An object-oriented approach to deploying highly configurable Web interfaces for the ATLAS experiment

Bruno Lange¹, Carmen Maidantchik¹, Kathy Pommes², Varlen Pavani¹, Breno Arosa¹, Igor Abreu¹ on behalf of the ATLAS Collaboration

¹Universidade Federal do Rio De Janeiro, COPPE/EE/IF, Brazil
²European Laboratory for Particle Physics, CERN, Switzerland

E-mail: {bruno.lange,carmen.maidantchik,kathy.pommes,atlas.glance}@cern.ch

Abstract.
The ATLAS Technical Coordination disposes of 17 Web systems to support its operation. These applications, whilst ranging from managing the process of publishing scientific papers to monitoring radiation levels in the equipment in the experimental cavern, are constantly prone to changes in requirements due to the collaborative nature of the experiment and its management. In this context, a Web framework is proposed to unify the generation of the supporting interfaces. FENCE assembles classes to build applications by making extensive use of JSON configuration files. It relies heavily on Glance, a technology that was set forth in 2003 to create an abstraction layer on top of the heterogeneous sources that store the technical coordination data. Once Glance maps out the database modeling, records can be referenced in the configuration files by wrapping unique identifiers around double enclosing brackets. The deployed content can be individually secured by attaching clearance attributes to their description thus ensuring that view/edit privileges are granted to eligible users only. The framework also provides tools for securely writing into a database. Fully HTML5-compliant multi-step forms can be generated from their JSON description to assure that the submitted data comply with a series of constraints. Input validation is carried out primarily on the server-side but, following progressive enhancement guidelines, verification might also be performed on the client-side by enabling specific markup data attributes which are then handed over to the jQuery validation plug-in. User monitoring is accomplished by thoroughly logging user requests along with any POST data. Documentation is built from the source code using the phpDocumentor tool and made readily available for developers online. Fence, therefore, speeds up the implementation of Web interfaces and reduces the response time to requirement changes by minimizing maintenance overhead.

1. Introduction
Since its inception, ATLAS [1] has experienced an increasing demand for systems to support the operation of its technical coordination. The high turnover in management positions motivated the development of the Appointment system. Likewise, the fivefold increase in the number of publications and conference notes from 2010 to 2012 culminated in Analysis [2], a tool that allowed the entire submission procedure to be closely followed. Information on the Collaboration itself is made possible through Membership - where contracts, institutes and funding agencies can
be assessed. *Traceability* [3] and *RackWizard* are responsible for equipment management. While the former allows for radiation levels on the detector and its components to be properly gauged and stored, the latter provides a graphical solution for inspecting racks, crates and boards.

These systems are heavily supported by Glance [4], a technology introduced in 2003 to overcome the difficulties in accessing and integrating the technical coordination data stored in different databases. Glance provides an abstraction layer on top of heterogeneous sources in order to create search interfaces and have their results available through a Web browser or exported in accessible formats like CSV, XML or JSON.

Due to the collaborative nature behind the joint effort that built and now operates ATLAS, the underpinning systems are constantly prone to changes in their requirements. With more than 5000 physicists, engineers and students from 39 countries, the high turnover in management positions implies that the applications must quickly adapt to new practices and observe ever-evolving needs and functionalities as different members take over their appointments. In fact, since August 2011, the support contact address Atlas.Glance@cern.ch received an average of 67 emails per month, out of which over 1200 tickets were opened. The statistics showed that the overwhelming majority of requests represented isolated changes in the systems’ behavior. It then became clear that there was a need for a framework for building applications in which such interventions could be swiftly applied with the minimal maintenance overhead.

This paper presents Fence - an acronym for *Front-END ENgine for glaNCE*, a Web framework that applies the object-orientation paradigms of inheritance and polymorphism while making extensive use of configuration files to build flexible systems and unify the generation of the applications that support the ATLAS Technical Coordination under a single framework.

2. Fence

Fence’s basic structure is illustrated in Figure 1. Glance serves as an API that provides the information necessary to generate the content for any given interface. Configuration files are loaded into the engine at each request to then generate the HTML response to the user’s browser.

The key concept is that an application’s behavior can be altered by simply targeting the configuration files as opposed to a more costly intervention performed at the source code level. Therefore, not only a wide range of systems can be built but eventual changes requested by eligible users can be put in effect by the developers more efficiently. In some cases, granted that the configuration files can be exposed in a friendly interface - a Web editor, developers might not even need to intervene. Users should be able to modify the system themselves according to the upcoming of set of requirements.

2.1. Technologies

PHP is a programming language widely used for Web development that remains greatly popular despite the appeal of modern frameworks like node.js or Django. Its support for OOP, the prompt availability in CERN’s Web servers and the extensive documentation available online made it a suitable option for the framework’s server-side scripting. The deployed markup is constantly tested against online validation tools to ensure its compatibility with the HTML5 standard as defined by the World Wide Web Consortium (W3C) [5]. This compliance steadily enhances the framework’s functionalities as Web browsers evolve and provide support for the new features the standard introduces.

Client-side scripting is supported by jQuery, a lightweight, cross-browser JavaScript library[6]. To promote code reuse, Fence provides templates for developers to write high-standard plugins that can be shared among the entire team and across applications. The overall client-side support is carefully handled based on the premise that its scope applies solely to improving the user’s experience. In contrast, the core functionalities of a system reside on the server, where sensitive data can be securely manipulated and access control be performed. Through
progressive enhancement, more advanced functionalities are gradually made available in modern browsers but the main features are still operational even if JavaScript is disabled.

With the ever-increasing popularity of mobile and portable devices, guaranteeing access and ensuring the functionality of the applications in different screen sizes was considered crucial. Bootstrap [7] is an open-source front-end framework designed to support the development of dynamic Web applications. Its development is supported by the largest community of contributors on the GitHub platform, making it a compelling choice for Fence’s front-end.

Data retrieval is primarily ascribed to Glance. Through instances of the GlanceSearch class, information can be securely fetched from the databases on the server-side. The class constructor receives an optional boolean parameter that defines whether the results from that particular query should be made cacheable during the session. This flag is often enabled on data that is not likely to change regularly like the list of funding agencies or institutes in ATLAS and greatly improves the performance of the engine. Additionally, connectors to Oracle databases are also available through static public functions in the DBManager class. These connectors can be overwritten in inheriting classes to allow integration with MySQL or MongoDB, for example.

Fence makes extensive use of configuration files to store the system’s rules. Given the potential abstractness behind the rules to be encoded, a schema-less standard was required so that this information could be stored in a flexible way and extended with minimal overhead. The JSON (JavaScript Object Notation) data-interchange format has gained popularity recently for its performance prowess when compared to alternatives like XML [8] and was selected as the framework’s standard.

2.2. Architecture
The longevity of the ATLAS Experiment requires that the solutions that support its operation to be robust and reliable, while featuring a high flexibility to minimize maintenance efforts. Object-oriented programming (OOP) is a software architecture that achieves flexibility by encouraging
modular design and code reuse [9].

In OOP, inheritance refers to the ability of child classes to automatically acquire public and protected member functions and attributes from their parents. Specialization can occur on child classes through polymorphism, where functions and variables can be overwritten thus actively changing the behavior of the heir instances.

These paradigms are the cornerstones of Fence’s architecture, not only shaping the formation of its class diagrams but also providing guidelines for how configuration files, client-side scripts and even stylesheet files are loaded at each request. Through inheritance and polymorphism, solutions to specific problems can be made accessible to the entire development team, avoiding duplicated efforts and further minimizing the response time to requirement changes.

The server’s directory structure is adopted to break down large applications into smaller units and apply the concepts of inheritance and polymorphism in the way Fence identifies and loads configuration files. Folders primarily map to systems which in turn may each contain any number of subsystems. For example, the /membership folder hosts subsystems for members, institutes and funding agencies.

Each directory can be locally configured by hosting its own configuration.json file. Given any requested Web page, Fence configures itself to enable local configuration files to overwrite and extend their predecessors in the directory structure. By applying this hierarchical scheme, the framework allows for subsystems to define their own set of rules and variables that are only relevant within their scopes. Therefore, subsystems inherit their parents configuration and can further specialize their behavior by overwriting previously set directives.

3. Content configuration
In a broad sense, the bulk of the systems operated by the ATLAS Technical Coordination consists of applications that perform read and write operations from and into a database. Figure 2 shows the result of a basic search interface on members of the Collaboration generated by the framework with the direct input from Glance. In Figure 3, a multi-step form within the ATLAS Appointment system is illustrated.

Fence logically splits its HTML response into three groups. Each interface consists of a header, a footer, and the content itself. While the header is tightly controlled by the configuration files in each folder within the path that leads to the requested interface, footers are generally static. A contact button is shown at the bottom right corner alongside with a JIRA [10] issue reporter.
Users are, at any moment, one button away from reporting bugs or submitting questions into the development team’s issue tracking system.

The content section occupies the central area, where actual data is presented. The build up and deployment of content is subsidized by the abstract base class `Content`. Its constructor requires a path to a configuration file that holds the JSON description of the entities it is meant to deploy in an HTML format that can thus be rendered by a Web browser. The base class provides a series of protected methods to process objects and assure that the information they encode can be accessed by the user’s credentials. Finally, any non-abstract child class must overwrite the public method `render`, responsible for deploying the content markup.

Virtually any interface can be rendered through children of the `Content` class. However, the fact that the instances require a configuration file in their constructors encourages developers to remove the rules from the source code. This approach guided the framework’s support for search interfaces, in which the Technical Coordination data can be assessed, and single or multi-step forms, in which data can be generated.

### 4. Forms

The ATLAS Appointment system was introduced so that certain members could be granted certain privileges during their mandates while also providing a tool for the Collaboration to keep track of whom occupies what roles at any given moment. Given its relative simplicity, it was the first system to be fully migrated to Fence and has been in production since October 2014.

The process of creating new appointments to members is conducted by the interface depicted in Figure 3. It is a 4-step procedure which involves the selection of a category, the definition of the appointment name followed by management flags and, finally, a standard confirmation step.

The class `Form` inherits `Content` and provides support to write operations in Fence. It is controlled by a configuration file that fully describes the resulting single or multi-step form. Labels, steps and inputs within each step are encoded in JSON and processed by the engine to generate the form’s markup. Fence checks for the integrity of the submitted values and flags any validation errors into the input instance. These errors can then be accessed by public methods to provide user feedback and prevent faulty values from being entered into the database. In a multi-step procedure, any breach halts the form’s submission. Figure 3 portrays an example of an unsuccessful attempt at advancing to the third step of the appointment assignment procedure. The alert box points out that the “Mandate” input should have been filled, as encoded in its JSON description through the `required` attribute within the `rules` method.
Clearance attributes can be attached to the input description as well to determine what set of users are eligible to act on them. In the example above, the “Mandate” input shall only be enabled for users who hold the EDIT_APPOINTMENT_MANDATE privilege. Permissions are stored in the database and are acknowledged in Fence through its user authentication methods. Clearance attributes also include usergroups and egroups. While the former is also stored in the database, the latter is provided automatically upon CERN’s Single Sign-On (SSO) authentication. Though validation routines are carried out primarily on the server-side for security reasons, sometimes they can be also assigned to the client-side in order to avoid unnecessary loads on the server and improve the user experience. The available methods within the rules attribute mimic those of the jQuery validation plug-in so as to simplify its integration. However, if JavaScript is disabled or, more likely, prone to failure or even manipulation by a malicious user, the system gracefully degrades and relies on the server’s protection to guarantee the conformity of the data that ultimately reaches the database.

5. Search Interfaces

The class GlanceSearchInterface is responsible for deploying interfaces in which users can retrieve information from Glance and assess the results in a tabular view. Searches can be replicated at any moment and passed along by members of the Collaboration since all the relevant parameters are transmitted to the server via GET requests and thus encoded in the URL. Results can be further filtered with the assistance of the DataTables jQuery plug-in and exported into CSV and PDF formats. An example of a search performed on members of the Collaboration through the Membership system can be seen in Figure 2.

The resulting table is entirely configurable through its JSON description. Each column holds attributes to define its header and location, as well as whether the contents it hosts represent data that should be exported. Clearance attributes can also be included, in which case the entire column might be removed if the user does not have the necessary privileges to view it.

Integration with Glance is enabled by means of wild cards. Column identifiers can be referenced by wrapping them around double enclosing brackets. The result table shown in Figure 2 features in its first column a link to the member’s profile page. The formation of its anchor tags can be encoded by the string {{membership}}/details.php?id=\[[member_id]\], where membership is a reference to the system’s path as set in the configuration files and member_id relates to the member identifier as retrieved by Glance.

5.1. SuperSearch

It is often the case that multiple parameters need to be defined and logically arranged to form a particular search criteria. In ATLAS, this is particularly true for the Speakers Committee
Advisory Board (SCAB). Its primary purpose is to aid the selection of speakers for international conferences by compiling a list of members who should be given priority while ensuring a fair distribution of talks among the Collaboration. This selection takes into account many parameters such as last talk, comments, tasks, profession and so forth. Personal preferences are also considered as members can select up to three conferences in which they would rather present their results at, and define any number of periods when they are not available to give talks.

Glance allows for any number of filters to be defined, at which point users can select whether records should match all of the set parameters or, alternatively, any of them. That is, if the filters should be connected by AND or OR operators. The SuperSearch class expands on this concept to allow for the combination of the search criteria into any logical arrangement.

In this context, a clause is defined by the combination of a parameter, an operator and a value - POV, and is represented by a circle (or node), in the logic workspace (see Figure 4). Any number of clauses can be added to the workspace and rearranged to form the compound search criteria. New clauses are automatically added into a new stack on the far right side of the workspace and, through HTML5’s drag and drop feature, can be reallocated to another stack. This graphical approach provides a simple visualization of the combined logic that forms the search criteria. Each stack represents the conjunction of the clauses it contains, that is, nodes on the same column are logically connected with the OR operator. Stacks, on the other hand, connect logically with one another by the AND operator.

A complete description of the search criteria can be encoded in a hierarchical structure. For the example shown in Figure 4, its corresponding description can be found in Table 1. Each clause is given an unique identifier $c_i$. The interconnection between the clauses can be encoded simply enough by defining their parents, i.e. the set of clauses that occupy the stack at their immediate left side. The reconstruction of the logic given the description of its clauses yields the expression $c_1 \land ( c_2 \lor c_4 ) \land c_4 \land ( c_5 \lor c_6 \lor c_7 \lor c_8 )$. Figure 5 shows a hypothetical nomination that matches the search criteria. More importantly, it shows how the regular tabular view can be overwritten through polymorphism to display the results in a different format.
Table 1. Clause arrangement for example shown in Figure 4.

<table>
<thead>
<tr>
<th>Clause ID</th>
<th>Parameter</th>
<th>Operator</th>
<th>Value</th>
<th>Parents</th>
</tr>
</thead>
<tbody>
<tr>
<td>c_1</td>
<td>Submitted by (Appointment)</td>
<td>is</td>
<td>Physics Coordinator</td>
<td>-</td>
</tr>
<tr>
<td>c_2</td>
<td>Priority</td>
<td>is</td>
<td>1 (Highest)</td>
<td>c_1</td>
</tr>
<tr>
<td>c_3</td>
<td>SCAB Priority</td>
<td>is</td>
<td>1 (Highest)</td>
<td>c_1</td>
</tr>
<tr>
<td>c_4</td>
<td>Days since last talk</td>
<td>greater than</td>
<td>90</td>
<td>c_2,c_3</td>
</tr>
<tr>
<td>c_5</td>
<td>Tasks</td>
<td>contains</td>
<td>ZZ (Photons)</td>
<td>c_4</td>
</tr>
<tr>
<td>c_6</td>
<td>Tasks</td>
<td>contains</td>
<td>tautau (Photons)</td>
<td>c_4</td>
</tr>
<tr>
<td>c_7</td>
<td>Tasks</td>
<td>contains</td>
<td>BSM Higgs (Photons)</td>
<td>c_4</td>
</tr>
<tr>
<td>c_8</td>
<td>Tasks</td>
<td>contains</td>
<td>SoftQCD (SM)</td>
<td>c_4</td>
</tr>
</tbody>
</table>

6. Conclusion

The ATLAS Technical Coordination relies on a wide variety of applications that are constantly subject to changes in requirements due to a high turnover in management positions. The proposed framework sets out to build flexible interfaces that can more efficiently incorporate new practices, extend its functionalities and rectify its current rules with minimal maintenance costs. By promoting code reuse and the outsourcing of rules into external configuration files, Fence can be quickly tuned to build varied systems that achieve high flexibility at a minimal maintenance overhead.

Following the successful test case of the ATLAS Appointment system, a new version of RackWizard has been partially placed in production as it pushes Fence’s native support for graphical solutions. These precedents pave the way to the complete migration of all systems that support the ATLAS Technical Coordination.

Furthermore, the framework’s evolution foresees capabilities that will allow users to shape the applications without the intervention of a developer and, ultimately, allow for fully operational systems to be built from the ground up without any particular knowledge of programming languages or the involved technologies required.

References