The SlickChair Conference Management System

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

ABSTRACT

Ut[ˆname] 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 process could be implemented by exchanging emails, manually gathering submissions, assigning them to the program committee, aggregating the reviews and finally notifying the authors would require substantial efforts. The use of a dedicated software, called conference management system, can greatly simplifies this process.

Over the last few years, numerous web conference management system have been developed. A recent comparative study [14] shows that Easychair [5] 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 conferences are EDAS [6] with 8.5%, Open Conference Systems [12] with 6%, START V2 [16] 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 for restricted versions and START V2 and EDAS are only commercial available.

The importance and confidentiality of the data manipulated by a conference management system implies that security a major concern. Closed source solutions are mainly available as hosted services, therefore requiring conference organisers to trust quality of the code, 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 user passwords by email in plain text when the registration is completed. HotCRP [10] has a similar flaw: it sends login links to users with passwords as part of the URLs. As none of the available solutions meet our requirements, we decided to build our own.

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. This report makes the following contributions:

- We briefly introduce SlickChair and present the aspects where this new system provides a gain compared already established conference management systems such as EasyChair. In particular, SlickChair offers the ability to authenticate using a Facebook or Google account, thus replacing the tedious password creation and email validation process.
- We present a new API to manipulate versioned date on an append only database. Build on top of the Slick, our API combine the power of Scala type-checking with the benefits of using an immutable database.
- We show how we implemented SlickChair's automatic paper-reviewer assignment algorithm. Unfortunately, accounting for both program committee member preferences and conflicts of interest leads to an NP-Complete problem. While aiming for the optimal assignment is not possible, we opted for a very simple heuristic algorithm implemented with the help of the OscaR operational research library.

2. OVERVIEW OF SLICKCHAIR

We begin our discussion with a short overview of SlickChair. We first present the three supported authentication mechanisms, Facebook, Google and email address, followed by an outline of the different user interfaces. The section concludes with a description of the default peer-review workflow and an example of customisation.

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 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 limited to the use of a single computer.

In SlickChair, we address this 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 [9], and is provided by the SecureSocial authentication module for the Play framework [15].

In addition to associating each visitor to a unique identity, authentication allows to get a valid email address contact users. When users login with Facebook or Google account, we trust the email address returned by the OAuth protocol, given users of these networks are required to confirm their email address. 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, he 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 and stored in the database. SlickChair also offers options to change password or recover an account in case of forgotten password. Although authentication via email address adds complexity to the system, it is useful to users that use different 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 this functionality can sometimes be confusing, and adds unnecessary 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 claim his identity by going thought the process of logging in using the corresponding email address.

2.2 User interfaces

Building SlickChair, our focused was on creating a simple and extensible system rather than offering a lot of configuration options. As a system becomes more complete, it's complexity is likely to go up, and maintenance and extensions becomes harder. SlickChair provides the essential components[1] to run an online peer review process, listed below as interfaces 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

The *Change user roles* interface allows a program chair to designate certain users as co-chairs or members of the program committee. While it is also possible to define all program committee members and chairs in advance in a configuration file, using the *Change user roles* interface has the advantage 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 another service such as Gmail.

The *Send emails and change conference phases* interface is related to the conference workflow, discussed in the following subsection. *Assign submissions for review* is discussed in detail section 4. Other interfaces are straightforward and wont be described for brevity.

2.3 Workflow

In order to coordinate the overall conference peer review process, we organised it as a chronological sequence on phases. Each phase has a set of interfaces enabled during the time frame allocated to this phase. We identified the following seven phases that may correspond to the workflow of a small conference: *Setup, Submission, Bidding, Assignment, Review, Notification and 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 recipients, subject and content of messages to be sent when transitioning to this phase. If a warning is returned by the second function, it will be display to the program chair before he confirms the change of phase.

[1]The authors of [4] identified nine *typical* functionalities of conference management systems which roughly correspond to the functionalities provided by SlickChair interfaces.

Conferences of different sizes and topics might want to use different workflows. In the current stage of the project, such configuration has be done at 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 respect the delays set by the program chair. The program chair might want to send a reminder to the program committee members that have not yet completed their reviews. This could be implemented by adding a *Review Reminder*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 returns no warning, and `lateReviewerEmails` is a function returning the email addresses of all late reviewers. In order to obtain this information, 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. Aside from `.join().on()` and `.list(db.s)`, the code would be identical if manipulating sets from the Scala collections. In reality, the entire body of this function is compiled into a SQL query. The `.join().on()` construct allows to join two tables on certain columns. The `.list(db.s)` wraps up the query definition and returns the result of execution as a list of Scala objects.

3. DATA MODEL

An original requirement of SlickChair was the ability to log all the actions and events that occurred in the system. To do so, a traditional approach would be to use the logging functionalities build into the Play framework. Concretely, this would corresponds to appending a timestamp and a messages to a text file whenever an interesting event occurs. We took an alternative approach of in memorizing every change at the level of the database.

The idea of append only databases is not new, but it is becoming more and more relevant as the prices of storage go down. The Datomic database [2], which recently entered the market, is build around this idea of never altering or deleting data. Instead, changes are made by adding new version of the existing data or marking it as being invalid. Datomic is a proprietary software and does not fit well the open-source platform SlickChair is build upon. In order to suite our needs, we build a small layer on top of Slick to manipulate versioned data. The semantic of the API presented in this section are inspired by Datomic's Clojure API [3].

3.1 Database as a value

One of the core concept of functional programming the manipulation of immutable data. But how could a database, this always mutating, shared entity, be seen as immutable? We define a `Database` to be a view of the data on at a particular time `date`:

```
case class Database(
    val date: DateTime, val s: Session) {
  val persons = table[Person]
  val personRoles = table[PersonRole]
  val papers = table[Paper]
  val paperAuthors = table[PaperAuthor]
  val paperDecisions = table[PaperDecision]
  val comments = table[Comment]
  val reviews = table[Review]
  val files = table[File]
  val emails = table[Email]
  val bids = table[Bid]
  val assignments = table[Assignment]
  val configurations = table[Configuration]
}
```

Manipulations of `Database` objects, such as the function define in subsection 2.3, will see the world as it was at time `date`. Hence, calling this function multiple times with the same argument will always return the same result: the query is a pure function and the `Database` is a value.

A `Connection` allows to retrieve the current value of the database, and insert new elements in the database. The `insert` method returns two `Database` objects, the value just before and just after the insertion.

```
case class Connection(s: Session) {
  def currentDatabase(): Database
  def insert(x: List[Model[_]]): (Database,
                                  Database)
}
```

The last element of our API is the `Model` trait, which enforces that elements stored in tables are case classes with a `metadata` field providing the appropriate information:

```
case class Id[M](value: UUID)

type Meta[M] = (Id[M], DateTime, String)

trait Model[M] {
  this: M { def metadata: Meta[M] } =>
  val (id, updatedAt, updatedBy) = metadata
}
```

This API takes full advantage of Scala and Slick type-checking capabilities. Thanks to the type parameter on `Id`, the compiler will detect errors such as misusing the `Id` of a person as the `Id` of a submission.

3.2 Implementation

Under the hood, SlickChair create multiple database records for a given *natural key*. When using the `Database` object, the user queries is composed with a temporal filter that extracts the most recent records where created before the time `date`. This corresponds to the *Type 2 slowly changing dimension management methodology*, as described in [11].

This approach has the advantage to no require any update, new records are simply appended to tables and only become visible for the newly accessed `Database` values. It also minimises the number of tables, which is beneficial because tables directly manipulated by the user via Slick's domain-specific language. We development this API with H2 and PostgreSQL, but as the implementation uses the Slick generic `JdbcDriver`, it should be compatible with all database systems supported by Slick.

Finally, we should note that in it's current stage, our *database as a value* API does not allow the expression of transactions. The timestamp based implementation is by nature atomic: the temporal filter used on every table ensures that the database will be viewed either before or after a group of insertions. However it is currently impossible to express a *transaction*, that is, to enforce that the database is not modified between the time it's value is queried and the insertions are effective. Fortunately SlickChair does not require transactions.

4. PAPER-REVIEWER ASSIGNMENT

One of the main responsibility of the program chair is the assignment of submissions to program committee members for reviews. In order to provide the best quality reviews, the program chair typically asks program committee members to bid on submissions, and uses this information to guide the assignment. While this task could reasonably be done manually for a small number of submissions, it quickly becomes difficult as the number of submissions goes up. In fact, the mathematical formulation is known to be NP-Complete. We will see how we formulated this problem using the OscaR operational research library, which provide different search patters to explore the space of all possible assignments under limited time constraints.

4.1 Mathematical formulation

This problem can be seen as a generalization of the stable marriage problem. On one side, the reviewers emitted preferences for the submissions in the form of bids, called the *interest preference*. On the other side, authors want their submissions to be reviewed by program committee members with topic knowledge, the *expertise preference*. The paper-reviewer-assignment problem is a generalization of the traditional stable marriage problem in the following ways:

- *Polyamory*: submissions and reviews are matched many-to-many instead of one-to-one.
- *Indifference*: *interest* and *expertise preferences* contain ties between instead of being total ordered.
- *Incomplete lists*: conflict of interest forbid certain assignment, removing elements from the candidate lists.

While each individual variant can be solved in polynomial time, the combination of the three generalizations is NP-complete [8]. An approximation algorithm to maximize fairness among the reviewers is discussed in [7], but it revolves around repetitively solving linear programmes, which not a practical solution.

4.2 OscaR formulation

OscaR is a Scala library for Operations Research [13]. Among other techniques, it allows to formulate problems using constraint programming. Given a matrix of `preferences`, a list of `conflicts`, and four integers `nPapers`, `nReviewers`, `nReviewPerPaper` and `nReviewsPerReviewer`, the following program searches for an assignment maximizing the sum of reviewer satisfaction:

```
val m = makeMatrix(nReviewers, nPapers, 0 to 1)

0 until nPapers foreach { j =>
  add(sum(m col j) == nReviewPerPaper) }
0 until nReviewers foreach { i =>
  add(sum(m row i) == nReviewsPerReviewer) }
conflicts foreach {
  add(m(_, _) == 0) }

maximize(weightedSum(preferences, m)) search {
  binaryMaxDegree(m.flatten.toSeq) }
onSolution { return m } start()
```

In this code, the `add` function allows to add constraints to the definition of the problem. It is used to enforce that all submissions have the same number of review, all reviewers have the same of review, and no conflicting assignment are made. The `maximize() search {}` and `onSolution {} start()` constructs are used to define the objective function, the search heuristic and start the resolution. Before getting to this stage, some preprocessing is required to aggregate and prepare the data, which notably includes the addition of dummy submissions such that repartition of reviews is perfectly balanced among reviewers.

5. CONCLUSION AND FUTURE WORK:

- Macros (remove boilerplate on the database API)

6. REFERENCES

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

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

[3] Datomic Clojure API. Available at: `http://docs.datomic.com/clojure/`.

[4] 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.

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

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

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

[8] J. Goldsmith and R. H. Sloan. The AI conference paper assignment problem. In *Proc. AAAI Workshop on Preference Handling for Artificial Intelligence, Vancouver*, 2007.

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

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

[11] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 2002.

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

[13] OscaR Team. OscaR: Scala in OR, 2014. Available at: http://bitbucket.org/oscarlib/oscar.

[14] 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.

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

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