# SlickChair

Olivier Blanvillain
École Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland
olivier.blanvillain@epfl.ch

## ABSTRACT

Ut in dolor et magna tincidunt mattis. Proin id pulvinar arcu. Donec ac turpis consectetur, dignissim eros at, mollis orci. Nunc sed tincidunt justo, eu dapibus risus. Vestibulum nisi mi, tempus nec cursus at, accumsan vitae metus. Cras mattis, velit ut convallis lacinia, metus enim rutrum sapien, quis egestas ante ipsum ut lacus. Aliquam erat volutpat. Mauris vitae commodo nisi. Nunc iaculis, enim vulputate cursus interdum, sapien libero sodales diam, viverra molestie diam tellus eu felis. Quisque gravida porttitor vulputate. Duis nec neque facilisis, porttitor ante id, molestie sem. Vivamus pellentesque venenatis est, ut consequat arcu tempus a. Phasellus ullamcorper nunc vel rhoncus suscipit. Curabitur commodo ornare accumsan. Mauris vitae lorem arcu. Mauris vel turpis mi. Fusce faucibus congue ante, eu gravida sem. Cum sociis natoque penatibus et magnis dis.

## 1. INTRODUCTION

Peer reviewing is a central process in the organisation of scientific conferences. It involves multiple interactions between it's participants: *authors* share their work with the *program committee* which is then in charge of reviewing these submissions. A *program chair* is designated to coordinate the process and decide on final submission acceptance. While this peer review process could be implemented by simply exchanging emails between participants, manually gathering submissions, assigning them to program committee members, aggregating the reviews and finally notifying the authors requires substantial efforts. The use of a dedicated software, called conference management system, can greatly simplifies this process.

Over the last few years, numerous web based conference management system have been developed. A recent comparative study [10] shows that Easychair [4] is by large the most popular platform, having been used in about 68% of conferences organized with online conference management system. The following four systems in term of number of organized conferences are EDAS [5] with 8.5%, Open Conference Systems [9] with 6%, START V2 [13] with 5.7% and ConfTool [1] with 5.3%. However, out of these five systems, only Open Conference Systems is open-source. Easychair and ConfTool offer free licenses of their restricted versions and START V2 and EDAS are only available as a commercial product.

The importance and confidentiality of the data manipulated by a conference management system imply that security a major concern when working with such system. Closed source solutions are often only available as hosted services, therefore requiring conference organisers to trust the company behind the product not only for the quality of the code, but also for the robustness of the infrastructure and the respect of data privacy. The open-source solutions we considered where not providing satisfactory levels of security. For example, Open Conference Systems sends a copy of the user passwords by email as plain text once the registration is completed. HotCRP [8], another open-source conference management system, has a similar flaw: it sends login links to users with the password as part of the url.

We present SlickChair, an open-source conference management system written in Scala. Build with the Play framework and the Slick database access library, SlickChair provides a highly flexible and extensible solution to manage peer review processes. Our contributions are in particular:

- The plan

## 2. OVERVIEW OF SLICKCHAIR

In this section gives an overview of functionalities of SlickChair. We first discuss how users.. (login, actors, phases (workflow))

### 2.1 User authentication

The first contact between a system and its users is often via an authentication interface. Most conference management systems require users to create new accounts, which usually implies filling a form, receiving a validation email and memorizing a new password. Recent web browser implement mechanisms to reduced the pain of filling form and memorizing passwords, but these solutions are usually limited to the use of a single computer.

In SlickChair, we address this common problem by supporting three types of authentication mechanisms:

- Authentication via Facebook account
- Authentication via Google account
- Authentication via email address

The implementation of Facebook and Google login uses the OAuth 2.0 protocol [7], and is provided by the SecureSocial authentication module for Play Framework [12].

In addition to associate each visitor with a unique identity, an objective of authentication is to get a valid email address to contact the user. This way, we avoid any chances of having typo in the email. When users login with Facebook or Google account, we trust the email address obtained via the OAuth protocol, given that both networks required their users to confirm their email address when they created their account. In the case of login via email, the process is obviously more complex. After providing his email address, a new user has to open his email client follow a validation link. From here, the user is asked his first name, last name, and a new password (of minimum 8 characters) before completing the registration. Passwords are hashed using the bcrypt algorithm [11] and stored in the database. SlickChair also gives its users the usual options to change their password or recover their account in case of forgotten password. Although authentication via email address add complexity to the system, it is appreciated by users that have separated email accounts for their personal and professional life.

In SlickChair, we identify each user by a single email address. Some other conference management systems provide the ability to link multiple email addresses to a single identity and to multiple merge accounts into one. We believe that such functionality can sometimes be confusing, and adds a lot of complexity to the system. As a side note, this design makes it possible for a user to close the Facebook or Google account he used to login, and still claim his identity by going thought the process of logging in using the corresponding email address.

## 2.2 SlickChair interfaces

Building SlickChair, our focused was on creating a flexible and extensible system rather than offering customization options via configuration. As a system becomes more complete, it's complexity is likely to go up, and maintaining and extending the system becomes harder. SlickChair provides the essential[1] components to run an online peer-review process, listed below as the user interfaces of available for each user role:

- Author
  - Make new submissions
  - Edit submissions
  - See reviews
- Program Committee Member
  - Bid on submissions
  - Write reviews
  - Write comments
- Program Chair
  - Assign submissions for review
  - Decide on submissions acceptance
  - Change user roles
  - Send emails and change conference phases

[1]The authors of [3] identified nine *typical* functionalities offered by conference management systems to run online peer-review processes, which roughly correspond to the functionalities provided by SlickChair interfaces.

Most interfaces are very straightforward and will not be discussed here for brevity. The *change user roles* interface allows a program chair to design certain users as being co-chairs or program committee members. While it is also possible to define all program committee members and chairs in a configuration file before setting up a SlickChair instance, using this *change user roles* interface might be preferred to allow each user to chose his favourite login method. This accounts for the typical situation of users that forward messages send to their professional email addresses to other services such as Gmail.

The *Send emails and change conference phases* interface is related to the conference workflow, discussed in the following subsection.

## 2.3 Workflow

In order to coordinate the overall conference peer-review process, we organised the course of this process as a chronological sequence on phases. Each phase corresponds to the set of interfaces enabled during the during the particular time frame of the phase. We identified the following seven phases that may correspond to the workflow of a small conference: *Setup, Submission, Bidding, Assignment, Review, Notification, Finished.*

In addition to the configuration of enabled interfaces, each phase is defined with a function to generate emails, and function to generate an optional warning:

```
case class Phase(
  configuration: Configuration,
  emails: Database => List[Email],
  warning: Database => Option[String]
)
```

SlickChair uses the `emails` function before transitioning to a given phase to help the program chair in the redaction of notification emails, by suggesting appropriate recipients and giving a template for the body of the message. If a warning is returned by the second function, it will be display to the program chair before he confirms the change of phase and the sending of notifications.

Conferences of different sizes and topics might want to use different workflows. In the current stage of the project, such configuration has be done has source code level. As an example, we will show the changes needed to add a new component to the conference workflow. Suppose that some program committee members are careless about their reviewing responsibilities and do not sent respect the delays set by the program chair. The program chair might to send a reminder to the program committee members that have not yet completed their reviews. This could be implemented by adding the following phase between *Review* and *Notification*:

```
Phase(
  Configuration("Review Reminder",
    pcmemberCanReview=true,
      pcmemberCanComment=true,
      chairCanDecideOnAcceptance=true),
    { db => Email(
      lateReviewerEmails(db),
      "Reminder: review deadline",
      "Dear Program Committee Member, ...")},
    noWarning
  )
```

Where `noWarning` a dummy function that never returns any warning, and `lateReviewerEmails` is a function returning the email addresses of all late reviewers. In order to compute the late reviewers, we need to use three tables of the database: the `assignments` and `reviews` tables are relations between program committee members and submissions, and the `persons` table contains personal information of SlickChair users.

```
def lateReviewerEmails(db: Database) = {
  val assignmentPairs = db.assignments
    .filter(_.value)
    .map(a => (a.personId, a.paperId))
  val reviewsPairs = db.reviews
    .map(r => (r.personId, r.paperId))
  (assignmentPairs diff reviewsPairs)
    .join(db.persons).on(_._1 is _.id)
    .map(_._2.email).list(db.s)
}
```

This function illustrates the use of the Slick database query library. Except for `.join().on()` and `.list(db.s)`, the code is would be identical if the members of `db` where Sets from the Scala collections. In reality, the entier body of this function will be compiled into a SQL query. The `.join().on()` allows to join two tables on certain columns, and returns pairs of matching rows. `.list()` wraps up the query definition and returns the result of execution as a list of Scala objects.

## 3. PAPER-REVIEWER ASSIGNMENT

Among his responsibilities, the assignment of submissions to program committee members can be a complex task. To be fair to all authors, submissions usually receive the same number of reviews, and this work has to be well distributed among program committee members so that no one overloaded. Moreover, program committee members might have conflicts of interests with certain submissions and different levels of knowledge depending on the topics. These constraints add up for

It's NP-Hard :( [6] `http://www.cs.uky.edu/~goldsmit/papers/GodlsmithSloanPaperAssignment.pdf`

## 4. DATA MODEL

- Logs where a requirement
- Inspired from Datomic [2] data model
- Re-implemented some of Datomic's functionalities as a layer on top of Slick/AnySQL
- Functionality and implementations are detailed in this section.

datastorage database do not need transactions for most operations still the go to product because of the abstraction layer, the backup capabilities and the power of the query engine.

- functional, immutable database
- database as a value, queries as a function
- timestamp implementation
- allows for concurrent read/write
- single thread alternative for

## 5. FUTURE WORK:

- Macros
- Scala.js

## 6. CONCLUSION

- ...

## 7. REFERENCES

[1] ConfTool Website. Available at: `http://www.conftool.net`.

[2] Datomic. Available at: `http://www.datomic.com/`.

[3] N. Di Mauro, T. M. A. Basile, and S. Ferilli. GRAPE: An expert review assignment component for scientific conference management systems. In *Proceedings of the 18th International Conference on Innovations in Applied Artificial Intelligence*, 2005.

[4] Easy Chair Website. Available at: `http://edas.info/doc`.

[5] EDAS Website. Available at: `http://www.easychair.org`.

[6] N. Garg, T. Kavitha, A. Kumar, K. Mehlhorn, and J. Mestre. Assigning papers to referees, 2010.

[7] D. Hardt. The OAuth 2.0 authorization framework. RFC 6749, RFC Editor, 2012.

[8] HotCRP Website. Available at: `http://read.seas.harvard.edu/~kohler/hotcrp/`.

[9] Open Conference System Website. Available at: `http://pkp.sfu.ca/?q=ocs`.

[10] L. Parra, S. Sendra, S. Ficarelli, and J. Lloret. Comparison of online platforms for the review process of conference papers. In *The Fifth International Conference on Creative Content Technologies*, 2013.

[11] N. Provos and D. Mazières. A future-adaptive password scheme. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '99, 1999.

[12] Secure Social. Available at: `http://securesocial.ws/`.

[13] START V2 Website. Available at: `http://www.softconf.com/`.