Architectural Requirements Notes

P.A.P.E.R.S (Publication And Papers Electronic Repository System)

University of Pretoria - Team Charlie

Notes By:
Claudio Da Silva - 14205892
Dillon Heins - 14035538

GitHub Repository - Team Charlie
For more information, please click here

5 March 2016

Contents

1 Architecture requirements

1.1 Architectural scope

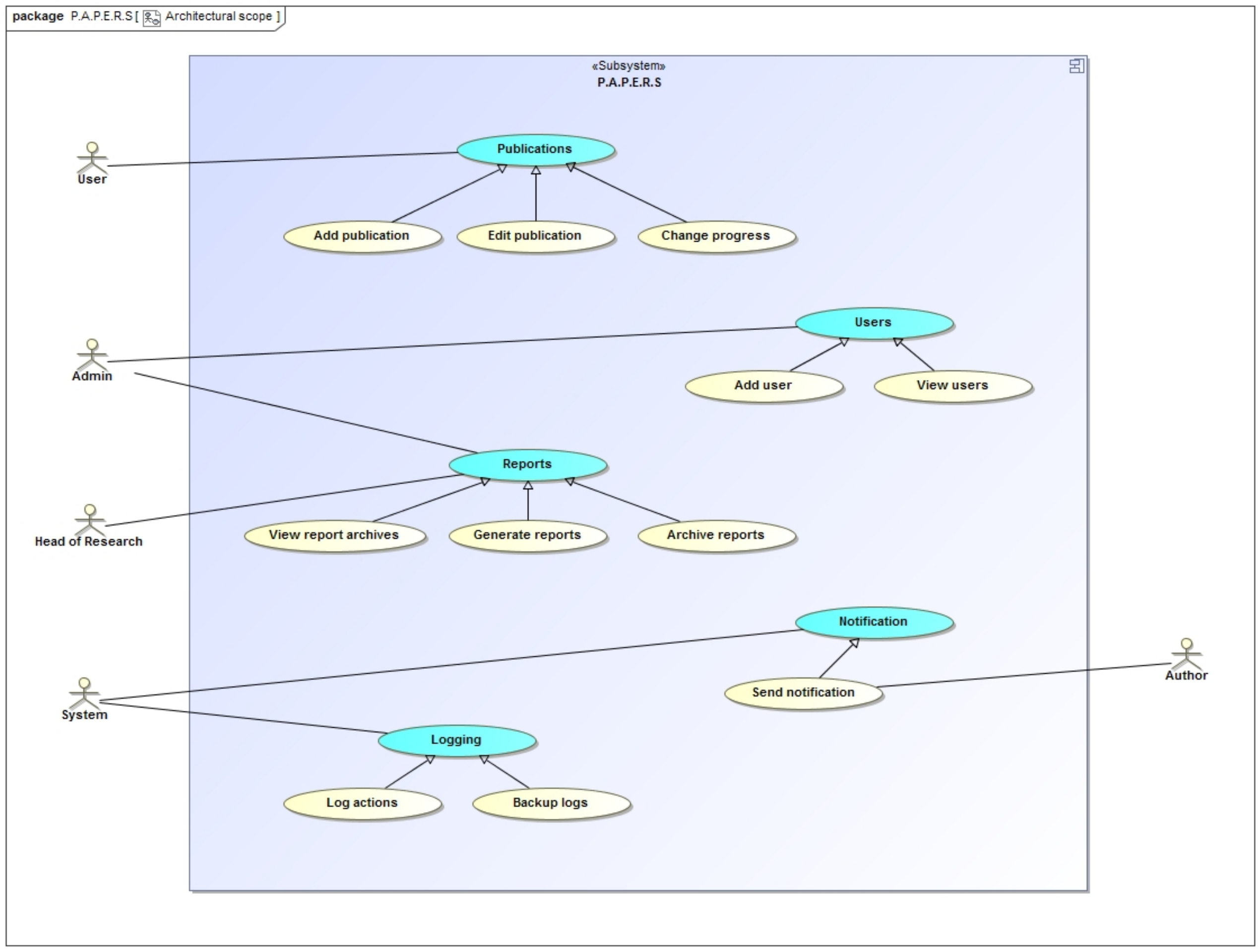

Figure 1: Architectural Scope - Use case for Architectural Scope (Further refined in use cases)

The system will be used to keep track of publications made for various research groups in the department, as well as keep track of the authors and the users in charge of things. From this we may later then generate reports on how various groups are doing as well as the department as a whole, and notify authors and users about upcoming deadlines and events.

In the event of dealing with queries on database items such as users and publications, for example insertion and deletion, as well as viewing, the system must first validate that those items do or do not exist in said database. Once that is verified, the queries should be run and the result returned. On success the user should be notified, on failure an appropriate exception should be thrown. Persistence must be maintained throughout the database with very little fault tolerance.

In dealing with reports, data being queried should be validated and the system should ensure that all queried data is recent and correct, halting any new additions during reporting. Reports should be properly archived for later retrieval and properly backed up regularly.

In dealing with notifications, the system should ensure that all emails being used for every author does in fact exist and is a valid email. When sending this email, likely from the mail server at UP, the SMTP connection used must be ensured to be secured and the email should be sent through valid channels as to not appear as spam mail to other users. The system should also validate that it is using the latest deadline updates at all times.

1.2 Quality requirements

The core quality requirements for this system include in order:

- Cost
- Integrability,
- Security,
- Auditability,
- Reliability,
- Usability,
- Maintainability,
- Flexibility
- Performance, and
- Scalability.

An in depth description of the various quality requirements:

- Performance
  - Performance in the system is of little worry as the system is not expected to grow to a large point in which it may slow down. The system should preferably be able to handle up to a hundred concurrent users at a time without major slow down effect and queries should be able to run within a matter of seconds.
- Reliability
  - The meta-data stored for each paper is required to be entirely correct at all times. To ensure this, we need to validate all possible input that can occur on the system in order to prevent any possible data inconsistencies. On the same note all data has to be persistent and remain in the database for the duration of the systems use, which means in all but the most extreme circumstances such as a database clean up for example, all entries must remain even if in a terminated state.
- Scalability
  - Scalability of this particular system is not of very particular importance. At the rate which the system will be used, chances of heavy fragmentation, data becoming too big or concurrency issues will be vary minimal.
- Security
  - Security is critical in the system as meta-data for unpublished papers can be sensitive information. Thus it is empirical that only authorised people have any access to the system at all. It is also vital to separate the concerns of each group from one another, allowing each group to only access the information from the groups they have access to. The Administrator should also be the only one allowed to make major changes to the system in anyway, and should be the only one authorised to make major persistence changes.
- Flexibility

  - The system should be developed as to not be platform dependant or browser dependant in it's operation, allowing it to essentially be used from theoretically any device capable of an internet connection and HTML5, this includes iPhones which may still connect through safari as opposed to an iPhone application
  - Flexibility may further be required in the event of change in units and venues for various existing conferences and journals, change in details for various users and authors registered on the system as well as flexibility for the addition or reduction of research groups.

- Maintainability
  - The system consists of a database which is required to stay consistent. To this end the system should contain well formatted documentation in order to properly maintain the system and ensure all data is in it's most consistent and reliable state.
- Auditability
  - All actions performed on the system should be audit-able via use of logging. A log entry should be created for each action such as adding or changing a publication for example. All actions performed in the system should be traceable to the user that performed them and possibly if so, where they performed it from in terms of IP as to have a form of accountability in the persistence of the meta-data.
- Integrability
  - The system should be easily integrable with the database system being used, allowing for simple queries in short amounts of time.
  - The system should be integrable with a web front end that should allow users to easily and securely use all the features of the system that their authorisation level permits.
  - The system should further be integrable with an Android front-end, which should allow for the exact same functionality of the web based front-end only designed ot be more mobile friendly.
- Cost
  - No proprietary software should be used in this system, and all licences for any software should be licensed under the open use policy.
- Usability
  - Users should easily be able to access the system securely from their most convenient location, either over their favourite web browser, the android application or via a mobile web page on their phone. All versions should allow full usage of all the features.
  - Reports generated should be easily readable and allow for a logical and quick method of discerning various statistics from one another.
  - Publications should be simple to view and easily distinguishable in type. One glance should allow a view of the most important details.
  - Users should be easy to manage for the Administrator, allowing termination and reinstatement of users as well as addition of new ones to be a simple and streamlined process.

1.3 Integration and access channel requirements

- The system must host a server to which API calls may be performed, to achieve this we use:
  - Django (A Python based web framework allowing easy implementation of the MVC principal.)
  - AngularJS (For easy unit testing functionality and dependency injection)
- The system should make API calls via browser, the browser functionality and calling is done via:
  - HTML5 (To ensure access to the latest and most stable versions of features)
  - Javascript with a JQuery framework (For the functionality)
- The system will require and android interface to interact with the API. To achieve this we make use of:
  - JAVA (The standard language of every android application.)
  - JQuery for mobile (A convenient method to manipulate data via JQuery on your android app.)
  - Sencha touch (An HTML5 framework for Android.)
- The system should be able to interact with the systems database, which will be hosted using:
  - PostgreSQL (An open source relational database that integrates perfectly with the Django framework.)
- The system should be able to send emails on a scheduled basis by sending mail from the CS mail server, for this we use:
  - Django-sitemessage (A framework developed to allow easy scheduling of emails as well as the ability to retry an email on a failure.)
- Protocols to be used include:
  - HTTP
  - SMTP
  - TCP

1.4 Architectural constraints

- The system should be developed using only non-proprietary technologies.
- The system is to be developed for Linux based systems (Mostly Arch Linux) used by the Computer Science department, however preferably the system should run on any general use operating system.
- The system should be browser independent, thus the system should be accessed via any browser without issue, including but not limited to (Firefox, Chrome, Safari, Edge etc.)
- The system should be handled locally and should not rely on outside internet sources in order to function.
- The API created must be queried by both a web browser, as well as an Android application.
- The system created should allow for users to use the application via mobile, employing technologies such as bootstrap in order to keep it mobile friendly. This will always allow users of iOS and Windows mobile to access the system via channels such as Safari and Edge.
- The system is required to be concurrent and make use of concurrent methods specifically in it's functionality, allowing up to a minimum of a hundred concurrent users to be active at the same time

2 Reference Architecture and frameworks

In this section we will discuss the reference architecture and frameworks that will be used in the P.A.P.E.R system.

2.1 An object-relational mapper

Object-relational mapper is a technique of converting data between incompatible systems by means of object orientated programming. For example the conversion of database attributes into a database object. This allow the interacting object-orientated language to manipulate this database object with ease, in essence the data can be presented in a way that any object is presented in that programming language.

Django uses Object-relational mapping(ORM) with regards to its Model-View-Controller (MVC) model. The data models and a relational database (Model) are manipulated with the ORM strategy to interact with the databases.

2.1.1 Advantages

- The advantage of using object-relational mappers with databases in particular is that joins aren't used that often as object types can be followed by means of referencing pointers.
- Relationships are also established by means of pointers which can increase efficiency for complex data.
- This approach works well for large amounts of data, as the object can be manipulated easily for each field.

2.1.2 Disadvantages

- Inefficient when used with small databases as objects will still be created which might be less efficient than a quick lookup of those particular fields.

2.2 Application Server

Software framework for web applications and a server to run the environment, is the principle that Application Server approach follows. An example of Application Server architecture framework is the Java EE framework. This architecture is based on the Layer model and Client-Server model and contains a service layer which is accessed by the developer. Django supports Java EE based Application Server.

2.2.1 Advantages

- Scalability
  - Resources are allocated efficiently
  - Objects reuse is ensured
- Integrability
  - Integrates well with REST
  - Database integration is provided
- Security
  - Authentication and confidentiality is supported

2.2.2 Disadvantages

- Uses a lot of system resources which might not necessarily be
- Relies on server to be running at all times
- Central point access can choke overall network's access to the server

3 Architectural Patterns

3.1 Introduction

Often times source code needs to be organised so that clear roles, responsibilities and relationships of different modules of the code can be well defined. Architectural patterns help in making this possible and numerous types will be discussed in this section. Using these patterns, code becomes easier to maintain, manage and visualise. They also make understanding of how each component works in a system easier. They are reusable solutions to a commonly occuring problems within a given context in Software Architecture.

3.2 Types of Architectural Patterns

- Layered Pattern
- Client-server
- Microkernel
- Master-slave
- Pipe-filter
- Hierarchical
- Model View Controller
- Blackboard Pattern

3.3 Architectural Patterns Proposed for PAPERS

3.3.1 Model View Controller

- Overview

  In the Model-View-Controller Pattern, an interactive application is divided into three parts: the Model is an object representing and storing the data and activities, the View displays information to the user and the Controller offers a way to change the state of the Model. To implement this patten for the system, the Python Django Framework will be used.

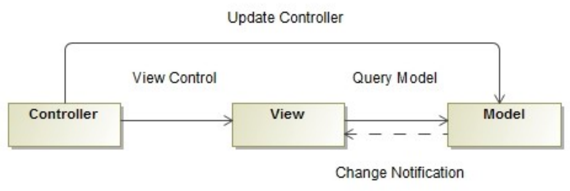

Figure 2: Model View Controller Pattern

- Motivation
  Separation of responsibilities is useful because functions can work normally when certain subsystems are changed. The Model View Controller makes this a possibility and hence it is chosen for use in this context.

- Components of the Model View Controller Pattern

  1. Model
     The Django Framework provides a Model functionality that handles the creation and storage of data in a relational databse that will contain essential fields and behaviours of the data stored for the system.

  2. View
     Normally, the View would be the HTML page or the Android Graphic User Interface, or GUI that displays data from the relational databse to the user for interaction purposes. However, Django implements the View as a "Python callback function for a particular URL, because that callback function describes which data is presented" to the user. Thus the View would be both the User Interface and Django's implementation of the View for the system.

  3. Controller
     The Django Framework itself will be the Controller as it will provide appropriate functionality for data manipulation. It will be used along with the REST, or Representational State Transfer API to retrieve data from the Android and the Web Interface.

3.3.2 Layered Pattern

- Overview
  The Layered Pattern assists in structuring applications that can be decomposed into subtasks. Each subtask provides services to the layer above it and services in that layer are implemented using those from the layer below it.

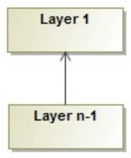

Figure 3: Layered Pattern

- Motivation
  A system typically consists of subsystems that have a certain functionality. These functions need to be placed in layers so that their services can be used by other layers and a change in behaviour of one layer does not affect the layers below it. This assists in developing different layers independently.

- Components of the Layered Pattern
  This pattern will be used to separate the internal functions within each subsystem and each function will be placed in a layer. It will be used to structure the Model View Controller Pattern described above to provide additional decoupling and structure to the system.

3.3.3 Master-Slave

- Overview
  The Master-slave Pattern can be described as a model where one process (Master) has unidirectional control over one or more processes that are known as Slaves.

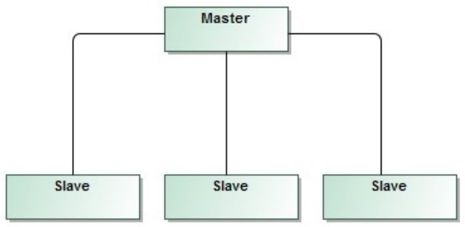

Figure 4: Master Slave

- Motivation
  An interactive application typically consist of multiple users that use it at different rates. These users make requests that usually expect a response to either use the data or edit it in some way and thus is imperative for the application have an ability to collect user information simultaneously.
- Components of the Master-Slave Pattern
  This pattern will be used to make the system concurrent, i.e to enable multiple users to use it simultaneously and continuously at different rates.
  - Master
    The Master will be the server that will respond to each request it receives from the Slave using the system, and will also provide instructions to these Slaves.
  - Slaves
    The Slaves will be the clients connecting to the server and will send requests concurrently to achieve a certain goal.

3.4 Other Patterns

3.4.1 Client Server Pattern

- Client Application provides access to humans, or systems (Fritz, 2012)
- Requests are typically handled in separate threads on the server.
- Inter-process communication causes overhead.
- Requests and data have to be transformed because they have different representation in Server and Client.

3.4.2 Microkernel

- Used for applications that must adapt to changing system requirements.
- This pattern provides minimal functionality (Internal Servers) as a building block and customizable components that can be adapted accordingly (External Servers).

- It provides an Adapter that provides a single access point (Fritz, 2012)
- It offers a more complex process management if not provided by architecture (Fritz, 2012)

3.4.3 Pipes and Filters

- Consists of Pipes and Filters:
  - Pipe - Input and Output
  - Filters - Stateless sevices or pure functions.
- Assemble higher-level stateless services from lower level ones (Fritz, 2012)
- It is not used because it is difficult to use for interactive processes such as Web Applications.

3.4.4 Hierarchical

- It is a recursive containment hierarchy (Fritz, 2012)
- It is not used due to its lack of flexibility.

3.4.5 Blackboard Pattern

- Blackboard repository hosts problem specification and current state of solution (Fritz, 2012)
- It is typically used where deterministic solutions are unavailable.
- It is not used because it requires complex synchronization and access control techniques.

4 Reference Frameworks

4.1 Django

Django is a web framework, written in Python which uses the Model-View-Controller architectural pattern, or MVC for short. Django's main aim is to provide a framework on which to build websites that are primarily based on complex databases. The use of Object-relational mapper in Django's MVC is described in 2.1. Django processes HTTP requests by means of a web templating system and regular-expression URL control (View and Controller).

Django further includes:

- Its own web server for developmental purposes.
- Form validation and storing of form data in database
- Caching framework with several cache methods
- Serialization system to produce and interpret XML and JSON representation of model instances.
- Python unit test framework

Django database support:

- PostgreSQL
- MySQL
- SQLite
- Oracle
- Microsoft SQL Server (through django-mssql)

Django can be used with:

- Python (Supported by default)
- JavaScript (through Swig)
- Ruby (through Liquid)
- Perl (through Template::Swig)
- PHP (Twig)

We prefer to use Django as it does have a slight learner curve, but will ease integrability with regards to our Android application.

4.2 Honorable Framework Mentions

4.2.1 AngularJS

Web-framework making use of client-side ModelViewController model. AngularJS makes use of the MEAN stack for its front-end, consisting of MongoDB database, Express.js web application server framework, Angular.js itself, and Node.js runtime environment. We prefer to use a server side approach as this eases integrability and simplifies implementation.

4.2.2 Ruby on Rails

Ruby based web-framework based on Ruby programming language. "Rails" uses a Model-View-Controller based model and emphasizes the use of JSON and XML for data transfer. Since Ruby has a high learning curve compared to Python for example, we prefer to use Django instead.

4.2.3 Zend Framework

Zend is an open source, object-oriented web application framework implemented in PHP 5. Zend supports multiple database systems and MVC is the preferred model of development. Since Zend uses only Object-Orientated PHP5 we decided against this approach as the integration might be difficult for our Android Application.

4.2.4 Bootstrap

Bootstrap is a front-end framework for creating websites and applications. It is mainly used for interface development and design. Bootstrap can be used in HTML, CSS and JavaScript. We decided on a Server-side approach rather than a front-end approach such as Bootstrap.

5 Technologies

For each of the following sections, all technologies we considered are listed.

5.1 Programming Languages

5.1.1 Java

Java is a general purpose, high-level programming language developed by Sun Microsystems. It is concurrent, class based and object-oriented. It was specifically designed to have as few dependencies as possible.

- Advantages:
  - Easy to use
  - Syntax is derived from C and C++
  - Comprehensive documentation
- Disadvantages:
  - Memory Inefficient

5.1.2 JavaScript

JavaScript is a high-level programming language that is, alongside HTML and CSS, one of the three essential technologies that allow content production for the World Wide Web.

- Advantages:
  - Because JavaScript is client-side, there is no delay by having to wait for a server response
  - Easy to learn and implement
- Disadvantages:
  - Security, the code being executed on the client-side is susceptible to malicious exploitation

5.1.3 Python

Python is a high-level programming language that emphasizes code readability.

- Advantages:
  - Efficiency, Python allows a programmer to solve the same problem in fewer lines of code than in other languages such as Java
  - Easy to read
- Disadvantages:
  - Syntax differs from conventional languages such as Java or C++, such as the omission of the semicolon
  - Indentation dictates blocks of code, a single wrong indentation will produce undesired or unexpected results from your code

5.1.4 PHP

PHP (PHP: Hypertext Preprocessor) is a server-side scripting language designed for web development.

- Advantages:
  - Works well with databases
  - Popular, most problems encountered have already been solved by other developers

5.1.5 C

- Advantages:
  - Fast run-time performance
- Disadvantages:
  - Low-level language, not ideal for applications or web development

5.1.6 C++

- Advantages:
  - Powerful language
  -
- Disadvantages:
  - No garbage collection, memory management has to be implemented by the programmer
  - Complex language

5.1.7 HTML

HTML (HyperText Markup Language) is the standard markup language to create web pages. It dictates the content of a web page. Alongside JavaScript and CSS, it is one of the three essential technologies that allow content production for the World Wide Web.

- Advantages:
  - Standardized, it is the standard markup language to create web pages

  - Easy to learn
- Disadvantages:
  - Different web browsers may render the page differently
  - Bland, it has limited styling capability.

5.2 Frameworks

5.3 Ajax

AJAX (Asynchronous Javascript and XML) is a group of technologies used to create asynchronous web applications. It is used to change the content of a web page dynamically without having to reload the entire web page

5.4 AngularJS

Described in section 4.2.1

5.5 Bootstrap

Described in section 4.2.4

5.6 Django

Described in detail in section 4.1

5.7 Libraries

5.7.1 jQuery

jQuery is a cross-platform JavaScript library designed to simplify client-side scripting. It is the most popular JavaScript library in use today

5.8 Protocols

All protocols listed below were discussed in section 1.3

- LDAP (Lightweight Directory Access Protocol)
- HTTPS
- HTTP
- SMTP

5.9 Database Systems

The System will utilize a relational database and will use PostgreSQL to handle queries.

5.10 Operating Systems

The System will be created to primarily work on Linux as the entire Computer Department is running on Linux. The System will however be compatible with other operating systems such as Windows, Apple OS.