can be identified in the metric architecture, and that can be seen in Figure 19, are the following:

- **Information parser:** is the module that parses the information received from the application and prepares it for the metric computation.
- **Information retriever:** is the module that gets the information from the corresponding source and prepares it for the metric computation.
- **Metric computation:** is the module that allocates and executes the algorithm that computes the metric.
- **Response wrapper:** is the module that prepares the response message that will be returned to the application with the result of the metric.

The REST Interface for computing a metric is the following: [http://demo.liquidpub.org/reseval_v3/resources/compute_metric/object/objectType/metric/metricOption/source/sourceOption/reloadOption/publist/][^2] where the input parameters represent:

- **Object:** name of the object to be evaluated.
- **ObjectType:** type of the entity that is evaluated (author or contribution).
- **Metric:** name of the metric that will be computed.
- **MetricOption:** this represents the algorithm that will be used to compute the metric.

[^1]: [http://reseval.org](http://reseval.org)

[^2]: [http://reseval.org](http://reseval.org)
D5.1v2 Design of the Liquid Publications Integrated Platform

An example of a valid call to the REST interface is: [http://demo.liquidpub.org/reseval_v3/resources/all_metrics/maurizio%20marchese/author/Google%20Scholar/&as_subj=eng/no/no](http://demo.liquidpub.org/reseval_v3/resources/all_metrics/maurizio%20marchese/author/Google%20Scholar/&as_subj=eng/no/no). This link computes the h-index of Maurizio Marchese using Google Scholar as a data source.

**Source:** is the link to the repository from where the input parameters have to be retrieved.

**SourceOption:** is the filter to be used when the information is retrieved from the specified source.

**Reload:** is the flag that indicates that recent information is requested from the source instead of cached one.

**Publist:** is the parameter telling if user wants to load publication list or not, "yes" to load and "no" to not.

**Data layer:** this layer is used to get the data needed in order to compute a specific metric. There are different sources where the tool can get the data: internal sources, such as a local database, or external sources found in the web. For these external sources an adapter must be implemented that is able to retrieve the information for each source. Furthermore, the adapter must be registered in the infrastructure modules (described in Section 6) in order to be available for use. The architecture includes the cache mechanism to store the information retrieved from underlying sources. The caching system is clearly motivated by performance reasons, since we are dealing with a web application that often queries external servers and, therefore, loses much time due to network’s overhead.
4.2.1 Implementation


5 Process lifecycle services

This section reports on LiquidPub Management System (LPMaSys) and Gelee System. These are tools developed in LiquidPub and dealing with the management of research and dissemination processes. In particular, LPMaSys deals with the specification and the automatic execution of the processes concerned with the creation, dissemination, and evaluation of research work. LPMaSys deals specifically with formal and repeatable processes, such as a process of a conference (submitting articles, reviewing, submitting camera ready) or of a journal submission, and manages the progression of such processes. The novelty of LPMaSys is that it automatically adapts the execution stage’s user interface according to the modification of the process specification, if any. Gelee is instead a tool for lightweight lifecycle management. It allows users to define the lifecycle of any artifact identifiable by a URI (e.g., SKOs such as deliverables, papers), to monitor its progress, and to automatically execute actions on resources upon the resource entering specific lifecycle states. A typical example application is to define and monitor a quality plan for the artifacts. Gelee does not include an engine (the state is advanced manually) and the lifecycle can be defined and evolved “as you go”, at runtime. It merely provides a guideline for execution that can be followed or even disregarded. As such, it is intended to be used in more informal and liquid processes, such as the collaborative writing and reviewing of a deliverable.

5.1 LPMaSys

LiquidPub Management System (LPMaSys) provides users with straightforward means for the specification and the automatic execution of the processes concerned with the creation, dissemination, and evaluation of research work. In other words, the aim is a user friendly system for managing research work.

The specification is done by means of charters. A Liquid Publication charter outlines the conditions under which a certain process that is concerned with the creation, dissemination and evaluation of research work is organized. A Liquid Publication charter (from now on referred to as charter) is defined by a set of rules. For instance, rules could address who is allowed to create, modify, assess scientific knowledge. Rules may also specify deadlines for actions, the accepted order of actions, and so on. We refer to these rules as the specification of the charter. Examples of
charters are creating and running conferences, managing projects, writing papers collaboratively, compiling books, and so on.

In addition to the specification of charters, the management system should also provide the automatic creation and update of a web-based user interface for the related researchers (or personnel). Existing examples of this are current conference management systems, such as ConfMaster and EasyChair. However, while existing conference management systems are hardcoded and the interface requires a manual update every time a change occurs, the Liquid Publication management system proposes an automated generation and update to the web-based user interface, eliminating the need for manual modifications.

To achieve the objectives above, the user needs to specify the charter and enrich it with information about user interfaces as shown in Figure 20. Ideally, we aim at providing a straightforward user interface that would allow the user to produce the specification through a simple drag-and-drop. Hence, we say the charter can be specified as a transition graph and an additional mapping file is needed to map the states of the transition graph to a set of 'views'. This is because each peer playing a specific role in the charter model should see a different set of views, based on its current state in the model. Naturally, a library of 'views' should also be provided to allow the user to choose from.

To implement the above, we make the following choices:

- charters are implemented as Electronic Institutions (EI) [15]. Islander editor, one of the tools of Electronic Institutions Development Environment (EIDE) [1], already provides an easy to use user interface for the specification of process models through a simple drag-and-drop. Once charter model is specified, it can be executed using AMELI, also provided by EIDE.

- the library of views will essentially be a library of widgets

- the file mapping between states of the charter and a set of views is specified in XML

Figure 21 shows the main elements of the LP Management system architecture:

**Electronic Institution.** Is the center of the system which provides the infrastructure for defining and executing charters.

**Web server.** This is the main link between a human user and the EI.
Staff agents. Institutional agents in charge different aspects related to the well-functioning of charters.

I-Agents. Agents that represent human users (charter personnel) in the EI. They also act as web servers (servlet) and connect to EI by interacting with the governors through institutional events.

Mapping info. Defines a map among charter states, charter user roles and views. That information is used by I-agents to update properly the web-based user interface.

Client application. We have adopted an approach based on two web technologies: Java Server Pages (JSP) and Asyncronous JavaScript And XML (AJAX). JSP technology provides a simplified, fast way to create dynamic web content by merging HTML with JAVA. In contrast, AJAX is considered a web design technique that provides a high level of interactivity, avoiding the undesirable reloading of the web pages after each human user action. Hence, the client application only needs a relatively recent web browser and therefore it is not necessary to install special software.

The I-Agent, acting as a servlet, receives information requests from the web browser of the user. I-Agent replies to the web browser according to the information containing in the mapping file and the relevant changes that have been produced in the EI.

5.1.1 Implementation

The code of the LPMaSys can be found at http://project.liquidpub.org/lpms/docs/lpmasys_demo.zip As the code becomes stable, it will be available on the LiquidPub svn.
5.2 Gelee

Gelee is a tool for modeling and managing lifecycle of web artifacts. The main principle of Gelee is to keep the lifecycle model simple, with only a few elements and constructs. This simplicity extends to the underlying architecture thus enabling a lightweight infrastructure. We first define the overall architecture, and then discuss two among the most important aspects: action-resource interaction and lifecycle widgets. Although they are “embedded” in the architecture section we draw the reader’s attention to it as they correspond to key decisions in terms of keeping the model simple, and in terms of usability and reusability.

The main concepts behind this work are drawn in [3].

**Architecture.** The Gelee architecture is simple, especially due to the fact that there is no analogous of a workflow engine that progresses the flow from step to step. In essence, the system supports design and monitoring as well as invocation of actions that, from the core system perspective, are black boxes and are embedded into resource type-specific plug-ins that can be added as needed. As the primary goal of Gelee is to manage online resources and to have a system that is simple and usable, it was natural to provide lifecycle management as a service, and therefore hosted. Figure 22 depicts the high-level architecture, composed, essentially, of three layers: the data tier, the kernel and the user interface.

The functionalities offered by the Gelee Core are provided by four sub-components:

- **Design-time manager:** It manages the lifecycle modeling and templating. Lifecycle models and templates are versioned and can be shared among groups of users.
- **Runtime manager:** The runtime manager manages the progression of lifecycle instances, performs change management and derives the action executions to the basic services layer.

Monitor: The monitor subcomponent allows getting the trace and status of lifecycle executions.

Configuration manager: It manages the user base, access control of lifecycles, and the configurations of the lifecycle management component.

The following services provided by the Gelee Core are exposed through a REST interface:

- Creating, updating, reading, deleting, searching lifecycle models and lifecycle model templates (CRUD + search). Note that by updating lifecycle models and templates we provide versioning.

- Creating, updating, reading, deleting, searching lifecycle instances (CRUD + search).

- Migrating lifecycle instances. This is performed by updating the current state of the lifecycle instance with a new state in the target lifecycle model. A state in this context refers to a set of properties including a reference to a phase.

- Performing a transition. This is performed by updating the current state of the lifecycle instance with a new state in the same lifecycle model.

- Getting a lifecycle instance trace. This service allows getting the execution log.

- Creating, updating, reading, deleting, searching users. It allows managing users in the lifecycle management workspace.

- Creating, updating, reading, deleting, searching groups of users (CRUD + search). It allows users to organize themselves in groups to share models and templates.

The interfacing between Gelee and a specific resource occurs through plug-ins or adapters offered by the infrastructure services (Karaku) described in Section 6. Developers can create adapters for any kind of resource, and implement actions that support a given functionality. The action implementation may correspond to an existing action type defined earlier for other resource types (e.g., send for review) or it can be a new action type that does not exist in Gelee. In both cases, the adapter needs to register the new action implementation in the Karaku, to make Gelee
are aware that there is an action implementation for a specific resource type has been added, or that a completely new action type is introduced. The registration also includes information that Gelee needs for invoking the action.

The action definition is standard and includes information about i) the action type, which defines how to access the action; ii) parameters, and the time at which their values have to be associated; and iii) general metadata. This definition allows Gelee to handle all actions in a standardized fashion.

Karaku also provides support for lifecycle modeling and execution through the use of the resources and actions definition and managing access rights.

**Lifecycle Widgets.** Gelee offers UI-level integration through resource type-specific widgets and web interfaces. Web UI are provided for the lifecycle designers Figure 24 and for the monitors. Both offer an AJAX-style interface, easy and immediate to use.

Execution of lifecycles is instead different, as ideally it is integrated with the resource, for example, it is shown in the same web page, or as a browser plug-in. For this aspect we use widgets. Widgets are components ready to be integrated with web applications or even desktops. Through widgets, users see the lifecycle and the resource they manage side by side, as shown in Figure 25.

The integration between lifecycle management and the resource, as shown in Figure 25, is done by combining execution features provided by the lifecycle manager interface, and resource-specific information provided by the resource manager. The latter offers the interface by which
we can render any resource in a transparent way.

Widgets are also subject to visibility attributes. Attributes like access rules are automatically auto-discovered from the lifecycle definition. Thus, a user interacting with a widget could be requested for authentication or not based on the visibility attributes, and also, different users could have different views of the same lifecycle (i.e., managers, resource owners, and stakeholders in general).

Users may have more specific needs beyond the services we provide through our widgets. Therefore, we allow them extending from the API to develop their own widgets or web components (e.g., a Facebook application). Moreover, because of the added value of composing the services from different source, we prepared our widgets to put in pipes (e.g., Yahoo Pipes). For example, users could feed our widgets with Google Docs feeds listing documents, and use that list to reflect the lifecycle of those documents.

5.2.1 Implementation

The code of the Gelee is available at https://code.launchpad.net/gelee. The information for Gelee developers is available at http://project.liquidpub.org/gelee/developers.html In particular, the API specification is available at https://dev.liquidpub.org/svn/holms/trunk/docs/gelee_reference.pdf.
6 Infrastructure services

Scientific contributions appear on the Internet in different places: repositories maintained by publishers (SpringerLink, IEEE Explorer, ACM Digital Library), authors’ personal web pages, other online archives, other sites that store research-related content, such as blogs, projects, announcements, mailing-lists, scientific workflows, etc. There is no complete repository we can rely on, rather, we rely on different and heterogeneous data sources on the Internet. This is particularly important as blogs, data, experiments are also considered as contributions in the LiquidPub world.

The infrastructure services represented by Karaku aim at providing a uniform access for such external data sources, plugging in them as one of the inputs to the knowledge bus, thereby making external content accessible to other modules of the LiquidPub platform. In other words, Karaku provides a bridge between non-LiquidPub world and the LiquidPub world.

Items in external data sources are represented resources. A resource is any artifact that has a URI and is accessible over the Web. This notion is very general and captures the requirement of supporting any arbitrary information such as simple web pages, online documents, web services, feeds, etc.

Figure 26 shows the overall architecture of Karaku, including the following functional components:

i. **Updater**: provides capabilities to pull in metadata from external sources, in order to populate and keep updated the locally cached metadata, available in the SKO repository in the knowledge bus.

ii. **Resman**: extracts metadata from external resources using dedicated adapters.

[http://project.liquidpub.org/karaku](http://project.liquidpub.org/karaku)
iii. **Adapter registry**: stores adapters for accessing external resources.

In the following sections we describe Updater and Resman components of Karaku in detail and discuss the role of adapters.

### 6.1 Updater: Crawling the Web for Scientific Resources

Here we describe how updates from the sources are carried out. The Updater (crawler) is the component responsible for fetching data from different Internet sources on the Internet, parsing their content, rendering it in a format suitable for the knowledge bus, and storing these data in the SKO metadata repository.

Crawling process consist of three steps, which must occur in sequential order. However, since each target source for update procedure is independent from all other sources, updates from different sources can be performed in parallel. The three steps, as depicted in Figure 27, are the following:

- **Crawling**: the actual fetching of data from publicly accessible sources on the Internet. More concretely, the crawling involves connecting to a remote source via HTTP connection, and issue a request for retrieving available data. Retrieved data is then passed to adapter-parser subcomponent for next stage in updating process: for the sake of performance, and depending on the actual size of content to be downloaded, the crawler may take advantage of a simple caching mechanism.

- **Parsing**: during this step, data coming from external sources are converted into internal metadata format which allows for them to be easily passed to knowledge bus and be inserted into the SKO repository.

- **Storing**: this step takes care of the saving the data into the repository via the REST API provided by the knowledge bus.

The whole process is still in prototype phase: the various steps of the process have been all implemented, however it’s still missing “glue” code, i.e., a framework for scheduling execution of

---

**Note**: Unfortunately, there’s no such a thing as a directory of publicly available scientific documents metadata resource; although, some well-known “resources-gatherer” exists, like, for instance, the Registry of Open Access Repositories at [http://roar.eprints.org/](http://roar.eprints.org/)
aforementioned steps. However, this may prove to be not necessary, since the various steps may be still executed (as they are now) as independent processes, with their own data storages. They may for instance be connected in an asynchronous, loosely coupled manner, following Pipeline software design pattern\(^{36}\) we are going to investigate this and, according to our result, implement corresponding solution in the next release also.

### 6.2 Resman: Accessing a Web of Heterogeneous Resources

The Resman\(^{37}\) component provides us with abstractions for modeling the vast amount of resources the Web offers and allows us to take into account also the technical aspects involved in accessing the resources. Indeed, the huge variety of resources that can be provide useful input to LiquidPub platform are managed by different service providers that may or may not have an API: Google Docs, wiki, SpringerLink, Google Scholar, etc.

In the scenario depicted by resources provided on the Internet, it is not trivial to provide abstractions, given the heterogeneity in the resource providers (aka resource managers). For instance, in Figure 28 we illustrate such heterogeneity showing some examples of currently available and relevant scientific domain services, all of them providing access to their content (the scientific contributions) via different APIs/protocols. In principle, there are no limitations for the kind of resource managers Resman can support, as long as they provide services for resources. Therefore, we consider the service or action, which describes the services provided by resource managers and that allow us to operate with the resources (e.g., to share, publish or search documents, or more complex actions such as crawling a web site such as Google Scholar or CiteSeer for scientific metadata).

As depicted in Figure 29 the Resman component is composed of two main modules: the resource space management and the access management modules. These two modules run the machinery for providing homogeneous access to resources and transparent extensibility in terms of multiple resource managers’ support.

\(^{36}\)See [http://en.wikipedia.org/wiki/Pipeline_(software)](http://en.wikipedia.org/wiki/Pipeline_(software)) for more detailed description

\(^{37}\)More information on Resman is available in [4]
6.2.1 Resource Space Management

The resource space management module deals with the resource managers (repositories, search engines, blogs, etc.) available to the upper layers. Thus, this module allows us to register resource managers and the related resources and actions. It also manages the mapping between these constructs and the abstractions of resource types, action types and resource manager types. The link between the actual resource managers and the abstractions we provide is performed through adapters [5], described in Section 6.3.

The registration or resource managers is performed using a specialized service that enables resource manager providers and programmers to populate a registry of resource managers and to make them available to upper layers. Note that it is also possible to define and register composite resources by combining actions from different resources into a complex resource type. This is particularly interesting for applications in which the conceptual resource can be composed of multiple low level artifacts (e.g., a virtual folder that contains elements which are references to Google docs, Zoho, or MS Word documents stored in an SVN). From the perspective of a client using the module, this acts as a “dictionary” that offers information about the resources, actions, resource managers and their abstractions, available in the registry.

6.2.2 Access Management

The access management module allows interfacing with different repositories and libraries through a standard interface. This module is able to operate on resources of the same type (e.g. documents) with the same set of operations (e.g., create, delete, share) using the resource-type level of abstraction. In other words, this module allows executing actions on the resource managers registered from the resource space management module. Note that this is different from executing opera-
D5.1v2 Design of the Liquid Publications Integrated Platform

Figure 30: Registering adapters and executing operations

actions directly on the adapters where one can perform operations only on actual resources, and so the set of operations available are specific to those specific resources. For example, consider executing the operation “sharing” over a set of resources provided by different resource managers. The actual implementation of the action “share” will likely have a different signature in each adapter. The access module abstracts these differences allowing clients to operate at the action type level of abstraction, which in this example will be the “share” action type.

As illustrated in Figure 30, the interaction with the resource managers (the services providers) is performed through adapters. The Access management module interfaces with the adapters and exposes their functionalities to the upper layers. The added value here is the possibility of working with different resources managers at a different level of abstraction; i.e., clients of this module do not need to know the details of the actual resource managers, indeed, they do not need to know which resource manager is providing a given service. The access management module, according to the specification of the resource types, manages this interaction.

6.3 The Role of Adapters

The approach we follow to guarantee extensibility, interoperability and maintainability is to provide a set of core modules that can manage the adapters and access to resource managers through these adapters. Each adapter provides a definition of the resources and operations supported and, if necessary, the implementation of the logic for accessing the resource managers (e.g., in case no API is provided). Figure 30 illustrates how the interaction with the adapters is performed.

Adapters are developed within LiquidPub or by third parties and made available to the upper layers through the registry of adapters. Note that the approach we take here allow us to extend the services we provide access to without introducing changes into the platform. This is one of the key aspects of the flexibility provided by the architecture.

To illustrate the above, consider the procedure for registering adapters. This procedure involves the adapter provider (the one that hosts the adapter) registering the adapter definition using the service provided by the Resman access layer for that purpose. This definition involves the
mapping between the existing resource types (e.g., documents, pictures, etc) and action types (e.g., share, export, update, etc.) and the implementations provided by the adapter (and offered by the correspondent resource manager). This definition is then processed by Resman, which registers these implementations. This is possible since resources types and action types have unique identifiers that allow reusing their definitions. However, nothing prevents an adapter to register new resource types and action types. In this case, these new definitions become available to other potential implementations.

As a result of the registration procedure, a new resource manager becomes available in the platform, implementing a set of actions and offering support for resources, sharing common functionalities with other resource managers semantically equivalent at given abstraction level.

6.3.1 Example

To make this more concrete, assume we want create a new resource type to operate, with the same actions, on documents subject to version control. Using Resman, we have to perform a call to the REST API as shown in the listing 1:

```
POST http://project.liquidpub.org/resman/resource-type.xml

Body

<resourcetype>
  <name> Versioned Document </name>
  <description> Resource type for versioned documents </description>
  <user-ref> http://project.liquidpub.org/gelee/api/user/8901 </user-ref>
  <creation-date> 2009–12–02 </creation-date>
  <actiontype-list>
    <link href='http://project.liquidpub.org/resman/action-type/145' value='Check out'/>
    <link href='http://project.liquidpub.org/resman/action-type/141' value='Commit'/>
    <link href='http://project.liquidpub.org/resman/action-type/144' value='Roll back' />
    ...
  </actiontype-list>
</resourcetype>
```

Table 1: Posting a resource type in XML format

The call in the listing 1 returns the URI to the newly created resource type. In the definition, we reference the action types that will be allowed by all the “Versioned Documents”. Then, clients can get the resource type definition by asking Resman about the resource type identified by the URI as shown in listing 2:

```
...
In this extensible approach, the resource manager and the concept of resource type collectively support a flexible binding approach that can range from static to dynamic binding to both adapters and (for services using the RSMS) to resources.

6.4 Implementation

The source code of the Karaku is available (using lp-guest and lp-password credentials) at https://dev.liquidpub.org/svn/liquidpub/prototype/lpbase/. The API specification is available at http://docs.google.com/View?id=dff3b97_3gfdm7c. More information on the implementation of Karaku is provided in D5.2v1 [13].

7 Testing

Each LiquidPub project component has a set of automated test cases, grouped in test suites, that can be executed in batch mode. The usual way of doing this is by defining a special target (called “test” in most cases) which takes care of running the various tests in build tool configuration file. A project developer should define unit tests for each compilation unit she introduced, while functional test should grant that desired feature is present. Finally, integration tests will make sure modules interaction is working as planned. Each developers is responsible for setting testing environments for its own tests, so this implies that every test must be self-contained.

LiquidPub project adopts continuous integration as integrated testing technology. We rely on Hudson (http://hudson-ci.org/), a dedicated server application, for supporting continuous integration in our project. Each LiquidPub component possess (at least) one configuration (“project” in Hudson jargon) which continuously iterates the steps of the build cycle. It also takes care of displaying the results, and keeping trace of builds outcomes.

LiquidPub project continuous integration tool is available at the following address: https://dev.liquidpub.org/hudson. For security reasons, access to this tool has been restricted to project members only, so unlike source code this is not to be seen by everyone.

38 Update the code from project repository - build the application - run the tests.
References


[18] KRAPIVIN, M., MARCHESE, M., AND CASATI, F. Exploring and Understanding Citation-based Scientific Metrics.


