Universidade Federal
do Rio de Janeiro
Escola Politécnica

FENCE: UMA ABORDAGEM ORIENTADA A OBJETOS NA CONCEPÇÃO DE SISTEMAS *WEB* ALTAMENTE CONFIGURÁVEIS PARA OS EXPERIMENTOS DO CERN

BRUNO LANGE RAMOS

Projeto de Graduação apresentado ao Curso de Engenharia de Controle e Automação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do grau de Engenheiro de Controle e Automação.

Orientadora
Carmen Lúcia Lodi Maidantchik, D.Sc.

Rio de Janeiro, Agosto de 2015

Fence: uma abordagem orientada a objetos na concepção de sistemas *Web* altamente configuráveis para os experimentos do CERN

Bruno Lange Ramos

PROJETO DE GRADUAÇÃO APRESENTADO AO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO DA ESCOLA POLITÉCNICA, UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, COMO PARTE DOS REQUISITOS NECESSÁRIOS À OBTENÇÃO DO GRAU DE ENGENHEIRO DE CONTROLE E AUTOMAÇÃO.

Examinada por:

Carmen Lúcia Lodi Maidantchik, D.Sc.

Prof. Afonso Celso del Nero Gomes, D.Sc.

Prof. Fernando Marroquim Leão de Almeida Jr., D.Sc.

Prof. Flávio Luis de Mello, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

Agosto de 2015

Lange Ramos, Bruno

Fence: uma abordagem orientada a objetos na concepção de sistemas *Web* para os experimentos do CERN.

108 p.: il. color; 29,7cm.

Orientadora: Carmen Lúcia Lodi Maidantchik.

Projeto de Graduação - UFRJ/Escola Politécnica/Engenharia de Controle e Automação, 2015.

Referências bibliográficas: p.103-108.

1. Sistemas *Web*. 2. Orientação a objetos. I. Maidantchik, Carmen Lúcia Lodi. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Controle e Automação. III. Fence: uma abordagem orientada a objetos na concepção de sistemas *Web* para os experimentos do CERN.

Agradecimentos

Primeiramente, como não poderia deixar de ser, agradeço à Lange e ao Ramos. Em reconhecimento a todas as dificuldades, serei, enquanto aqui estiver, grato.

Agradeço à família escolhida, que eu acolhi e que me acolheu. No Leblon, na Barra ou em Caxias.

À quem me instigou a pensar, desde o Bahiense até o CERN, obrigado.

Às amizades da praia e do clube.

À *elle*.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Controle e Automação.

Fence: uma abordagem orientada a objetos na concepção de sistemas *Web* altamente configuráveis para os experimentos do CERN

Bruno Lange Ramos

Agosto/2015

Orientadora: Carmen Lúcia Lodi Maidantchik, D.Sc.
Curso: Engenharia de Controle e Automação

No CERN, colaborações internacionais que reunem mais de 10 mil físicos, engenheiros e estudantes foram responsáveis pelo comissionamento e operação dos detectores de partículas acoplados ao LHC (*Large Hadron Collider*). Suas gerências empregam sistemas computacionais cujas aplicações variam desde a monitoração dos níveis de radiação dos equipamentos dos detectores até o processo de publicação de descobertas científicas. Conforme os experimentos progridem, os softwares acompanham esta evolução por meio de constantes adaptações. Além disto, a grande diversidade de práticas, tecnologias e necessidades inerente à natureza colaborativa dos experimentos implica em frequentes mudanças nos requisitos. Neste contexto, o *framework* Fence foi proposto e implementado para permitir a concepção e acelerar o desenvolvimento de sistemas *Web*. Apoiado vastamente em paradigmas da orientação a objeto, a solução vincula aos objetos arquivos de configuração para minimizar o impacto de alterações e permitir uma rápida resposta da equipe de desenvolvedores. Os sistemas gerados são naturalmente integrados através do compartilhamento de funcionalidades comuns encapsuladas em classes bases e transferidas às diferentes aplicações por herança. A demonstrada capacidade de reduzir de esforços de manuntenção e promover a rápida evolução das interfaces motiva a migração de 21 sistemas para o Fence.

Palavras-chave: tecnologias *Web*, orientação a objetos, reutilização de código, evolução de software, análise de impacto de mudança, experiência do usuário

Abstract of Final Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Engineer.

FENCE: AN OBJECT ORIENTED-APPROACH TO DEPLOYING HIGHLY CONFIGURABLE WEB SYSTEMS FOR THE EXPERIMENTS AT CERN

Bruno Lange Ramos

August/2015

Advisor: Carmen Lúcia Lodi Maidantchik, D.Sc.

Course: Control and Automation Engineering

The main detectors coupled to the LHC (Large Hadron Collider) at CERN represent international collaborations involving more than 10 thousand physicists, engineers and students from all over the world. The experiments dispose of several Web systems to support their operations. These applications, whilst ranging from tracking radiation levels in the equipment in the experimental caverns to managing the process of publishing scientific papers, are constantly being adapted in order to keep up with the experiments' evolution. Moreover, the inherent diversity of people, technologies and needs make the systems prone to changes in requirements. In this context, the Fence Web framework is proposed and implemented so as to promote the systems generation and support their evolution. Largely grounded by object-orientation paradigms, the proposed solution binds objects to configuration files thus minimizing the impact change and allowing for faster responses to requests. The resulting systems are integrated by inheriting common functionalities from base classes. The reduced maintenance overhead and capability for fast evolution pave the way for the migration of 21 Web systems.

Keywords: *web technologies, object-oriented programming, code reuse, impact change analysis, software evolution, rich user experience*

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Abreviaturas e Siglas

ALICE	A Large Ion Collider Experiment
API	Application Programming Interface
ATLAS	A Toroidal LHC Apparatus
CERN	European Organization for Nuclear Research
CMS	Compact Muon Solenoid
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
LHC	Large Hadron Collider
OOP	Object-Oriented Programming
PHP	Hypertext Preprocessor
SCAB	Speakers Committee Advisory Board
SQL	Structured Query Language
WWW	World Wide Web
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Sumário

Capítulo 1

Introdução

Desde a concepção da *World Wide Web* pelo cientista britânico Tim Bernes-Lee ao final dos anos 80 nos laboratórios do CERN (*European Organization for Nuclear Research*) [1], a Internet viu sua abrangência expandir-se de seletos centros de pesquisa para o que hoje representa uma complexa rede de bilhões de computadores, telefones móveis e sistemas embarcados. A conectividade tão característica da geração que se seguiu consolidou a utilização de software como mais expoente modelo de trabalho, seja no mundo corporativo ou acadêmico [2].

O sucesso de empresas depende cada vez mais da capacidade de soluções em software de proverem diferenciais frente à concorrência. A busca por redução de custos e acesso a uma mão de obra capacitada levaram diversas corporações a buscar instalações de desenvolvimento de software remotas [3]. Na área de pesquisa, esta tendência de globalização beneficia a interdisciplinaridade ao possibilitar a rápida e eficaz troca de informações entre centros de pesquisa. A combinação de distintas especialidades acadêmicas expande os limites de complexidade que sistemas podem alcançar, fornecendo ferramentas mais poderosas para a ampliação das fronteiras do conhecimento.

O crescimento tanto em tamanho quanto em complexidade dos sistemas é normalmente guiado de forma gradativa pela evolução de suas especificações. A visão de que os requisitos devam ser acordados em etapas fixas do ciclo de vida do software foi há muito abandonada pela concepção de que estes são, de fato, referências

maleáveis que demandam flexibilidade aos sistemas. Softwares evoluem, seja para a reparação de erros, incorporação de melhorias, adaptações a novas necessidades ou integração de novas tecnologias. De qualquer forma, a alteração de requisitos é vista como uma prerrogativa dos *stakeholders* que perdura durante todos os estágios do processo de desenvolvimento [4] e que deve, portanto, ser devidamente gerenciada.

Este dinamismo provoca desafios à Engenharia de Software já que desenvolvedores precisam lidar com mudanças imprevistas e, por muitas vezes, disruptivas nos requisitos que os sistemas devem observar [5]. Esta problemática é agravada pela crescente dependência de programas de terceiros, sobre os quais se tem pouco ou nenhum controle [6]. Robert Glass, autor de "*The Dark Side of Software Engineering*" [7], resume o conceito: "*Caminhar sobre a água e programar de acordo com as especificações é simples - contanto que ambas estejam congeladas*".

Na esfera científica, mudanças são naturais, decorrentes da evolução dos experimentos. O próprio CERN, que opera atualmente o mais poderoso acelerador de partículas do mundo em um túnel que antes abrigava o LEP (*Large Electron-Positron Collider*), é exemplo deste processo gradual de aperfeiçoamento que, invariavelmente, implica em frequentes mudanças nos sistemas computacionais empregados. Os softwares são sujeitos a constantes adaptações para acompanharem o avanço tecnológico dos experimentos e o amadurecimento das técnicas empregadas pelos especialistas. Soma-se a esta predisposição, o fato de colaborações internacionais exibirem uma ampla diversidade de práticas, tecnologias e necessidades, além de uma alta rotatividade de cargos gerenciais, para vislumbrar-se os benefícios na proposta de uma plataforma de geração de sistemas que detenha em sua estrutura mecanismos de rápida resposta a mudanças de requisitos.

1.1 Motivação

No que configura o mais ambicioso experimento científico já almejado, o CERN construiu e opera uma cadeia de aceleradores de partículas que culmina no LHC (*Large Hadron Collider*) [8]. Trata-se de um acelerador circular de aproximadamente

27 quilômetros de circunferência que se estende pelas fronteiras francesas e suíças a 100 metros abaixo da superfície. Nele, dois feixes de prótons ou íons são acelerados em sentidos opostos a velocidades próximas a da luz e feitos colidir em quatro pontos de interseção, onde enormes detectores são posicionados e avaliam até 1 bilhão de colisões a cada segundo [9]. A análise dos dados coletados pelos detectores põe à prova concepções formuladas por físicos teóricos acerca das mais fundamentais leis que governam o Universo ao mesmo tempo que sugere fronteiras inexploradas a serem investigadas pela Física.

O ATLAS (*A Toroidal LHC ApparatuS*) [10] é o maior dos detectores do LHC. De fato, como mostrado na Figura 1.1, seus 25m de diâmetro e 44m de comprimento lhe concedem o título de maior detector de partículas já construído. Sua concepção e operação envolveram um esforço colaborativo de mais de 3000 físicos, engenheiros e estudantes de 174 universidades e laboratórios em 38 países [11]. Assim como o ATLAS, o CMS (*Compact Muon Solenoid*) [12] é um experimento de propósito geral, isto é, projetado para investigar uma vasta gama de possíveis campos da Física. Independentemente, ambos experimentos chegaram a conclusões que comprovaram a existência do elusivo Bóson de Higgs em Julho de 2012, descoberta que confirmou o mecanismo através do qual partículas elementares adquirem massa e concedeu a Peter Higgs e François Englert o prêmio Nobel de Física no ano seguinte [13, 14].

Há ainda detectores de objetivos específicos. No ALICE (*A Large Ion Collider Experiment*) [15], colisões de íons (de chumbo, tipicamente) recriam o chamado plasma quark-glúon, estado da matéria de tamanha densidade energética que físicos acreditam ter existido somente durante uma fração de segundo após o Big Bang [16]. O LHCb (*Large Hadron Collider beauty*) [17], por sua vez, observa padrões de decaimento da partícula conhecida como quark *beauty*, ou *b*, para investigar diminutas assimetrias que possam explicar a dominância da matéria sobre a anti-matéria no início do Universo.

Colaborações internacionais do porte do ATLAS, CMS, ALICE e LHCb representam desafios as suas gerências que motivaram a criação de diversos sistemas de suporte. Em particular, como institutos do mundo inteiro foram incumbidos

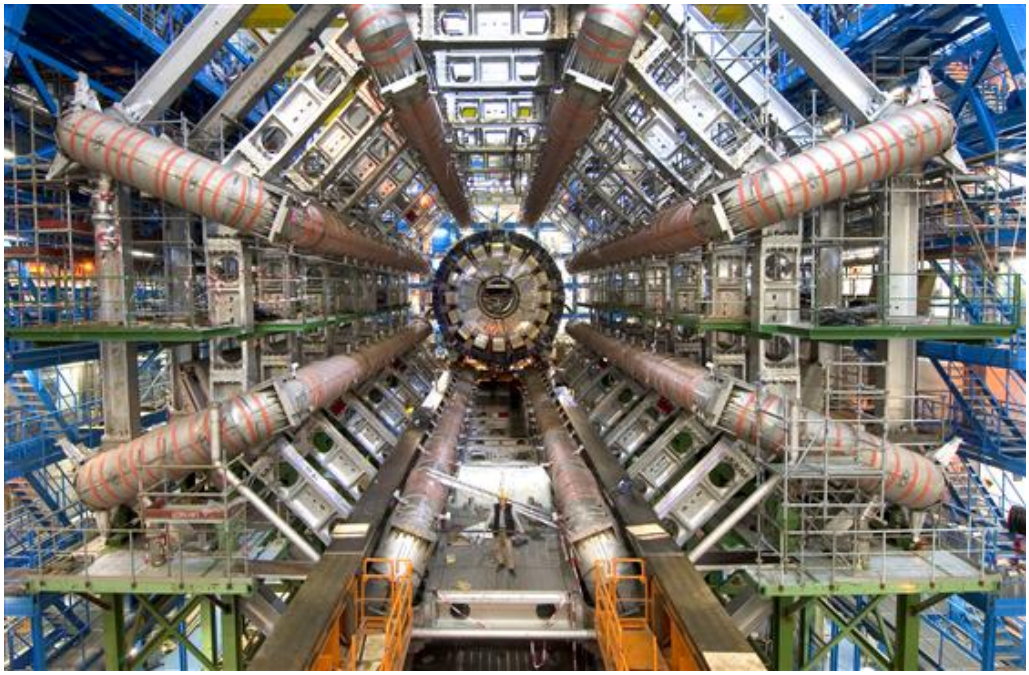


Figura 1.1: O detector ATLAS. Extraído de http://www.atlas.ch/photos.

do comissionamento e operação de diferentes partes do detector, a dificuldade em integrar-se informações armazenadas em banco de dados heterogêneos motivou o desenvolvimento da plataforma Glance [18]. Ao prover uma camada abstrata sobre as distintas tecnologias que armazenam os dados da gerência dos experimentos, o Glance abriu caminho para criação de 17 sistemas *Web* para a Coordenação Técnica do ATLAS, além de 2 para o ALICE e 2 para o LHCb.

Além da grande diversidade de práticas, necessidades e tecnologias empregadas por diferentes países, as colaborações por trás dos experimentos do CERN apresentam uma alta rotatividade em seus cargos gerenciais. A combinação destes dois fatores sujeita os sistemas de suporte a constantes mudanças em seus requisitos. Desde Agosto de 2011, o sistema de gerenciamento dos projetos do ATLAS registrou, sozinho, mais de 1200 ocorrências dos quais aproximadamente 3/4 representaram tarefas gerais de manutenção, aprimoramento de funcionalidades existentes e pedidos de novas funcionalidades, conforme mostrado no gráfico da Figura 1.2.

Ao contrário de outros tipos de produtos, softwares destinam-se a ser soluções maleáveis, empregados quando delas se espera uma evolução constante. E, embora softwares não estejam sujeitos a deterioração, esforços de manutenção normalmente

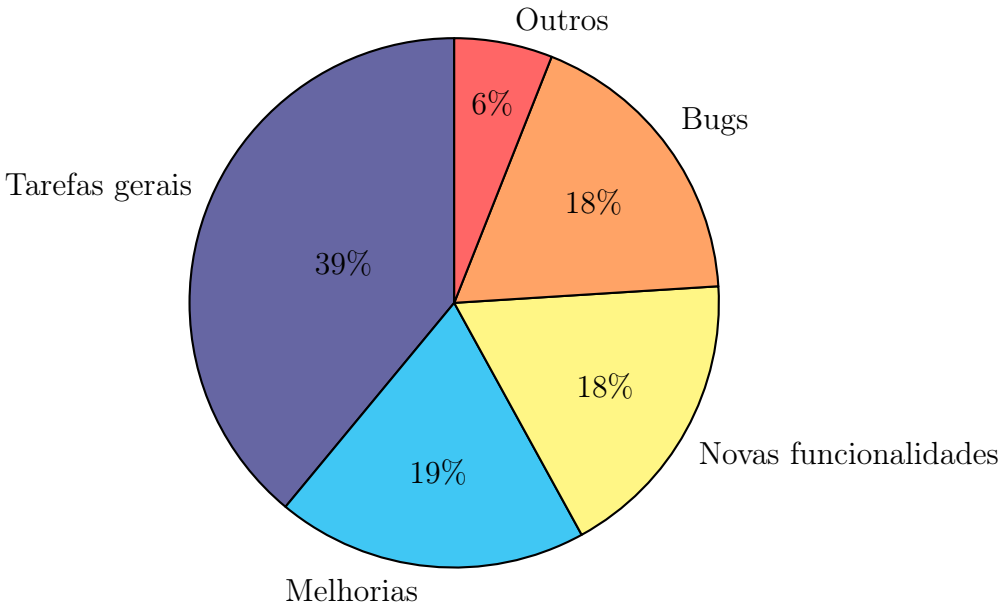


Figura 1.2: Composição das mais de 300 intervenções nos sistemas de suporte do ATLAS desde Setembro de 2014.

envolvem mudanças que os degradam [19, 20]. Como os sistemas de suporte às gerências dos experimentos são expostos a constantes mudanças de requisitos, entende-se que seus desenvolvimentos sejam apoiados por arquiteturas que facilitem a implantação de novas especificações ao atenuar o impacto de mudanças e minimizar esforços de manutenção. Ao reduzir, consequentemente, o tempo de resposta a alterações de requisitos, a arquitetura permite uma evolução mais rápida dos sistemas de gerência.

1.2 Objetivo

O objetivo deste projeto é criar um framework *Web* sob o qual os sistemas de suporte às gerências dos experimentos do CERN possam ser desenvolvidos de forma a facilitar a implementação de novas funcionalidades e minimizar os custos de manutenção das interfaces, reduzindo-se portanto, o tempo de resposta a mundanças nos requisitos.

Para tal, a arquitetura do framework deve ser convidativa a eventuais alterações nas regras das interfaces. Quando mudanças nas especificações forem requisitadas, espera-se que intervenções pontuais sejam capazes de se propagar automaticamente

por todas as aplicações. Além de evitar trabalhos dobrados, esta abordagem garante coerência entre os sistemas.

Como as coordenações técnicas lidam frequentemente com informações sensíveis, há uma grande preocupação com a proteção dos dados. A solução proposta deve prover garantias de que interfaces protegidas sejam acessíveis somente por usuários autorizados. Portanto, instrumentos de controle de acesso e monitoramento de usuários também se fazem necessários e devem ser nativamente fornecidos pela solução proposta.

Adicionalmente, a qualidade e consistência das informações armazenadas nos bancos de dados dependem da conformidade dos dados inseridos pelos usuários com as regras do sistema. Operações de escrita devem ser controladas pelo framework por meio de rigorosas rotinas de validação.

Por fim, visto que os sistemas são acessados por membros das colaborações em todo o mundo, o framework deve ser confiável e otimizado para a geração de páginas *Web* o mais rápido possível. Com a crescente popularização de dispositivos móveis como *smartphones* e *tablets*, a garantia de acesso e funcionamento e performance das aplicações em telas menores também é considerada crucial.

1.3 Metodologia

Muito da literatura especializada dedicou-se a documentar as dificuldades [5] e análisar o impacto causados por mudanças nos softwares [19] como uma importante parte da chamada Engenharia de Requisitos ou RE (da sigla em inglês *Requirements Engineering*) [6].

Dentre os vários fatores que motivam a mudanças dos sistemas, pode-se citar dois cenários que, no contexto deste projeto, se fazem mais relevantes. Primeiramente, alterações são realizadas quando ocorrem falhas na operação do programa. Seja por erros de implementação, vulnerabilidades do próprio algoritmo ou problemas infraestruturais, ocorrências dos chamados *bugs* polarizam a força de trabalho dependendo da gravidade das complicações que tais anomalias introduzem.

Em seguida, intervenções são promovidas para que softwares possam acompanhar eventuais variações nas circunstâncias em que operam e observar mudanças nos requisitos impostos pelos *stakeholders*. Esta dinâmica é tão necessária para a sobrevivência de softwares - especialmente frente à competitividade de ambientes corporativos, que o termo *software evolution* [21] vem sendo utilizado cada vez mais frequentemente para descrever tanto esforços de manutenção quanto para a implementação de novas funcionalidades e melhorias.

Softwares científicos se enquadram neste cenário, porém por conta de uma característica própria. A evolução de seus requisitos se dá pela própria progressão dos experimentos. Sob esta ótica, soluções computacionais são empregadas para apoiar a geração de conhecimento e, portanto, as especificações são expandidas conforme novas áreas vão sendo exploradas.

Para que o *framework* promovesse a evolução das aplicações sem comprometer suas integridades, ficou clara a necessidade de reduzir-se o impacto de mudanças. A solução encontrada envolveu o registro das regras dos sistemas em **arquivos de configuração** externos ao código-fonte. Esta abordagem permite que certas alterações possam ser implementadas com um impacto virtualmente nulo.

Raras são as ocasiões em que o escopo de um requisito abrange uma única página *Web*. A coesão do sistema depende, portanto, da correta propagação das especificações por todas as aplicações. Para isso, é importante que cada unidade abstrata de conhecimento do sistema seja representado por uma interface única, autoritária e inequívoca [22]. Esta abordagem, guiada pelo princípio conhecido como DRY (*Don't Repeat Yourself*), facilita a manutenção dos códigos e reduz chances de erros e inconsistências.

Embora os experimentos do LHC compartilhem diversas semelhanças, suas gerências tem total liberdade de aplicar práticas específicas de acordo com suas necessidades. A capacidade de generalização e propagação de requisitos do framework não pode, de forma alguma, oferecer empecilhos à instituição de procedimentos particulares a cada experimento. A **Orientação a Objetos**, ou OOP (*Object-Oriented Programming*), é uma arquitetura de programação que promove o compartilhamento

de recursos entre diferentes entidades (classes) sem comprometer suas capacidades de especialização [23].

Estes pilares deram, portanto, suporte à construção do framework e guiam sua evolução. Os estudos de análise de impacto de mudanças e gerência da evolução de softwares forneceram evidências convincentes de que uma estratégia que envolva a externalização das regras do sistema e o emprego de OOP pudesse reduzir os esforços necessários e promover a rápida evolução dos sistemas.

1.4 Sobre este documento

O **Capítulo 2** contextualiza este projeto no ambiente do CERN. O laboratório, seus objetivos e algumas de suas contribuições são apresentados. Seu maior acelerador de partículas, o LHC, e os detectores a ele acoplados são brevemente descritos.

A análise das gerências dos experimentos do LHC é feita no **Capítulo 3**, com o enfoque para os desafios tecnológicos que motivaram a proposta do sistema Glance pela Universidade Federal do Rio de Janeiro em 2003. A análise dos 21 sistemas desenvolvidos para o ATLAS, ALICE e LHCb, fruto do sucesso do Glance, dá bases para a avaliação do cenário que motivou a concepção deste projeto.

O **Capítulo 4** identifica as especificações da solução proposta para então sugerir a arquitetura do framework. Conceitos-chave como o estabelecimento de ambientes, controle de qualidade do código-fonte, emprego de arquivos de configuração e a aplicação dos paradigmas de Orientação a Objetos são analisados em profundidade para embasar a implementação da solução proposta, coberta no **Capítulo 5**.

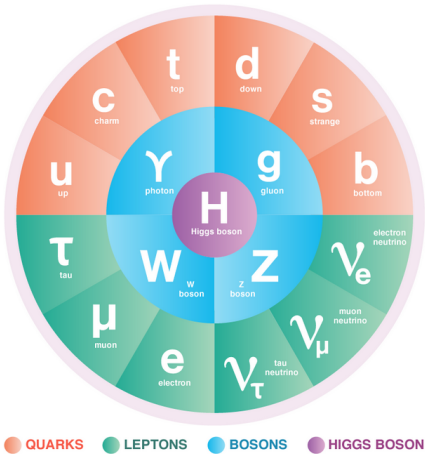
O **Capítulo 6** apresenta as conclusões deste projeto.

Capítulo 2

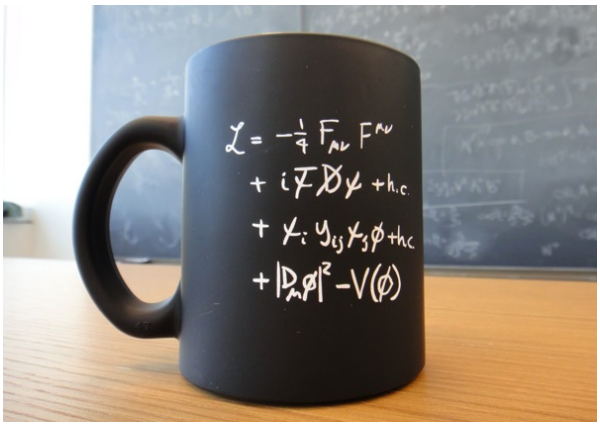
O CERN e a Física de Partículas

Fundado em 1954 para unir e retomar o pioneirismo científico de uma Europa fragilizada após a Segunda Guerra Mundial, o CERN é atualmente o maior laboratório em pesquisa fundamental do mundo. Seu propósito, "*...promover a colaboração entre estados europeus em pesquisa nuclear de caráter puramente científico e fundamental...*" [24], é hoje corroborado pela aliança de 22 países membros que conta, ainda, com outros 66 colaboradores espalhados por todo o mundo.

A pesquisa fundamental e, consequentemente, o CERN, existe porque a atual compreensão acerca do funcionamento do Universo é incompleta. A mais bem sucedida teoria científica já desenvolvida, modestamente chamada de Modelo Padrão, sugere que o Universo é composto de um reduzido conjunto de blocos construtores chamados de partículas fundamentais que interagem entre si de acordo com a regência de forças fundamentais. A Figura 2.1 mostra um arranjo das 17 partículas fundamentais e a Figura 2.1b mostra a equação que descreve a dinâmica destas partículas dentro do Modelo Padrão. No entanto, a teoria, entre outros problemas, falha em incorporar a gravidade entre estas forças e precisa dar lugar à Relatividade Geral no âmbito macroscópico onde a influência da força gravitacional não é negligível. A conciliação dos mundos micro e macro em uma Teoria de Tudo representa o objetivo último da Física. O CERN espera pôr a prova a validade das teorias candidatas utilizando-se dos mais poderos aceleradores de partículas já construídos.



(a) Partículas que compõe o Modelo Padrão. Extraído de [25].



(b) A equação que descreve a interação entre partículas do Modelo Padrão. Extraído de [26].

Figura 2.1: O Modelo Padrão da física de partículas.

2.1 O Grande Colisor de Hádrons (LHC)

O Grande Colisor de Hádrons, ou LHC (*Large Hadron Collider*) é o maior e mais poderoso acelerador de partículas do mundo. Inaugurado em Setembro de 2008, trata-se da última adição ao complexo de aceleradores do CERN, como ilustrado na Figura 2.2. Partículas são gradativamente aceleradas a cada etapa desta cadeia, ganhando mais energia antes de serem injetadas no próximo acelerador. A relação de energia cinética e velocidade para um próton no caminho até o LHC é mostrada na Tabela 2.1.

No LHC, 1624 eletroimãs formam um anel de 26.659 metros que, a velocidades próximas a da luz, são percorridos pelas partículas mais de 11 mil vezes a cada segundo. Dois feixes viajam em sentidos opostos em tubos mantidos a pressões comparáveis ao vácuo do espaço. As partículas carregadas são curvadas ao longo do LHC por campos magnéticos de 8.33 T, mais de 100 mil vezes mais fortes do que o campo magnético da Terra. A criação deste campo requer que os eletroímas operem em um estado supercondutor, isto é, que sejam capazes de eficientemente conduzir grandes correntes elétricas sem qualquer resistência. Para tal, os mesmos são resfriados a 1.9 K ($-271.3\,^{\circ}$C) [8], temperatura mais baixa do que os 2.7 K da radiação residual do Big Bang que permeia o espaço interestelar [28].

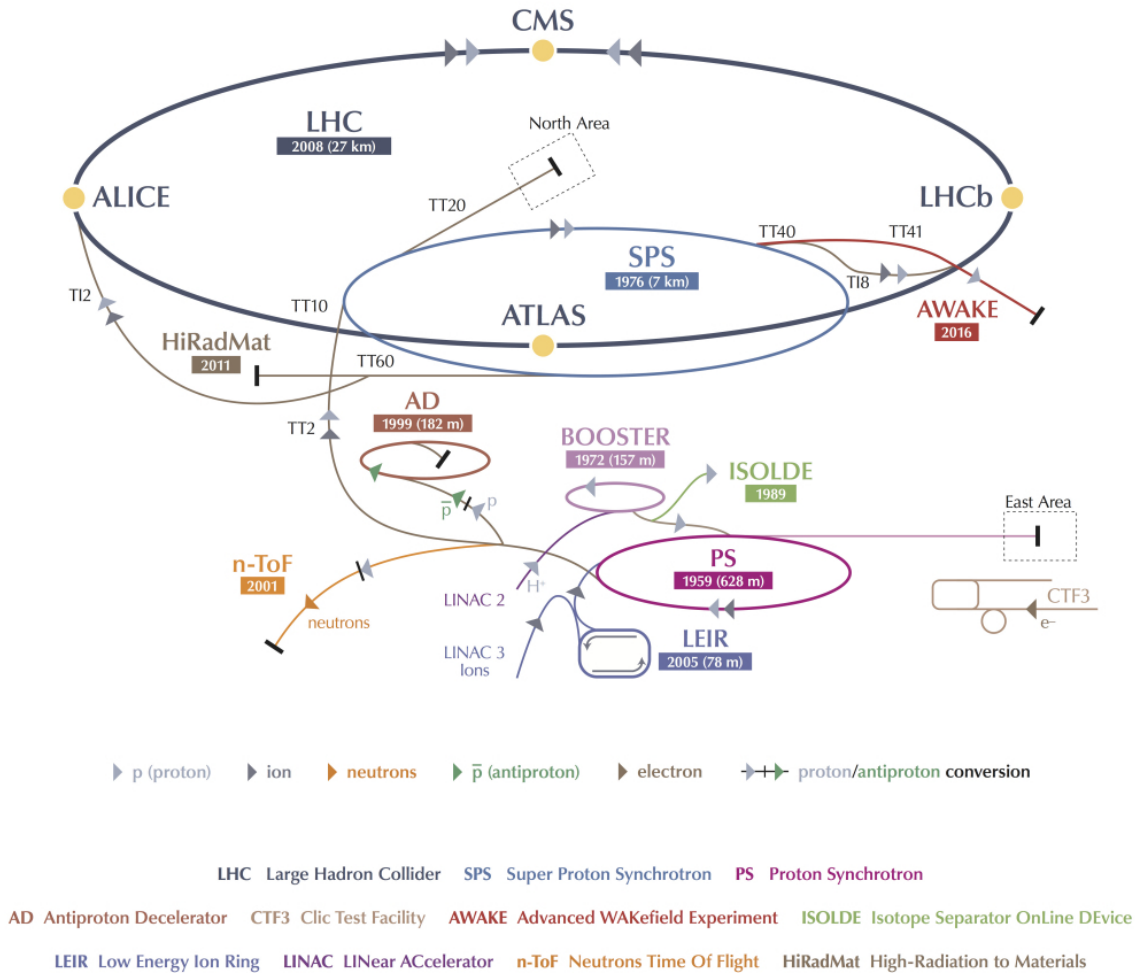


Figura 2.2: Complexo de aceleradores do CERN. Extraído de [27].

Em quatro pontos de interação onde os feixes se interceptam, detectores são posicionados para analisar o resultado das colisões. O objetivo do LHC é fornecer aos detectores pacotes de partículas, ou *bunches*[1], o mais compacto e com a maior energia possível. A redução da área efetiva destes pacotes intensifica a chamada **luminosidade** [29] do detector, aumentando as chances de colisão. E quanto mais energéticas, maiores as chances das colisões de criarem partículas maciças como o bóson de Higgs. Sua busca se estendeu por quase 50 anos justamente pela dificuldade de aceleradores predecessores do LHC em atingir níveis de energia capazes de criar amostras estatisticamente significantes da partícula.

[1] Cada feixe no LHC é composto por 2808 *bunches*, cada qual com aproximadamente 100 bilhões de prótons, comparável ao número de estrelas na Via Láctea. O cruzamento destes pacotes nos pontos de interação envolve, portanto, 200 bilhões de prótons, dos quais somente 20, em média, efetivamente colidem.

Acelerador	Energia cinética	Velocidade (%c)
Linac	250 MeV	31.4%
PS Booster	1.4 GeV	91.6%
PS	25 GeV	99.93%
SPS	450 GeV	99.9998%
LHC	7 TeV	99.9999991%

Tabela 2.1: Relação entre energia e velocidade de um próton na cadeia até o LHC. A massa inercial do próton é de 0.938 GeV/c^2. Adaptado de [28].

Depois de um período de manutenções e atualizações, o LHC foi reiniciado em Junho de 2015 após um hiato de quase 2 anos. Em sua segunda *run*, o LHC tem a capacidade de entregar aos detectores até 1 bilhão de colisões a cada segundo, com energias de até 13 TeV. O fluxo de dados, da ordem de 1 PB por segundo[1], esgotaria a capacidade de armazenamento em disco rígido do mundo [9, 30] em pouco mais de 3 dias se fosse, de fato, registrado. No entanto, somente 1 em cada 1 milhão de colisões correspondem a eventos de interesse. Sofisticados sistemas eletrônicos selecionam 1 a cada 10 mil eventos, resultando em um fluxo de 10GB/s que são transmitidos para a análise paralela de 15000 processadores, onde 99% dos eventos são descartados. Mesmo depois destas drásticas reduções, os quatro grandes experimentos armazenam aproximadamente 25 Pb a cada ano.

O processamento dos dados é incumbido à WLCG (*Worldwide LHC Computing Grid*), ou, simplesmente, *Grid*, que combina o poder de computação de mais de 170 centros em 42 países em diferente camadas. O CERN, ao centro, é conectado aos institutos colaboradores na primeria camada (*Tier-1*) por uma rede dedicada em fibra óptica [31].

[1] 1 PB (*petabyte*) = 10^3 TB = 10^6 GB = 10^9 MB ≈ 300 milhões de músicas em formato MP3.

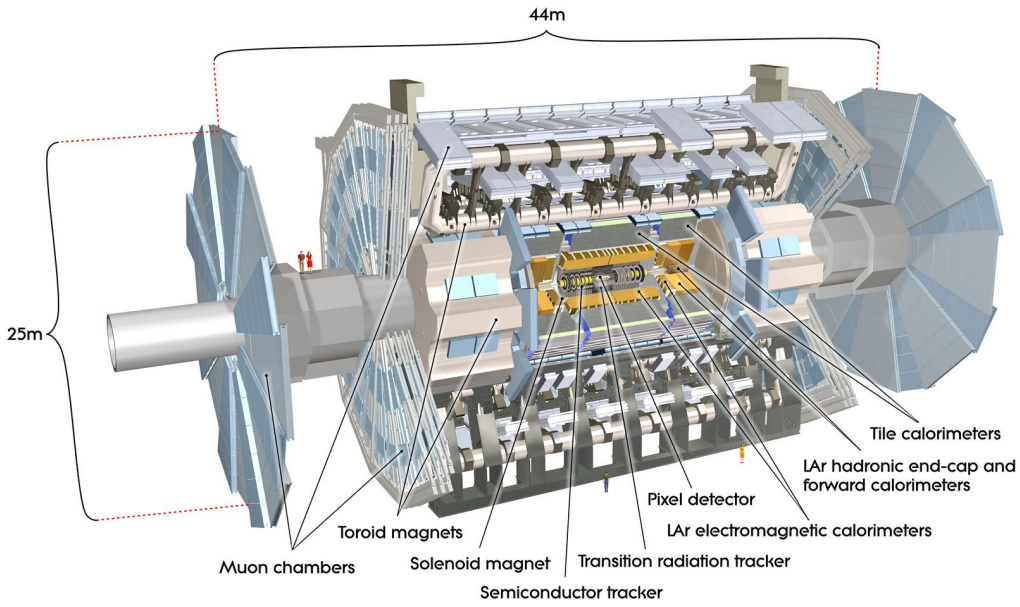


Figura 2.3: Detalhe do detector ATLAS. Extraído de [32].

2.2 O detector ATLAS

Dos quatro principais detectores acoplados ao LHC, o ATLAS (*A Toroidal LHC ApparatuS*) é o maior deles. Sua estrutura é detalhada na Figura 2.3. O detector é composto por diversas camadas, cada qual com sua função específica na reconstrução dos eventos de colisão.

Na camada mais interna, a aplicação de um campo magnético força partículas eletricamente carregadas a curvarem-se para um lado ou o oposto, dependendo do sinal da carga. Ao interagirem com o detector interno, a carga elétrica e o momento das partículas podem ser inferidos. Seguem-se os calorímetros, instrumentos utilizados para capturar e medir a energia das partículas. O calorímetro eletromagnético absorve elétrons e fótons, enquanto que o hadrônico absorve partículas formadas por quarks. Múons - uma espécie de elétron maciço, no entanto, não são absorvidos pelos calorímetros e sua detecção se dá em câmaras dedicadas na camada mais externa do detector e em suas extremidades. A detecção de múons é imprescindível para, entre outros fatores, permitir a dedução da presença de neutrinos, partículas que escapam completamente o detector e, com isso, possibilitar a completa reconstrução dos eventos.

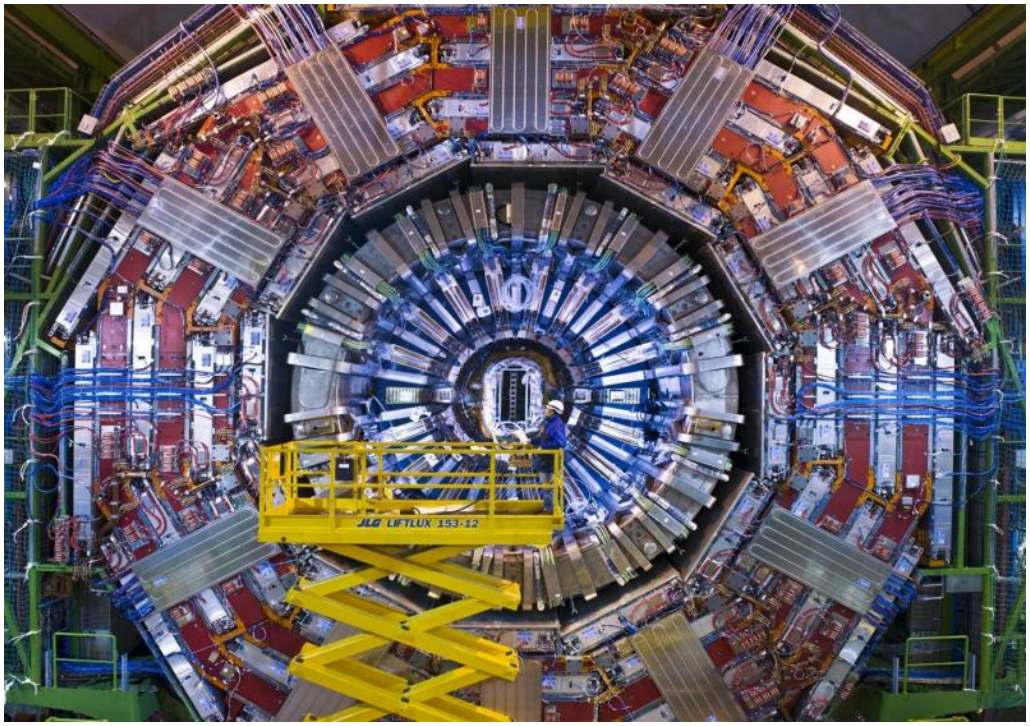


Figura 2.4: O detector CMS. Extraído de [33].

2.3 O detector CMS

O CMS (*Compact Muon Solenoid*) é, como o ATLAS, um experimento de propósito geral, isto é, seu projeto e operação não favorecem a detecção de um tipo especial de partícula. O detector, mostrado na Figura 2.4, investiga teorias além do Modelo Padrão na busca por pistas que possam ajudar a esclarecer questões fundamentais para as quais ainda não se tem resposta. Por exemplo, sabe-se que galáxias são gravitacionalmente mantidas por conta de um tipo exótico de matéria que não interage com a luz, tornando sua detecção extremamente difícil. CMS e ATLAS esperam elucidar a natureza da chamada matéria escura, possivelmente através da detecção de partículas além do Modelo Padrão.

Embora tenha os mesmos objetivos do ATLAS, o CMS se utiliza de diferentes técnicas e um campo magnético interno duas vezes mais intenso, conseguindo atingir resoluções semelhantes em uma estrutura mais compacta. O aparente antagonismo entre os dois detectores é importante dentro do método científico pois a independência de um experimento em relação ao outro substancia eventuais descobertas ao

Figura 2.5: Reconstrução de um evento no detector ALICE. Extraído de [16].

minimizar a influência de possíveis propensões de um método em relação ao outro. Resultados discrepantes convocam ambas as colaborações a reverem seus métodos e eventuais descobertas são mais confiáveis por serem produto de duas fontes distintas e independentes.

2.4 O detector ALICE

Todas as estruturas familiares que podemos observar no Universo são formadas por somente três partículas elementares: elétrons, quarks *up* e quarks *down*. Jamais se observou quarks em isolamento; há uma tendência natural em encontrá-los normalmente em pares ou trios[1], confinados por conta dos chamados *glúons*. Prótons e nêutrons são exemplos de partículas atômicas formadas por diferentes combinações de quarks.

Porém, acredita-se que a densidade energética no Universo primordial superava a

[1]Em Julho de 2015, o LHCb encontrou indícios da existência de um agrupamento de 5 quarks chamado pentaquark.

capacidade de confinamento de glúons, permitindo que quarks vagassem livremente. Este estado da matéria, conhecido como plasma quark-glúon, é de grande interesse pois sua própria existência e propriedades são questões chave na teoria conhecida como QCD (*Quantum Chromodynamics*), que descreve a chamada força forte. O ALICE (*A Large Ion Collider Experiment*), busca estudar este estado através da colisão de íons pesados. A Figura 2.5 mostra a reconstrução de um evento de colisão de íons de chumbo no detector. As temperaturas alcançadas de até 5.5 trilhões de graus Celsius [34] - 300 mil vezes mais quentes do que o núcleo do Sol, recriam as condições existentes milissegundos após o Big Bang.

2.5 O detector LHCb

Para cada partícula do Modelo Padrão, existe uma anti-partícula correspondente que exibe as mesmas características à exceção da carga elétrica. Para um elétron, por exemplo, deve existir um anti-elétron, ou pósitron, com propriedads idênticas porém com uma carga elétrica positiva. Anti-partículas podem combinar-se para formar estruturas da mesma forma que a matérica convencional. Um pósitron pode ser capturado por um antipróton para formar um átomo de anti-hidrogênio e mesmo galáxias inteiras formadas de anti-matéria são teoricamente possíveis.

No entanto, matéria e anti-matéria são aniquiladas quando em contato, liberando em radiação a energia referente à total conversão de suas massas segundo a equação $E = mc^2$. O *Big Bang* teria dado a origem a iguais quantidades de ambas as formas e apesar disso, o Universo parece composto quase que exclusivamente de matéria.

O LHCb (*LHC-beauty*) é um detector de propósito específico, concebido para investigar fenômenos que possam explicar quebras de simetrias no Universo primordial que impediram que matéria e anti-matéria se aniquilassem, por completo, deixando um universo dominado por radiação somente. Em termos simples, o LHCb tenta explicar porque estamos aqui.

A chave para este mistério, acreditam os físicos, está em padrões de decaimento de um tipo de quark conhecido como *beauty*, ou *b*. O LHCb, para tal, abre mão

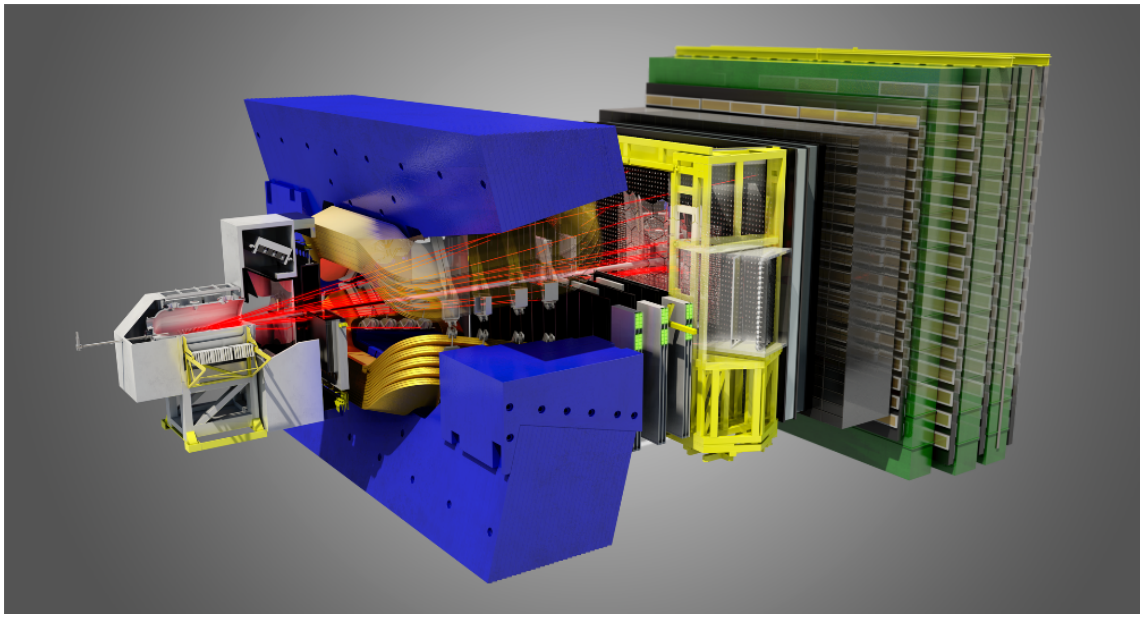


Figura 2.6: Detalhe da arquitetura assimétrica do detector LHCb. Extraído de [35].

de encapsular por completo o ponto de interação, como fazem os demais detectores, para focar em partículas *forward*, cujas trajetórias pouco se desvencilham do feixe de prótons após as colisões. A estrutura do detector é ilustrada na Figura 2.6, que mostra uma série de subdetectores posicionados paralelamente a partir do ponto de colisão.

2.6 Curiosidade e inovação

Especialmente em tempos de instabilidade econômica, a pesquisa fundamental é questionada, tornando-se um alvo primário de cortes em seus fundos. No entanto, diversos estudos comprovam a eficiência e rentabilidade de instituições científicas. Para cada dólar investido no programa espacial Apollo, estima-se que 14 foram injetados de volta na economia americana [36]. Avanços na medicina foram possíveis pela aplicação da tecnologia de detectores de partículas nos chamados *pet scans*, sendo diretamente responsáveis pelo diagnóstico e acompanhamento de milhões de pacientes. Na tecnologia da informação, o CERN foi o berço da *World Wide Web*, efetivamente transformando a forma como a sociedade vive, se comunica, trabalha e inova em apenas pouco mais de três décadas desde a sua invenção.

Figura 2.7: Físicos observam primeiras colisões no detector ATLAS. Extraído de [37].

A Figura 2.7 captura o momento em que físicos testemunharam as primeiras colisões na sala de controle do ATLAS. O entusiasmo daquele instante, muito além do alívio em justificar os 6.5 bilhões de euros investidos no LHC, representa o sucesso de uma colaboração internacional de mais de 88 países. Diferentes línguas, culturas e realidades econômicas, unidas pelo objetivo comum de melhor compreender o Universo em que vivemos.

Capítulo 3

Sistemas *Web* dos Experimentos

Combinados, os quatro grandes experimentos do LHC representam um esforço colaborativo de mais de 10 mil físicos, engenheiros e estudantes. Como é de se esperar, a coordenação de colaborações deste porte não é sem desafios. Neste capítulo, exploramos as gerências dos experimentos e os sistemas que foram criados para dar suporte as suas operações.

Enquanto que o intercâmbio acadêmico entre os diversos institutos de pesquisa foi fator imprescindível para que os experimentos pudessem ser construídos e operados, as diferentes tecnologias, práticas e modelagens empregadas no armazenamento de dados de cada um dos componentes dos detectores impuseram desafios à integração dos dados.

A partir da identificação da necessidade de uma plataforma de integração de fontes heterogêneas, a Universidade Federal do Rio de Janeiro propôs, em colaboração com o ATLAS, o sistema de recuperação de dados Glance em 2003. O caso de sucesso do ATLAS deu bases a sua expansão para o ALICE e LHCb, sendo empregado atualmente como fornecedor de dados que são expostos em 21 sistemas.

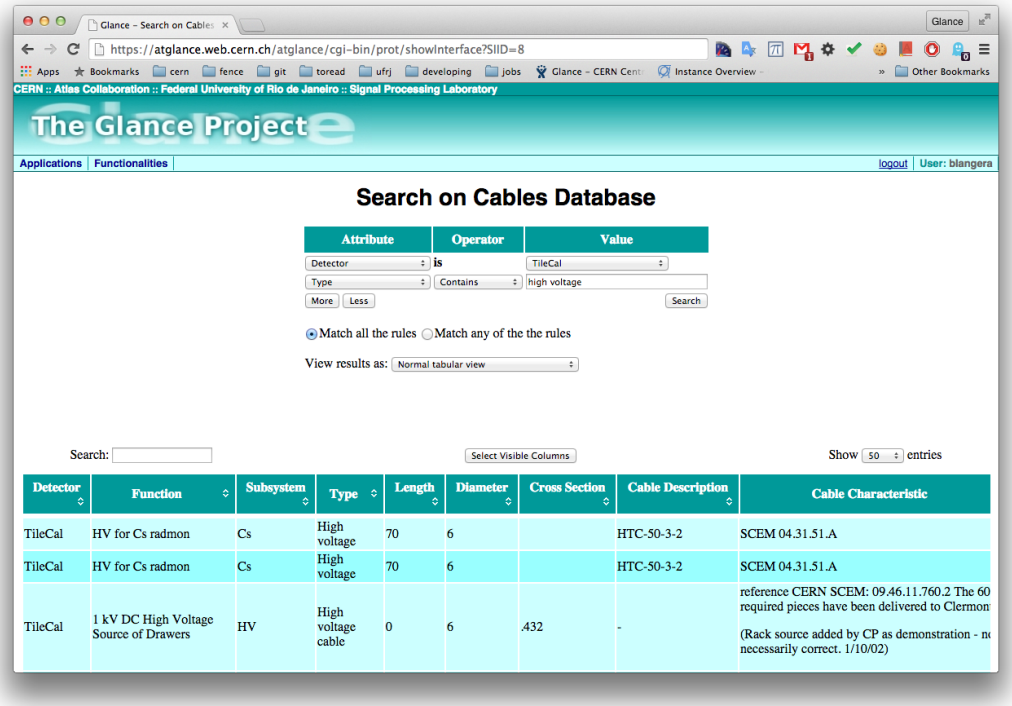


Figura 3.1: O sistema Glance aplicado na recuperação de cabos do ATLAS.

3.1 A plataforma Glance

O Glance se propõe a ser uma camada de abstração em cima de distintas tecnologias, separando o usuário das particularidades de cada banco de dados [18]. De fato, sem qualquer conhecimento de linguagens de consulta[1], usuários são capazes de criar **interfaces de busca** acessíveis via *Web*, sem a necessidade da instalação de qualquer programa (à exceção do navegador, é claro). Cada coluna das tabelas integradas é automaticamente transformada em parâmetros de busca que podem ser combinados para filtrar o conteúdo dos bancos de dados de acordo com a necessidade do usuário. A Figura 3.1 ilustra este conceito aplicado à recuperação de dados dos cabos do ATLAS, uma das aplicações historicamente mais relevantes do Glance.

A visualização dos resultados em um navegador *Web* é de extrema conveniência para usuários gerais dos sistemas. No entanto, o Glance permite que interfaces de

[1]Em programação, linguagens de consulta correspondem a sintaxes específicas para a recuperação e manipulação de dados armazenados em um banco de dados como, por exemplo, SQL (*Structured Query Language*).

busca sejam acionadas programaticamente, tendo seus resultados transmitidos em formatos como XML (*Hypertext Markup Language*) ou JSON (*JavaScript Object Notation*), podendo ser incorporados por desenvolvedores em diferentes aplicações. Em termos técnicos, o Glance fornece uma API (*Application Program Interface*), que pode ser consumida por outras aplicações na formação de suas interfaces.

O Glance, portanto, ao simplificar o acesso às informações contidas nos bancos de dados da Coordenação Técnica do ATLAS, permitiu o desenvolvimento de 17 sistemas *Web* cujas aplicações variam desde a gerência de contratos de membros até o monitoramento de níveis de radiação dos equipamentos na caverna.

ALICE e LHCb seguiram o exemplo do ATLAS e, hoje, cada detector conta com dois sistemas baseados no Glance que foram adaptados para suas necessidades particulares. Estes sistemas serão descritos a seguir.

3.2 Sistemas desenvolvidos

Esta seção descreve brevemente alguns dos mais importantes sistemas desenvolvidos para o ATLAS, citando os casos em que as aplicações tenham sido adaptadas para o ALICE e/ou LHCb.

As aplicações foram, em sua maioria, escritas em PHP (*Hypertext Preprocessor*), linguagem de programação especialmente projetada para o desenvolvimento de aplicações *Web*. A incorporação de dados oriundos do Glance, por sua vez, ocorre no lado do cliente através de chamadas assíncronas para sua API. Esta tecnologia, conhecida como AJAX (*Asynchronous JavaScript And XML*), é amplamente empregada em modernas aplicações por permitir que dados sejam buscados no servidor e exibidos para o usuário de maneira transparente, sem a necessidade de carregar-se a página por completo.

3.2.1 Membership

A gerência de membros, contratos, institutos e agências financiadoras do ATLAS é conduzida pelo *Membership*, mostrado na Figura 3.2a. O sistema apóia, ainda, o processo de qualificação de autores através da submissão e acompanhamento de um projeto que culmina na inclusão do membro na lista de autores do ATLAS. Atualmente, 2862 físicos e engenheiros assinam conjuntamente todos os artigos científicos produzidos pela colaboração.

Muitas das informações exibidas no *Membership* são tratadas com um certo grau de confidencialidade e o sistema se utiliza de uma estratégia conhecida como RBAC (*Role-based Access Control*) [38] para garantir que informações protegidas sejam visíveis somente para usuários credenciados. A autenticação envolve uma consulta ao banco de dados acerca dos grupos aos quais o usuário pertence e os privilégios associados que o sistema deve emitir. Os privilégios são armazenados em memória e persistem no servidor por até 8 horas, quando a chamada sessão do usuário é expirada por motivos de segurança. Um exemplo de como o *Membership* verifica se um usuário hipotético pode acessar a data de qualificação dos autores do ATLAS é mostrado, de forma simplificada, no Código 3.1.

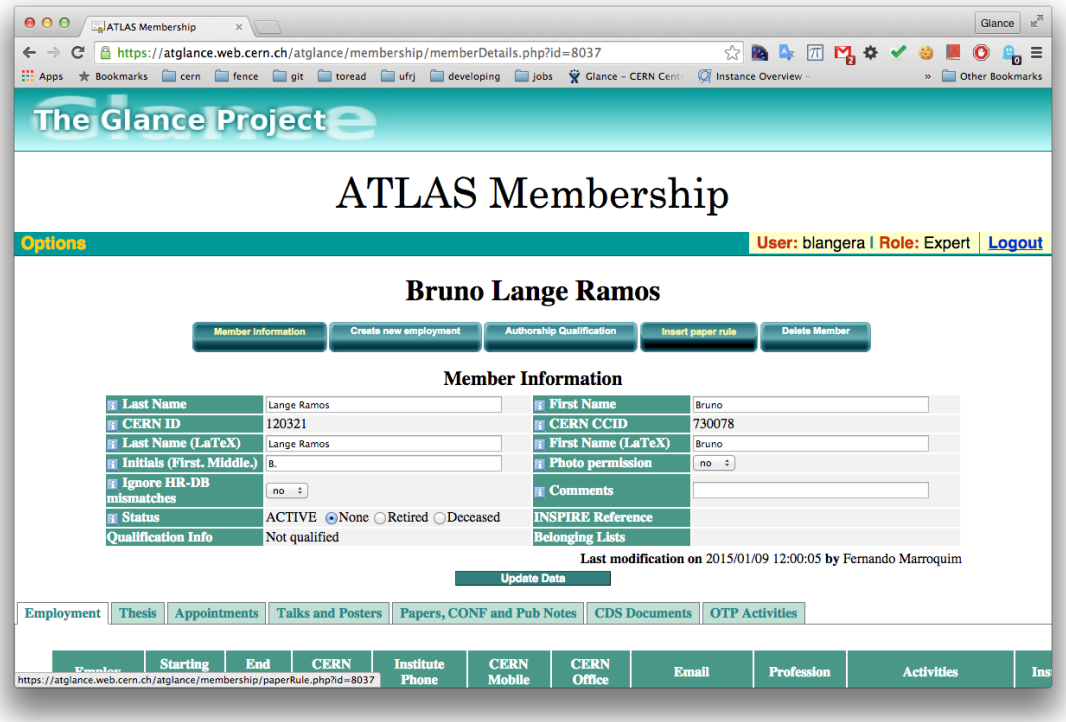
Código 3.1: Aplicação da abordagem RBAC no *Membership*.

```
<?php if ($_SESSION['QUAL_DATE'] == 'granted') {
    // user can see qualification date
}
```

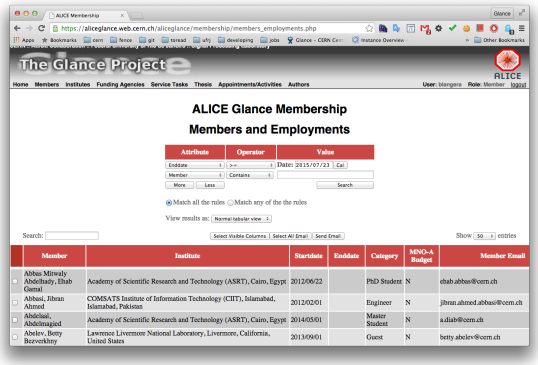
A eficaz integração dos dados de membros do ATLAS chamou a atenção de ALICE e LHCb, que hoje utilizam versões adaptadas do *Membership* original para a gerência de seus membros. Capturas de tela dos respectivos sistemas são mostrados nas Figuras 3.2b e 3.2c.

3.2.2 Appointment

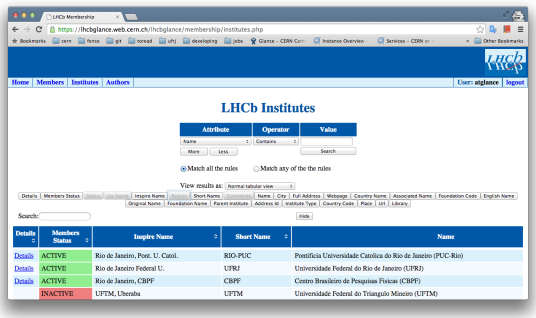
O controle de cargos gerenciais no ATLAS é intermediado pelo *Appointment*. O sistema permite a edição de diversos parâmetros atrelados a cada cargo e registra novos mandatos conforme acordados pelo Conselho da colaboração.



(a) ATLAS Membership.



(b) ALICE Membership.



(c) LHCb Membership.

Figura 3.2: Membership: sistemas de gerência de membros e institutos de ATLAS, ALICE e LHCb.

A nomeação de membros para cargos gerenciais implica em responsabilidades que normalmente envolvem a manipulação de dados sensíveis. Por esta razão, o sistema é convenientemente utilizado para o controle de acesso a interfaces protegidas ao definir, dentro da abordagem RBAC, um grupo a cada papel dentro da gerência do ATLAS.

3.2.3 Analysis

O processo de publicação de artigos científicos no ATLAS envolve os esforços coordenados de diversos membros da colaboração. O sistema *Analysis*, subdividido em três subsistemas - *Papers*, *Conference Notes* e *Publication Notes*, fraciona o procedimento em etapas e delega privilégios a grupos específicos que são acionados quando chega sua vez de agir.

Para facilitar a colaboração entre os membros de cada grupo de uma publicação, o sistema se utiliza de uma API disponibilizada pelo departamento de Tecnologia da Informação do CERN para automaticamente criar os chamados *e-groups*, associações de membros sob um endereço de e-mail que são vastamente utilizados por todo o CERN no controle de acesso de usuários a páginas *Web*, arquivos e diretórios DFS (*Distributed File System*) e eventos como reuniões e palestras.

A conclusão de etapas normalmente envolve a notificação de membros responsáveis pela preenchimento dos dados do passo subsequente. O sistema se encarrega, para tal, do envio automático de e-mails para recipientes específicos. Administradores do sistema são capazes de modificar o conteúdo dos e-mails através de interfaces de edição conforme mostrados na Figura 3.3.

3.2.4 Speakers

Uma das mais prestigiosas atividades no mundo acadêmico consiste na apresentação de resultados em conferências internacionais. O ATLAS se apóia em um comitê (*Speakers Commitee*) especialmente designado para selecionar candidatos qualificados a representar a colaboração, procurando sempre assegurar uma distribuição igualitária de palestras entre países e institutos do ATLAS.

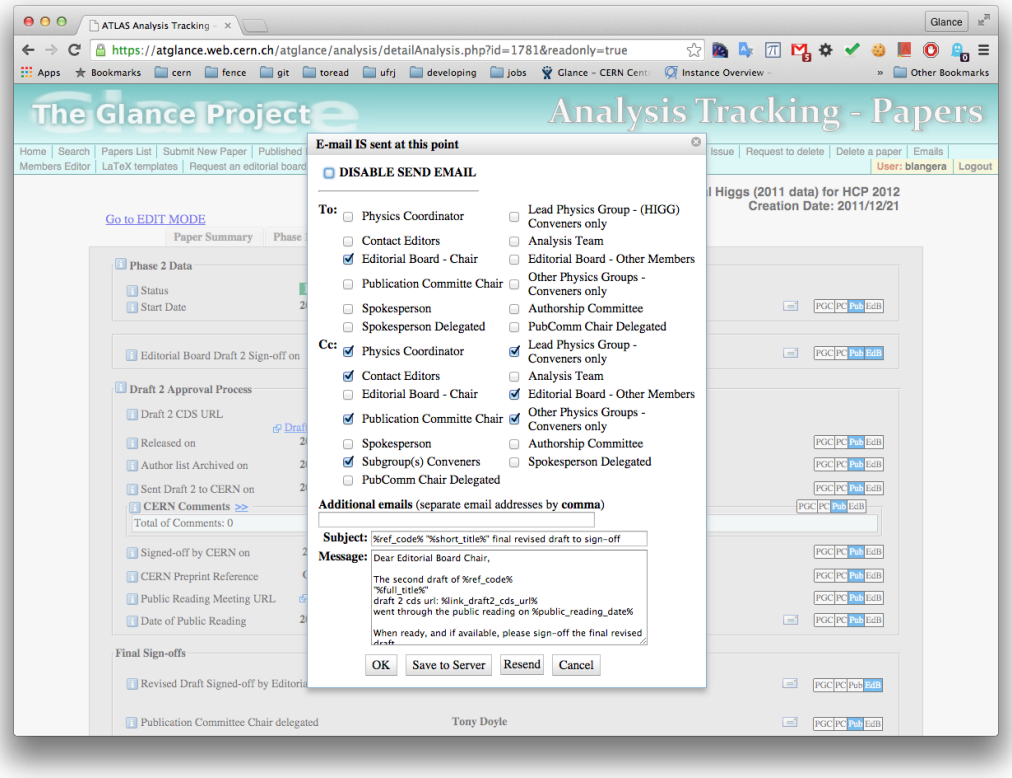


Figura 3.3: Configuração de emails do *Analysis Papers*.

Membros da colaboração podem ser nomeados por coordenadores de projeto, representantes de institutos e ocupantes de cargos gerenciais em geral. As nomeações são analisadas pelo SCAB (*Speakers Committee Advisory Board*), que produz uma lista de possíveis candidatos para a consideração do *Speakers Commitee*. Atualmente, existem mais de 8200 nomeações atribuídas a 3618 membros do ATLAS. A tarefa de reduzir este universo a um único representante não é das mais simples e, para tal, o sistema *Speakers* oferece suporte a este processo ao disponibilizar mais de 40 parâmetros de busca de nomeações, além de gerar gráficos e histogramas para a availação da distribuição de palestras.

3.2.5 Conferences and Talks

Uma vez que o *Speakers Committe* recebe as sugestões do SCAB, é seu papel encaminhar os convites aos membros selecionados e registrar as palestras aceitas e rejeitadas. A interface entre os membros do comitê e os dados armazenadas no

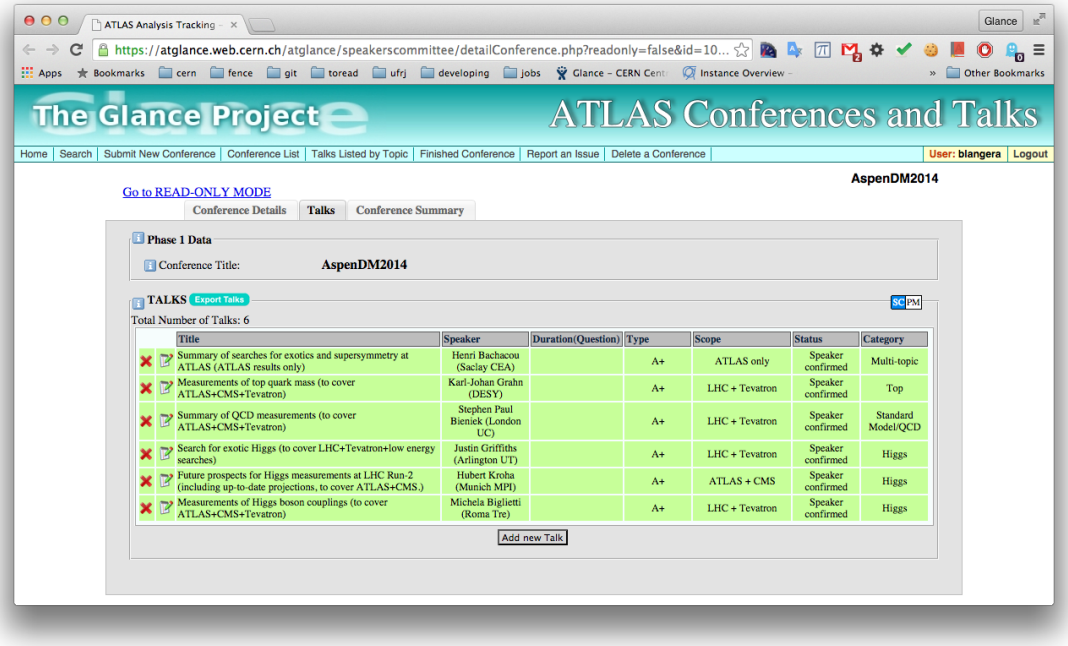


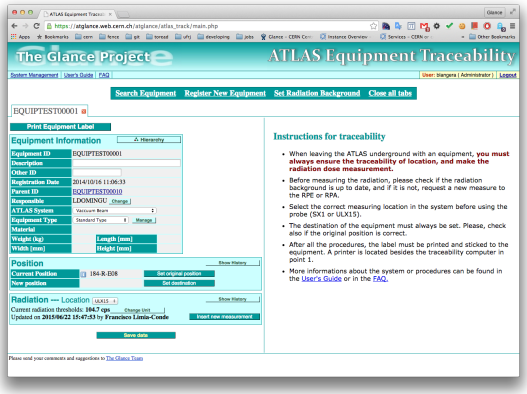
Figura 3.4: Captura de tela do sistema Conferences & Talks.

banco de dados é feita pelo *Conferences and Talks*, mostrado na Figura 3.4. Nele, conferências e as palestras associadas podem ser registradas e modificadas. Interfaces de busca subsiadas pelo Glance permite que as mais de 7 mil palestras do ATLAS sejam rapidamente encontradas de acordo com seus tópicos e classificações.

O sistema, a modelo do *Analysis*, se utiliza de uma série de modelos de e-mails pré-definidos que podem ser acionados para facilitar o processo de convite e aceitação de palestras por membros da colaboração. De fato, há outras diversas similaridades entre os dois sistemas como o fracionamento de longos processos em etapas, a delegação de papéis a diferentes grupos e alternância entre modos de visualização em leitura e escrita.

3.2.6 Traceability

A curvatura das partículas ao longo do LHC expõe os detectores a um tipo de radiação conhecida como radiação síncroton. O manuseio dos equipamentos expostos em períodos de manutenção precisa ser conduzido de acordo com as legislações suiças e francesas. No ATLAS e LHCb, procedimentos de remoção, transporte, reparo



(a) ATLAS Traceability.

(b) LHCb Traceability.

Figura 3.5: Traceability: sistemas de monitoramento da radiação de equipamentos.

e descarte dos equipamentos do detector são auxiliados pelo *Traceability*, conforme ilustrados nas Figuras 3.5a e 3.5b, respectivamente. Os sistemas mantém um completo rastreamento do estado de cada equipamento, incluindo suas localizações e medidas de níveis de radiação.

3.2.7 DSS Viewer

O sistema de segurança DSS (*Detector Safety System*) [39] monitora a operação do ATLAS através de inúmeros sensores. Alarmes são disparados quando o detector é exposto a situações de risco - normalmente causadas por *quenches* - onde a supercondutividade de imãs é comprometida por aumentos de temperatura, fazendo com que enormes quantidades de corrente sejam repentinamente sujeitas a uma resistência elétrica, podendo causar incêndios e interromper a operação do LHC por meses. O *DSS Viewer* acessa o estado dos alarmes e informa suas condições de ativação, latência e procedimentos de emergência.

3.2.8 RackWizard

O *RackWizard* foi originalmente concebido para o CMS [40] como uma solução gráfica para a configuração da eletrônica dos componentes anexos do detector. O sistema foi adaptado para o ATLAS [41] e propõe uma organização hierárquica para

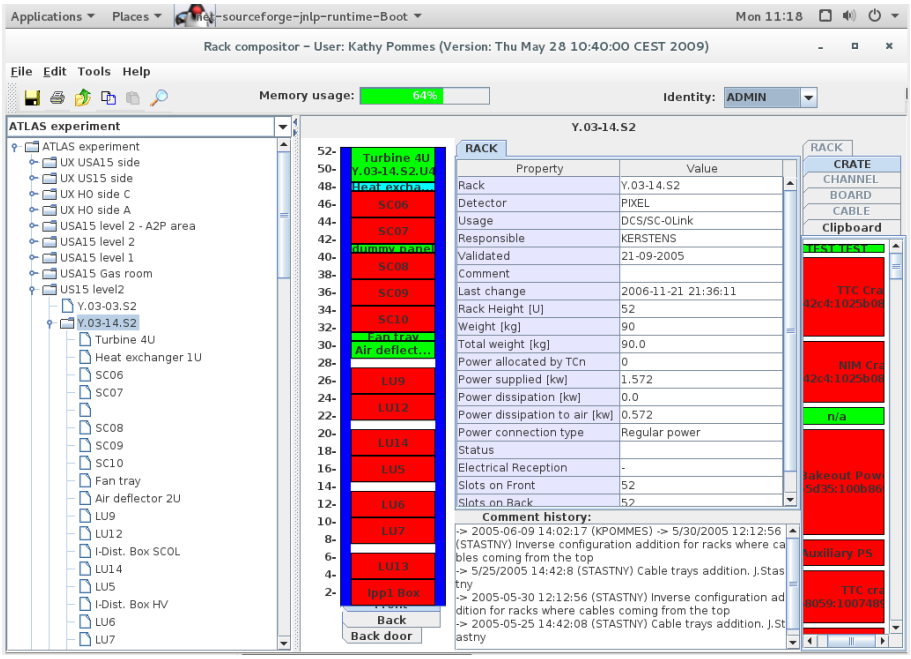


Figura 3.6: Conteúdo de um *rack* do ATLAS sendo exibido pelo *RackWizard*.

a localização de zonas, *racks*, prateleiras (*crates*), *boards* e cabos, que são dispostos de forma intuitiva em um arranjo similar ao explorador de diretórios em um sistema operacional conforme ilustrado na Figura 3.6.

Ao contrário dos demais sistemas, o *RackWizard* foi desenvolvido como um aplicativo Java que é instalado e executado localmente nos computadores dos usuários. As exigências do sistema tornaram esta abordagem mais atraente visto à limitações de navegadores, à época, em suportar nativamente soluções gráficas.

3.3 Evolução dos sistemas

À exceção do *RackWizard*, o Glance foi utilizado como a ferramenta de recuperação de dados por trás dos sistemas de suporte à gerência de ATLAS, LHCb e ALICE. No entanto, embora haja diversos aspectos similares entre as aplicações, o desenvolvimento de cada sistema se deu de maneira independente.

Na Figura 3.7, a área de cada retângulo é diretamente proporcional ao número de linhas de código (LOC) dos maiores sistemas de suporte à gerência do ATLAS. Estes 7 sistemas respondem por aproximadamente 90% de todos os pedidos de implementa-

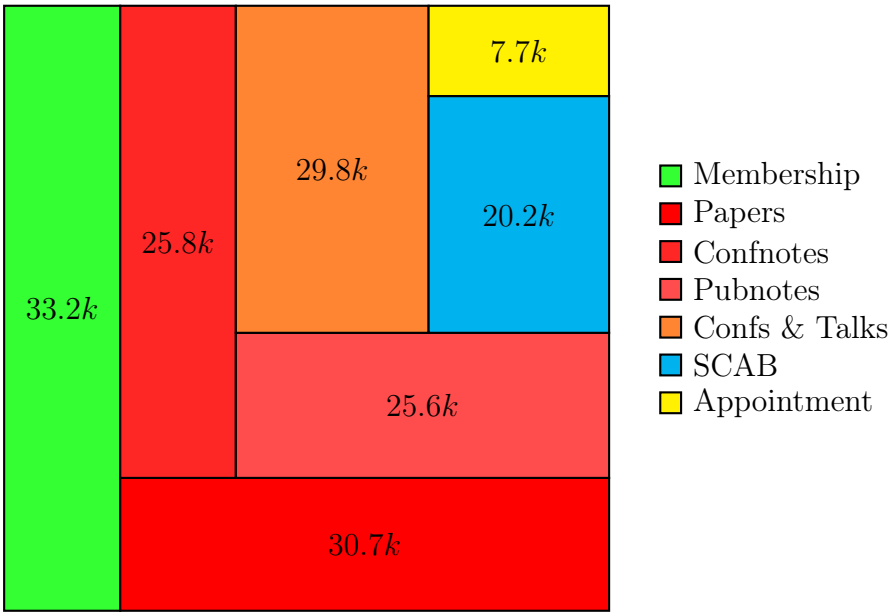


Figura 3.7: Análise LOC dos maiores sistemas de suporte à gerência do ATLAS.

ção de melhorias e alterações de requisitos que chegam ao `Atlas.Glance@cern.ch`, endereço de contato para o suporte técnico das interfaces. Desde Agosto de 2011, o grupo de desenvolvedores recebeu uma média de 67 emails por mês.

Sem o suporte de uma plataforma comum onde alterações e melhorias possam ser imediatamente propagadas, a manutenção dos sistemas envolve constantes esforços duplicados, já que a alteração de requisitos precisa ser imposta em diferentes trechos de código-fonte para diferentes interfaces. Em especial, os sistemas *Analysis* (*Papers*, *Confnotes* e *Pubnotes*), em tons vermelhos na Figura 3.7, são normalmente sujeitos as mesmas alterações, uma vez que compartilham muito dos requisitos definidos pelo ATLAS *Physics Office*.

A exemplo dos cargos gerenciais nos experimentos do LHC, o time de desenvolvedores, composto em sua maioria por estudantes de engenharia da UFRJ, também é caracterizado por uma alta rotatividade. Há, portanto, uma etapa no processo de manutenção e implementação de novas funcionalidades que envolve a leitura e compreensão da documentação e códigos-fonte existentes, o que invariavelmente acarreta atrasos na alteração dos sistemas. Desta forma, o estabelecimento de padrões de documentação e controle de versão do código também se faz necessário.

Estes problemas serão analisados mais detalhadamente no capítulo seguinte para definir as especificações da solução proposta como tema desta projeto: uma plataforma integradora, um *framework* para a geração de interfaces *Web* especialmente projetado para atender às necessidades das gerências dos experimentos do LHC.

Capítulo 4

Proposta

O CERN é caracterizado por um conjunto de fatores para os quais, sob a perspectiva da Engenharia de Requisitos, não há ferramentas disponíveis que se proponham a conciliar o cumprimento das demandas do experimento bem como os interesses específicos de cada um dos grupo que compõem as colaborações.

No passado, os eventuais desafios causados pela distribuição geográfica e a heterogeneidade de culturas, áreas de atuação e recursos foram contornadas pela invenção da *World Wide Web*, permitindo a rápida comunicação entre centros de pesquisas sem exigir de seus usuários conhecimentos específicos acerca dos protocolos empregados.

Este capítulo analisa o ambiente em que se encontram os experimentos do CERN, dando bases ao levantamento de requisitos para a elaboração de uma nova abordagem na concepção dos sistemas *Web*.

4.1 Análise do problema

Durante décadas e, em especial, antes da invenção da *Web*, a Engenharia de Requisitos era vastamente tida como uma etapa fixa precursora ao desenvolvimento de software onde as especificações eram definidas e acordadas [5]. Embora permitisse que a composição do código-fonte fosse conduzida de forma ordenada e controlada,

esta abordagem deixava poucas margens para a implementação de mudanças, sendo, desde o início da década de 90, amplamente considerada como uma estratégia inapropriada [42]. O reconhecimento da característica flexível dos requisitos como uma prerrogativa de clientes e *stakeholders* permite a sua classificação em quatro principais categorias. São elas:

- **Requisitos estáveis**

  Não obstante à percepção de que requisitos dinâmicos há muito se tornaram a norma, há sempre um conjunto de requisitos básicos que têm suas origens associadas às atividades características de uma empresa ou instituição e que, portanto, perduram durante todo o ciclo de vida do software. Às especificações que não estão sujeitas a modificações dá-se o nome de requisitos estáveis.

- **Requisitos mutáveis**

  Refere-se à classe de requisitos que se adaptam como resposta a mudanças no ambiente em que se situam. Flutuações do mercado financeiro, fatores políticos e mudanças na legislação são exemplos de fatores externos que podem influenciar a especificação de softwares. Há, também, elementos internos capazes de impor alterações nos requisitos. Em especial, a alternância de cargos gerenciais normalmente implica em mudanças nos comportamentos dos sistemas à medida que diferentes visões e propostas são colocadas em prática.

- **Requisitos emergentes**

  Por vezes, o processo de desenvolvimento de software é interrompido antes mesmo de uma única linha de código ter sido escrita pela dificuldade de clientes e *stakeholders* em sequer produzir uma lista inicial de requisitos. Seja por incertezas na definição dos objetivos, falhas em compreender de que forma as tecnologias disponíveis podem alcançá-los ou mesmo pela falta de um consenso, nem sempre os requisitos podem ser levantados por completo quando uma solução é proposta. Muitos acabam emergindo durante a implementação do projeto, sendo classificados, portanto, como requisitos emergentes.

- **Requisitos consequentes**

  Quando soluções eficazes são disponibilizadas aos usuários, a experiência ob-

tida invariavelmente implica na descoberta de novas formas de trabalho e abre caminho para a proposta de abordagens mais ambiciosas ao prover novas perspectivas sobre as atividades e dados processados. Como resultado, cria-se expectativas para a implementação de funcionalidades que sequer podia-se prever em um estágio inicial. A evolução do software é, neste cenário, conduzida por possibilidades introduzidas por sua própria tecnologia, caracterizando os novos requisitos como consequentes.

4.1.1 Mapeamento do ambiente

Por tratar-se de um ambiente científico, o CERN é caracterizado por uma evolução natural que se reflete em constantes adaptações nos sistemas dos experimentos. Além disso, devido a certas particularidades do modelo colaborativo de grande porte no qual os experimentos do LHC se enquadram, foi possível identificar a predominância de dois tipos de requisitos que guiam a evolução dos sistemas de suporte.

O comissionamento e construção dos detectores são fruto de um esforço colaborativo de diversos centros de pesquisa no mundo. Para que os diversos países envolvidos participem também da administração dos experimentos, os muitos cargos gerenciais são sujeitos a uma alta rotatividade. No ATLAS, para o qual este projeto foi inicialmente desenvolvido, tantas são as posições como a alternância entre elas que um sistema dedicado para o acompanhamento dos cargos - *Appointment*, como descrito na seção 3.2.2, foi proposto e implementado.

Soma-se à característica progressista de ambientes científicos com a diversidade de práticas, necessidades e tecnologias empregadas em colaborações internacionais - acentuada, ainda, por frequentes mudanças em posições administrativas, para então concluirmos que os sistemas computacionais dos experimentos encontram-se em ambientes caracterizados como *turbulentos* [43]. As aplicações se adaptam para observarem mudanças no meio em que operam, configurando a influência de requisitos **mutáveis**.

Paralelamente, observa-se frequentes incertezas quanto a definição dos objetivos que os sistemas devem alcançar. Dependendo da complexidade das aplicações, a

antecipação de certos cenários torna-se complicada, inviabilizando a possibilidade de especificar-se como as interfaces devam reagir dentro destas situações. Além disso, novos ocupantes de cargos administrativos nem sempre dispõe do tempo para familiarizarem-se com as soluções existentes e encontram dificuldades em especificar novas funcionalidades. Com a incerteza dos objetivos em etapas iniciais de desenvolvimento, requisitos **emergentes** se fazem comuns a medida que a solução vai tomando forma.

4.1.2 O cenário atual

A volatilidade intrínseca destas classes de requisitos, identificadas como as mais proeminentes no ambiente não só no ATLAS mas também no ALICE e LHCb, faz com que os sistemas desenvolvidos para os experimentos sejam constantemente sujeitos a alterações, de forma que possam enquadrar-se às especificações correntes. Como consequência, a manutenção dos softwares existentes envolve esforços consideráveis, primeiramente pelo grande volume de pedidos de alteração de funcionalidades existentes, bem como de implementação de novas. Mais de 2 mil ocorrências foram registradas em pouco menos de 4 anos, período no qual a equipe de desenvolvedores recebeu, em média, 67 emails por mês.

A força de trabalho responsável pelo suporte técnico e desenvolvimento de novas tecnologias é composta por alunos da Escola Politécnica da UFRJ e, portanto, a equipe de desenvolvedores é caracterizada por um certo dinamismo à medida que estudantes concluem suas colaborações e dão lugar a novos membros. O projeto em muito ganha com a renovação de idéias mas os processos de aprendizado das tecnologias já empregadas e familiarização com os diversos sistemas existentes demandam tempo. Há, portanto, um atraso inevitável até que soluções inovadores possam ser propostas e implementadas pelos alunos.

A situação é agravada pelo relativo isolamento dos sistemas. Embora as interfaces compartilhem uma considerável parte de suas funcionalidades como, por exemplo, a autenticação de usuários, rotinas de validação e proteção de dados sensíveis, muito do código-fonte por trás destas implementações é encontrado concomitantemente em

diversas aplicações. Aproximadamente 173 mil linhas de código compõe os 7 maiores sistemas de suporte ao ATLAS e, frequentemente, intervenções precisam ser aplicadas repetidamente em diferentes trechos de código para que mudanças de requisito sejam colocadas em vigor. Em especial, os sistemas *Analysis* são caracterizados pela semelhança com que processam as diferentes publicações gerenciadas pelo ATLAS *Physics Office*. Sem a garantia de uma uniforme propagação das regras do sistema por todas as interfaces em seu escopo, ambiguidades podem surgir e comprometer a coesão das informações expostas.

Este problema foi inicialmente detectado na duplicação de *scripts* que são executados do lado do cliente[1] para que as informações contidas nos bancos de dados pudessem ser incorporadas às interfaces mediante inúmeras solicitações assíncronas à API do Glance. O desenvolvimento da biblioteca *Glance.js* padronizou esta comunicação pelo intermédio de um único agente, em muito reduzindo o número total de linhas de código sem qualquer efeito colateral que pudesse comprometer a funcionalidade das aplicações. Pelo contrário, esta estratégia tornou a manutenção dos sistemas consideravelmente mais simples e permitiu que novos recursos pudessem ser incorporados dentro de maneira estruturada, estando prontamente disponível a todos os desenvolvedores.

Para a geração e processamento de dados no lado do servidor, no entanto, não há uma plataforma unificadora que possa auxiliar o desenvolvimento e evolução conjuntos das aplicações. Seguiu-se à identificação desta necessidade uma análise de diversas ferramentas existentes que pudessem ser empregadas. Concluiu-se que, embora haja modernas plataformas de auxílio ao desenvolvimento de sistemas em PHP como *Symfony*, *Laravel* e *ZEND*, nenhuma das arquiteturas disponíveis engloba por completo o particular conjunto de requisitos que compõe os sistemas em questão e que uma solução eficaz teria de combinar variadas abordagens.

[1] Aplicações *Web* são caracterizadas pela dicotomia entre os lados do **servidor**, responsável por servir as páginas, e do **cliente**, responsável pela apresentação do conteúdo, normalmente um navegador.

4.2 Levantamento de requisitos

O mapeamento do ambiente em que se situam os diversos sistemas de auxílio às administrações dos experimentos do LHC deu base para as especificações da plataforma que assumirá o papel de criar e guiar a evolução das interfaces. A idéia central gira em torno da redução dos esforços para a gerência de requisitos mutáveis e emergentes, realocando a atenção da equipe de desenvolvedores para os bem-vindos requisitos consequentes como resultado do engajamento dos desenvolvedores na pesquisa, desenvolvimento e integração de tecnologias inovadoras que expandem as capacidades dos sistemas. Para que isso seja possível, o *framework* proposto deve observar uma série de requisitos, listados e detalhados em seguida.

- **Rápida resposta a mudanças de requisitos**

De maneira geral, os requisitos do *framework* devem alinhar-se à natureza volátil das especificações dos sistemas de suporte às gerências dos experimentos. Para isto, a solução proposta deve incorporar ferramentas de modo a tornar a resposta da equipe de desenvolvedores frente a alterações de requisitos mais eficiente. Idealmente, mudanças mais simples sequer devem ser intermediadas por um desenvolvedor. Usuários credenciados, através de intuitivas interfaces de edição de regras, devem ser capazes de alterar o comportamento dos sistemas sem necessariamente qualquer conhecimento de linguagens de programação.

- **Segurança dos dados**

A preocupação com a proteção de dados sensíveis se faz presente em praticamente todas as aplicações. Nomeações de palestrantes, por exemplo, são de confidencialidade dos comitês incumbidos da seleção de candidatos. Datas de término de contrato e períodos de inclusão na lista da autores são também normalmente mantidos em sigilo, sendo acessáveis somente pelo próprio usuário ou pelo comitê de qualificação. Em resumo, as informações contidas nas interfaces devem ser filtradas de acordo com as credenciais do usuário, levando sempre em consideração a perspectiva de mudanças nas regras associadas. Assim sendo, a plataforma deve identificar similaridades

nas rotinas de autenticação de usuários de diferentes sistemas para viabilizar uma estratégia centralizada, segura e flexível da proteção dos dados das colaborações.

- **Conformidade dos dados**

Em sua mais básica descrição, sistemas *Web* fornecem interfaces a um ou mais bancos de dados. Como esta interação ocorre nos dois sentidos, isto é, as aplicações proveem não somente a capacidade de visualização (leitura) dos dados armazenados, mas também permite que informações sejam criadas ou alteradas (escrita), o controle dos dados submetidos pelos usuários é, similarmente, de vital importância. Como muitos dos requisitos envolvem a especificação de regras a respeito de quais usuários podem inserir determinadas informações, além de diversas restrições ao próprio formato dos dados, espera- se que o *framework* possa impor essas diretrizes a cada um dos chamados *inputs*, instalando rotinas de validação primariamente no lado do servidor e, quando aplicável, também no cliente.

- **Reprodutibilidade de erros**

Embora não haja, na prática, aplicações completamente isenta de falhas, eventuais experiências negativas de usuários podem ser atenuadas se houver um acompanhamento imediato da equipe de desenvolvedores. A plataforma deve, portanto, contar com um sistema integrado de alerta ao suporte técnico quando detecta comportamentos anormais, contendo todas as informações necessárias para que o problema possa ser ser reproduzido.

Ao mesmo tempo, é importante que o *framework* reaja de maneira elegante e informativa quando uma página não puder ser corretamente produzida. Usuários não devem jamais ser expostos a mensagens de erros específicas das tecnologias empregadas. Um erro da forma

```
SQL Error: ORA-00001: unique constraint (ATLAS_STAGE.SYS_01) violated
*Cause:     An UPDATE or INSERT statement attempted to insert a
            duplicate key. For Trusted Oracle configured in DBMS
```

```
            MAC mode, you may see this message if a duplicate entry
            exists at a different level.
*Action:    Either remove the unique restriction or do not insert
            the key.
```

não só é inconveniente ao usuário como também expõe certos detalhes sobre o banco de dados que, por questões de segurança, não podem ser feitos públicos. Alternativamente, estes erros devem ser internamente captados e "traduzidos" quando apresentados aos usuários.

- **Acessibilidade**

Os sistemas são projetados para serem acessados por membros de colaborações internacionais em mais de 80 países. A língua oficial deve ser o inglês, com suporte para outras línguas desejável porém não mandatório. A todo instante, ao menos 1 servidor deve encontrar-se operacional, sendo recomendada a utilização de instâncias redundantes. No caso de apenas 1 servidor disponível, serviços de manutenção jamais podem deixar os sistemas inacessíveis por mais de 45 minutos. Finalmente, os sistemas resultantes devem ser operacionais tanto em computadores (*desktops* ou *notebooks*), quanto em dispositivos móveis como *smartphones* e *tablets*.

- **Performance**

Naturalmente, o *framework* deve ser otimizado para a geração de páginas *Web* de maneira mais eficiente possível de forma a não comprometer a experiência do usuário. Como demonstrado em [44], o tempo de carregamento de páginas *Web* tem uma relação inversamente proporcional com o engajamento do usuário. Experiências negativas tendem a acarretar um impacto maior e devem, portanto, ser minimizadas. Para isso, além de uma boa disponibilidade de recursos físicos nos servidores, é importante que a plataforma seja modular o bastante para permitir incorporação de novas tecnologias sem grandes intervenções em seu núcleo.

- **Documentação e controle de qualidade de código-fonte**

Para que novos desenvolvedores estejam aptos a propor soluções inovadoras e integrá-las o mais rápido possível, é preciso reduzir-se o tempo investido na compreensão das regras de cada colaboração e atenuar a curva de aprendizado das tecnologias já empregadas. Com este objetivo, a documentação do código se faz importante a ponto de ser listada com um requisito dentro da solução proposta. Além disso, a definição de um estilo de codificação padrão também é exigido para que a eficiência na assimilação de implementações não seja comprometida por preferências pessoais de cada desenvolvedor.

4.3 Proposta

A solução proposta, tema deste projeto de graduação, consiste na especificação e implementação de um *framework* para o desenvolvmento de sistemas *Web* de suporte às gerências dos experimentos do LHC. O estabelecimento de uma plataforma única reduz esforços repetidos e permite que a evolução das aplicações frente à volatilidade de seus requisitos ocorra de forma conjunta, rápida e eficiente.

Propõe-se uma arquitetura baseada na **Orientação a Objetos** (OOP) para promover o compartilhamento de recursos entre diferentes sistemas sem comprometer suas capacidades de especialização. Adicionalmente, uma estratégia de redução do impacto de mudanças é apresentada no extensivo emprego de **arquivos de configuração**. A combinação destas duas metodologias, motivadas tanto pela conciliação de argumentos encontrados na literatura especializada quanto por experiência pessoal adquirida como desenvolvedor, representa a verdadeira contribuição deste projeto.

A plataforma recebeu o nome de **Fence** e detalhes de sua implementação serão apresentados no capítulo seguinte.

Capítulo 5

Fence

Fence é um acrônimo para *Front-End Engine for Glance*, o *framework Web* proposto neste projeto de graduação para unificar a geração dos sistemas desenvolvidos para os experimentos do LHC. A partir da análise do problema e o consequente levantamento de requisitos mapeados anteriormente, este capítulo apresenta em detalhes a implementação da plataforma e de que forma a solução apresentada é capaz de promover a rápida evolução das aplicações.

A análise do histórico das intervenções em códigos-fonte que se fizeram necessárias ao longo de 4 anos mostrou que a principal força motriz desta dinâmica tem suas origens na característica progressista e no âmbito colaborativo e universal em que os grandes experimentos do CERN se encontram. Do ponto de vista da engenharia de software, o ambiente em que os sistemas se desenvolvem pode ser classificados como *turbulentos*, o que se reflete na característica mutável e emergente de uma grande parte de seus requisitos.

A solução proposta busca estabelecer uma arquitetura que se antecipe à natureza volátil destes tipos de requisitos, fazendo com que eventuais pedidos de alteração das regras das interfaces sejam concluídos de maneira mais eficaz pelos desenvolvedores. Entende-se que certas modificações sequer devam ser intermediadas por um desenvolvedor. Um estágio futuro prevê, com este objetivo, um canal de interação da plataforma diretamente com usuários credenciados que poderão alterar regras codificadas nas aplicações sem qualquer conhecimento de linguagens de programação.

Com a redução do tempo de resposta da equipe de desenvolvedores frente ao dinamismo de uma considerável parcela dos requisitos, a força de trabalho antes empregada na manutenção dos sistemas pode dedicar-se com maior diligência ao estudo de novas tecnologias e à elaboração de soluções inovadoras que ofereçam novas abordagens ao modelo de gestão dos experimentos, ratificando o ímpeto inovador da colaboração entre a UFRJ e o CERN. De fato, a reengenharia de algumas das aplicações de suporte já sob a nova plataforma evidencia esta tendência, como será discutido em mais detalhes em breve.

5.1 Arquitetura

Como o nome sugere, a solução apresentada se propõe a complementar as funcionalidades introduzidas pelo Glance. Mais especificamente, busca prover uma camada entre a recuperação de dados e a visualização destas informações de acordo com uma série de regras estabelecidas por cada experimento. Segundo a arquitetura proposta, mediante sistemáticas chamadas à API do Glance, dados das gerências dos detectores armazenados em fontes heterogêneas são capazes de serem incorporados pelo Fence, onde são processados e formatados de acordo com diretrizes codificadas em arquivos de configuração para então gerarem as diversas interfaces que compõem os sistemas dos experimentos.

Esta dinâmica é ilustrada na Figura 5.1, onde um dos conceitos-chave por trás desta arquitetura é exposto de maneira clara. Como a resposta do *framework*, escrito em PHP, é condicionada a regras especificadas em arquivos de configuração, a alteração do comportamento das interfaces pode, até certo ponto, ser imposta pela manipulação de parâmetros externos e, portanto, alheios ao código-fonte. Esta abordagem permite que os sistemas possam ser adaptados às variações de seus requisitos com um impacto de mudança virtualmente inexistente.

Obviamente, esta afirmação só se sustenta enquanto os arquivos de configuração forem capazes de abstrair as regras dos sistemas e dado que o façam de modo eficaz e intuitivo. Para sustentar esta premissa, a plataforma se utiliza largamente de uma

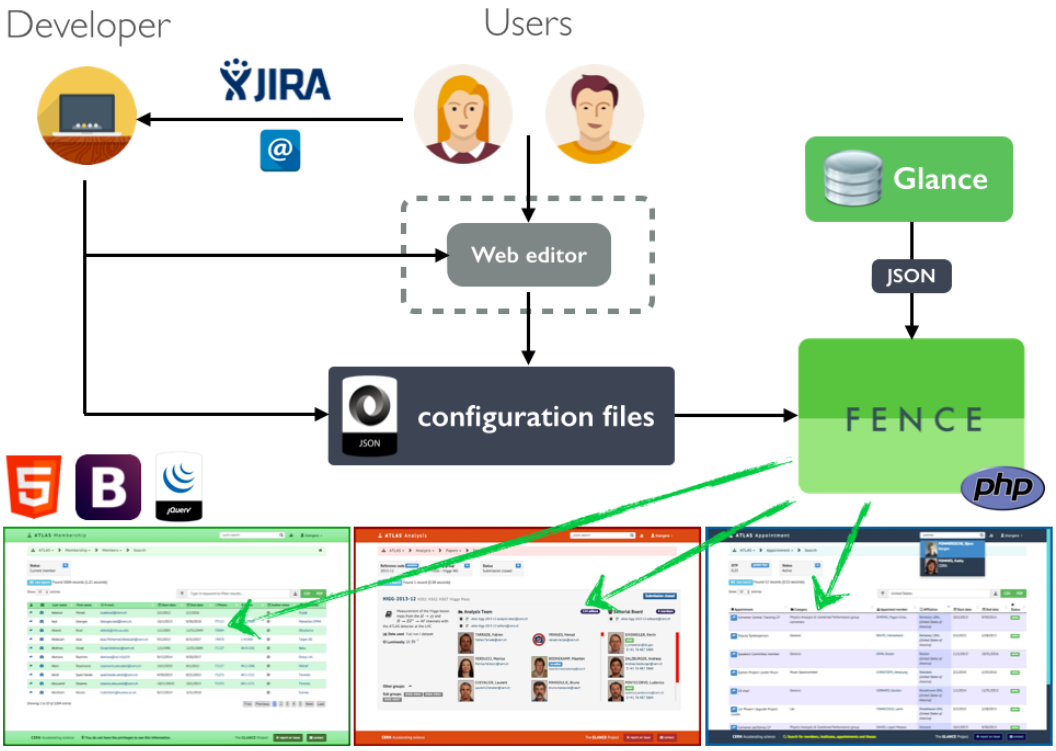


Figura 5.1: O *framework* Fence. Arquivos de configuração controlam a aparência e funcionalidades das interfaces.

tecnologia de abstração e transferência de dados conhecida como **JSON** (*JavaScript Object Notation*) [45]. Antes de explorarmos com mais detalhes de que forma o *framework* incorpora esta estrutura de dados, é preciso introduzirmos paradigmas que, internamente, deram bases à implementação da solução proposta.

5.2 Orientação a objetos

Internamente, o Fence é estruturado pelo modelo de programação orientada a objetos (OOP). Esta abordagem é concretizada pela implementação de uma série de classes, isto é, abstrações das diversas entidades que compõem uma solução em software. O comportamento de cada classe é codificado em seus **métodos** que, quando invocados, alteram o estado do objeto através da manipulação de seus **atributos**. Um exemplo de uma classe seria `Aluno`, com `matrícula` como um de seus atributos e `estudar` como um método. O encapsulamento de propriedades e procedimentos em classes dedicadas facilita a compreensão do código e torna sua

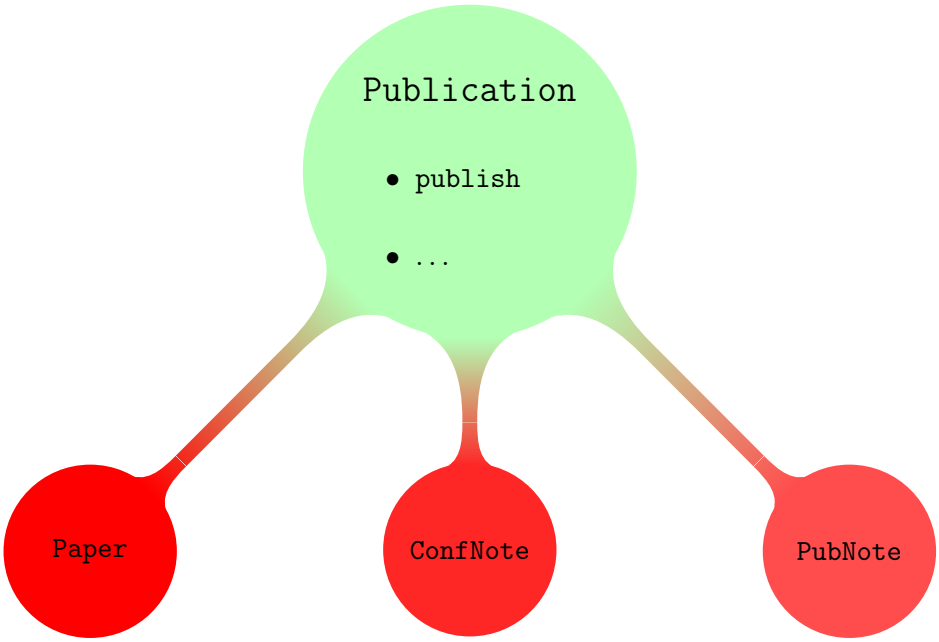


Figura 5.2: Diagrama de classes para os sistemas *Analysis*. Semelhanças entre os sistemas são codificadas na classe base e transmitidas automaticamente para as herdeiras.

manutenção e expansão mais simples [23].

No entanto, o grande apelo desta modelagem, no contexto dos diferentes experimentos e sistemas que a plataforma servirá, reside não necessariamente no conceito de encapsulamento, mas na estrutura hierárquica em que as classes podem ser arranjadas. Através de um mecanismo conhecido como **herança**, métodos e atributos (coletivamente conhecidos como membros) comuns a diversas instâncias podem ser definidos em uma classe base, ou pai, e serem automaticamente herdadas pelas classes filhas. Esta organização é convenientemente ilustrada pelos chamados **diagramas de classes**, como o exemplificado na Figura 5.2.

Imediatamente, identifica-se que a aplicação do dispositivo de herança pode ser empregada como um mecanismo de redução de duplicação de código, já que funcionalidades comuns podem ser implementadas uma única vez em uma classe base para então serem compartilhadas entre classes herdeiras. De fato, a estrutura ilustrada na Figura 5.2 é utilizada pelo Fence para que as similaridades dos sistemas *Analysis*, discutido na Seção 3.2.3, sejam agrupadas em uma única unidade de abstração.

Instâncias das classes `Paper`, `ConfNote` e `Pubnote` compartilham a mesma definição do método `publish` e, assim sendo, alterações no protocolo de publicação que venham a ser requisitados são transferidos automaticamente a todos os subsistemas mediante a manipulação de somente um método na classe base.

Código 5.1: Classe base.

```php
<?php
abstract class Publication {
    protected static $properties = array(
        "id" => array(
            "type" => "number",
            "min"  => 1,
            "max"  => 9999
        ),
        "title" => array(
            "type" => "text",
            "minlength" => 5,
            "maxlength" => 255
        )
    );
    protected abstract function getData();
    public function __construct($id){
        $this->set("id", $id);
        $this->getData();
    }
    public function publish() {
        $this->storeInDatabase();
        $this->shipToJournal();
    }
    // ...
}
```

Código 5.2: Classe herdeira.

```php
<?php
class Paper extends Publication {
    protected function getData() {
        // connect to database or
        // proxy through Glance
        // to get Paper information
    }
    /**
     * Overwriting
     * publish method
     */
    public function publish() {
        parent::publish();
        // Additional step,
        // notify collaboration!
        $this->nofify();
    }
    private function notify() {
        // trigger email
    }
}
```

Por outro lado, e se as especificações preverem regras ligeiramente distintas para a publicação de uma destas classes? Suponha, por exemplo, que o *Physics Office* do ATLAS estabeleça que a publicação de artigos no sistema *Papers* deva envolver um passo adicional onde toda a colaboração é notificada através do envio automático de um *email* após a submissão. O método herdado `publish` não vislumbra esta possibilidade e nem deve ser alterado, já que afetaria também os demais sistemas para os quais o novo requisito não existe. A quebra de herança invariavalmente acarretaria a

duplicação dos demais métodos e atributos que ainda pertencem ao escopo comum. Felizmente, através de um mecanismo conhecido como **polimorfismo**, as classes filhas podem redefinir a implementação de métodos herdados, dispondo da autonomia para impor suas próprias especificações. No Código 5.2, o método `publish` foi redefinido para observar este cenário hipotético.

5.2.1 Mecanismos de herança

O PHP estabelece três diferentes níveis de visibilidade para membros de classes. São eles:

- **Membros públicos**: identificados pela palavra-chave `public`, estes métodos e atributos são acessíveis diretamente pelas instâncias das classes. Métodos e atributos públicos fornecem uma interface através da qual os objetos podem ser manipulados exteriormente. No exemplo do Código 5.2, o método `publish` pode ser invocado por uma instância de `Paper` da seguinte forma:

```
$paper = new Paper($paperId);
$paper->publish();
```

- **Membros protegidos**: são métodos e atributos identificados pela palavra-chave `protected`, tornando-os acessíveis somente dentro da própria classe e das classes herdeiras.

- **Membros privados**: são de uso exclusivo da classe que os declara, explicitamente impedindo sua propagação a classes herdeiras via herança. Métodos e atributos privados são assim definidos pelo emprego do identificador `private` em suas declarações.

Outro aspecto que cabe mencionar diz respeito à declaração da classe base no Código 5.1 como *abstrata*. Por meio deste artifício, instâncias diretas de `Publication` não são permitidas. Ao invés, classes abstratas funcionam como espécie de contrato de herança, permitindo que objetos herdem seus membros públicos e protegidos, contanto que forneçam uma implementação dos métodos declarados abstratos.

Observa-se que, no Código 5.2, a classe filha `Paper`, de fato, implementa o método `getData`, declarado abstrato na classe base (Código 5.1, linha 15).

Finalmente, o Código 5.1 ilustra o uso de membros estáticos. Como o nome sugere, métodos e atributos estáticos são alocados em um único endereço de memória, sendo referenciado por todas as instâncias de uma classe. Esta estratégia provou ser de grande valia pelo amplo volume de dados armazenados pelos experimentos. Por exemplo, uma interface típica do sistema *RackWizard* envolve a recuperação de equipamentos na ordem de centenas ou até milhares a cada requisição. Como a descrição das propriedades é a mesma para todos os equipamentos, a declaração estática permite que todas as instâncias compartilhem uma única estrutura de dados, acelerando a geração das interfaces e reduzindo o consumo de memória no servidor em até 87%.

5.3 Arquivos de configuração

Dada a abstração e potencial volatilidade dos requisitos por trás dos sistemas de suporte às gerências do experimentos, ficou claro que uma estratégia de externalização das regras do código-fonte teria de ser mediada por uma estrutura de dados flexível, de fácil e eficiente incorporação em diversas tecnologias.

Historicamente, o formato mais comum para o armazenamento e intercâmbio de dados dinâmicos entre aplicações é o XML (*Extensible Markup Language*). No entanto, estudos comparativos mostram que o JSON tem alcançado melhores indicativos de performance [46]. Além disso, sua sintaxe simples encontra respaldo em estruturas de dados suportadas nativamente por diversas linguagens de programação. Por estes motivos, o formato vem ganhando popularidade nos últimos anos e foi adotado como o padrão utilizado nos arquivos de configuração incorporados pelo *framework* proposto.

A sintaxe do JSON [47] é mostrada no esquema da Figura 5.3. O formato é, simples o bastante, baseado em duas estruturas. São elas:

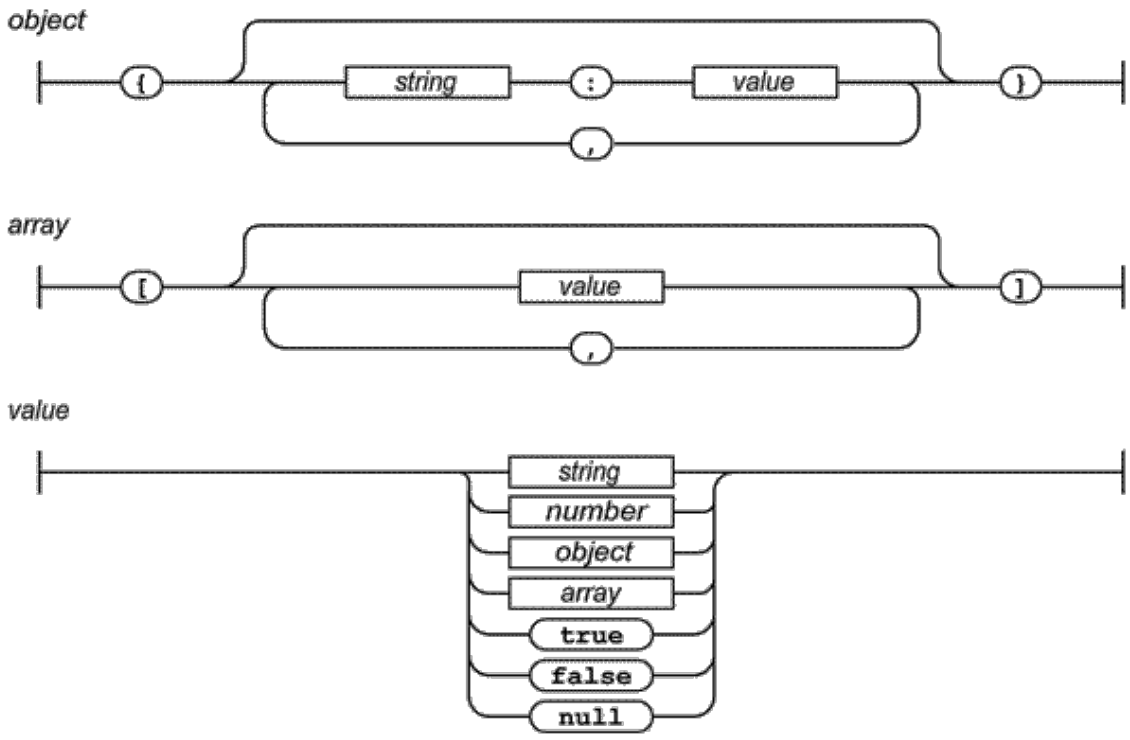


Figura 5.3: O formato JSON. Extraído de [45].

1. **Objeto**: trata-se de uma coleção de pares chave-valor, onde uma chave deve, obrigatoriamente, ser do tipo *string*, e o valor pode assumir um dos diferentes formatos mostrados na Figura 5.3. Esta estrutura de dados é encontrada em diversas linguagens de programação sob diferentes nomenclaturas como dicionário, vetor associativo e tabela de *hash*.

2. **Vetor** (*array*): são listas ordenadas de valores. Esta topologia é também conhecida como lista ou sequência em diferentes linguagens de programação.

A extensa capacidade de abstração deste modelo se dá pela diversidade de formatos que os valores dentro de objetos e vetores podem assumir. Não somente estruturas simples de dados como *strings*, números e variáveis booleanas e nulas podem ser atribuídas, mas valores podem também assumir novos objetos e listas.

Um exemplo ilustrativo deste caráter recursivo é mostrado no Código 5.3, onde o formato foi aplicado para armazenarmos algumas propriedades do LHC e seus experimentos. O documento em si corresponde a um objeto para o qual diver-

sos atributos estão definidos. A chave `name` guarda o nome do acelerador, uma *string*, enquanto que `countries` assume uma lista, já que os aproximadamente 27 quilômetros de circunferência do LHC se expandem pela Suiça e França. O estado operacional do LHC é determinado pela variável booleana associada à chave `beam`. E, finalmente, os experimentos acoplados ao acelerador correspondem a uma lista de objetos associada à chave `experiments`.

Código 5.3: Exemplo de dados descritos pela sintaxe JSON.

```
{
    "name": "Large Hadron Collider",
    "countries": ["France", "Switzerland"],
    "circumference": 26.659,
    "energy": {
        "unit": "TeV",
        "value": 13
    },
    "beam": true,
    "experiments": [
        {
            "abbreviation": "ATLAS",
            "name": "A Toroidal LHC Experiment",
            "generalPurpose": true,
            "diameter": 22
        },
        {
            "abbreviation": "LHCb",
            "name": "Large Hadron Collider - beauty",
            "generalPurpose": false,
            "diameter": null
        },
        {
            "abbreviation": "ALICE",
            "name": "A Large Ion Collider"Experiment",
        // ...
    ]
}
```

5.3.1 Escopos global e local

A eficiência na manutenção de grandes aplicações, com dezenas de milhares de linhas de código e centenas de arquivos, depende invariavelmente de uma intuitiva organização do código-fonte no servidor. O Fence adota a própria estrutura de diretórios para decompor grandes aplicações em pequenas unidades funcionais. O sistema *Membership* do ATLAS, por exemplo, pode ser dividido em unidades de gerência de membros e de institutos. Assim sendo, os subsistemas são hospedados em pastas dedicadas.

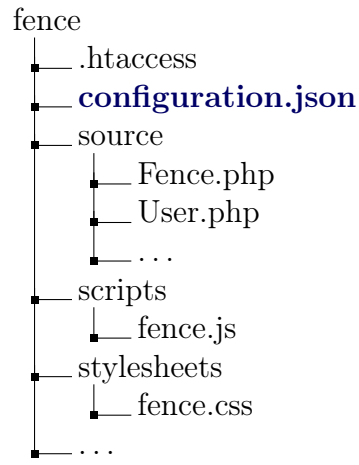
A Figura 5.4a mostra a estrutura de diretórios do Fence. Na raíz do repositório, encontra-se destacado em negrito o arquivo `configuration.json`. Trata-se do arquivo de configuração **global** da plataforma, isto é, as diretrizes ali definidas são automaticamente propagadas por todas as interfaces geradas pelo Fence. Em um paralelo com o paradigma de herança da OOP, cada sistema gerado pela plataforma herda uma série de parâmetros já definidos globalmente.

Esta abordagem é escalável para cada experimento, seus sistemas e subsistemas. Neste contexo, cada diretório contendo um arquivo de configuração próprio (obrigatoriamente entitulado `configuration.json`) encapsula um escopo **local**, onde diretrizes globais podem ser expandidas ou, se necessário, completamente redefinidas. O *framework* se vale do conceito de polimorfismo ao priorizar escopos locais em detrimento de globais, permitindo que cada experimento imprima a seus sistemas um conjunto específico de regras.

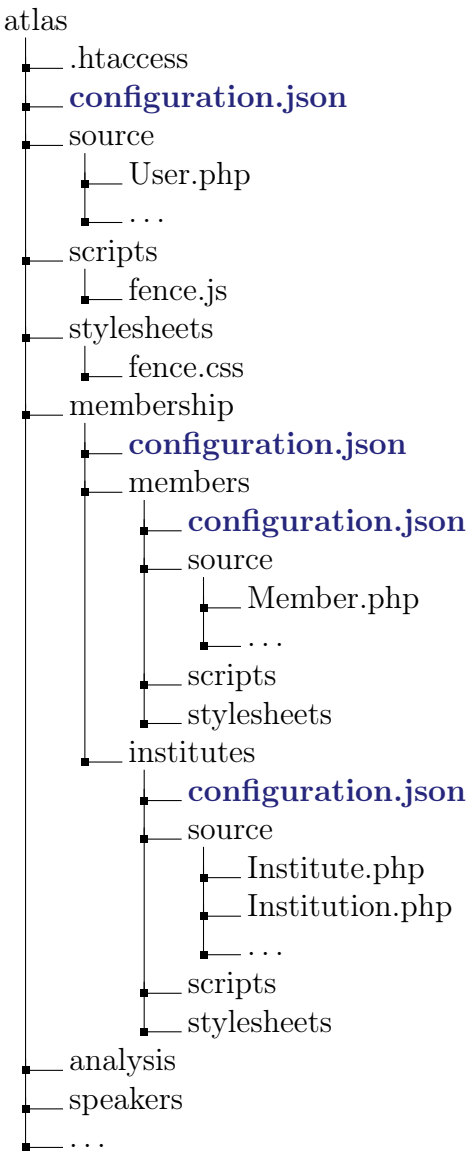
Pela convenção estabelecida pelo *framework*, a definição e implementação de classes fica reservada às pastas `source` dentro de cada escopo. E, além dos arquivos de configuração, a plataforma se aproveita da estrutura de diretórios para incorporar também *scripts* e folhas de estilo. Suponha que um usuário solicite a página de busca de membros do ATLAS através da seguinte URL:

```
https://glance.cern.ch/atlas/membership/members/search
```

O Fence inspeciona cada um dos diretórios que compõe o caminho até a interface requisitada (`atlas`, `membership`, `members`), integrando a sua resposta quaisquer



(a) Estrutura de diretórios do Fence.



(b) Estrutura de diretórios do ATLAS.

Figura 5.4: Organização de diretórios no servidor *Web*.

arquivos `fence.js` e `fence.css` encontrados. A simples presença destes arquivos assinala que seus conteúdos devem ser incorporados as interfaces. Novamente, esta estratégia permite que *scripts*, cujas funções serão elucidadas em breve, e folhas de estilo - linguagens de descrição da aparência e formatação das páginas, também possam ser projetados para diferentes escopos.

5.3.2 Diretrizes dinâmicas

Para expandir a capacidade de abstração dos arquivos de configuração, o Fence estabelece uma série de caracteres especiais que podem ser utilizados para encapsular variáveis de fontes terceiras. Chaves duplas (`{{key}}`) fazem referência a constantes definidas anteriormente, seja em arquivos de configuração precedentes ou até no mesmo próprio arquivo, contanto que respeitada a causalidade na ordem em que as variáveis são declaradas. No exemplo mostrado no Código 5.4, este recurso é utilizado para identificar-se o diretório que hospeda as classes do ATLAS, chamado de `atlas_src`, em função de `atlas_path`, definido na linha anterior e que, por sua vez, depende de `base_path`, definido no arquivo de configuração global.

```
1
2     "atlas_path": "{{base_path}}/atlas",
3     "atlas_src":  "{{atlas_path}}/source",
4
5     "target_id":  "{%id%}"
6
```

Código 5.4: Trecho do arquivo de configuração do ATLAS.

Dados submetidos pelo usuário podem também ser incorporados aos arquivos de configuração. Os caracteres `{%` e `%}` encapsulam referências a campos tipicamente enviados ao servidor quando o usuário acessa um link especialmente formatado ou por intermédio de formulários HTML. Internamente, o Fence busca por possíveis referências na combinação dos métodos HTTP `GET` e `POST`. No exemplo do Código 5.4, `target_id` assumiria o valor 42 se o sistema fosse configurado através de uma requisição à URL `https://glance.cern.ch/atlas/appointment/details?id=42`.

5.3.3 A classe `Configuration`

A assimilação dos arquivos de configuração de diretórios é intermediada por uma instância única (*singleton*) da classe `Configuration`, armazenada em uma variável global do PHP chamada de `$_SESSION` e que persiste entre solicitações dos usuários autenticados por até 8 horas. Este objeto assume a tarefa de navegar a árvore de diretórios a partir da interface requisitada, garantindo a prioridade de escopos locais sobre o global e ocupando-se da identificação e interpretação dos caracteres de escape discutidos anteriormente.

Diretrizes armazenadas nos arquivos de configuração podem ser acessadas mediante chamadas ao método `get`, que recebe uma chave como único argumento e retorna o valor associado. Alternativamente, o método `set` pode ser utilizado para programaticamente definir-se novas chaves ou sobrescrever-se existentes.

5.3.4 Atributos de acesso

A segurança no acesso e manipulação das informações é uma constante preocupação da gestão de sistemas. O Controle de Acesso Baseado em Papéis (RBAC) estabeleceu uma estratégia de ampla abrangência ao identificar que permissões são atribuídas a papéis (ou grupos) dentro de uma organização, com usuários podendo assumir um ou mais papéis [38].

A flexibilidade do RBAC levou ao seu emprego no controle ao acesso das informações expostas nos sistemas atuais, especialmente nos sistemas *Membership*. A minuciosa definição de papéis e permissões já estabelecidos nos bancos de dados dos experimentos motivou a incorporação desta abordagem também no Fence. Em acordo com sua proposta de externalização das regras, o *framework* reconhece diferentes atributos de acesso na descrição de qualquer objeto JSON oriundo de um arquivo de configuração ou "artificialmente" fabricado pelo PHP. São eles: `usergroups` e `permissions`, associdados aos papéis e permissões, respectivamente, conforme sugeridos pela estratégia RBAC.

Adicionalmente, o Fence introduz um atributo complementar denonimando `egroups`,

dada a já vastamente estabelecida utilização de listas de e-mails no ambiente do CERN. O *framework* explora a familiaridade de usuários com a ferramenta para permitir que o controle das interfaces e objetos em geral possa ser realizado sem a mediação de desenvolvedores, já que o controle dos membros de cada grupo pode ser feito diretamente por administradores dos *e-groups*.

Para que acesso ao objeto protegido seja permitido, é necessário que o usuário se enquadre em todos os itens descritos em ao menos uma das listas. Para o exemplo do Código 5.5, o usuário logado só pode acessar o objeto associado se pertencer simultaneamente aos grupos `active` e `physicists` ou se pertencer ao grupo `fence-developers`.

Quando aplicados aos arquivos de configuração de diretórios `configuration.json`, os atributos de acesso protegem todas as interfaces ali hospedadas. Quando quaisquer dos atributos não forem verificados, o Fence abstém-se da recuperação de dados e formatação de conteúdos e diverge sua saída para uma página padrão com uma simples mensagem informando o usuário que a interface requisitada não pode ser acessada, como mostrada na Figura 5.5. Um botão de contato é também disponibilizado para o contato direto com a equipe de desenvolvedores caso haja quaisquer dúvidas.

```
1 "egroups": [
2     ["active", "physicists"],
3     ["fence-developers"]
4 ]
```

Código 5.5: Atributos de acesso: lista de listas de *e-groups* protegendo um diretório hipotético.

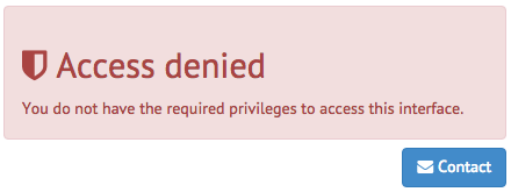


Figura 5.5: Acesso a uma interface negado pela falta de credenciais.

5.4 Desenvolvimento colaborativo

O rápido crescimento da plataforma - impulsionado pelo interesse em sua adoção na geração de um total de 21 sistemas, fez com que ferramentas de desenvolvimento

colaborativo fossem imbutidas no modelo de trabalho de modo a facilitar a incorporação de funcionalidades e correções introduzidas por uma equipe de desenvolvedores. Esta seção apresenta algumas dessas ferramentas.

5.4.1 Controle de versão

Para que os sistemas de suporte acompanhem o dinamismo de seus requisitos, intervenções no código-fonte podem ser minimizadas (ver Seção 5.3.4) mas, na prática, o código-fonte será alvo de alterações ao longo de seu ciclo de vida. A imprescindível gerência desta dinâmica se dá através de instrumentos de controle de versão. Estas ferramentas não só proveem um compreensivo histórico das mudanças em código-fonte como permitem que diversos colaboradores trabalhem paralelamente no mesmo projeto.

O **Git** [48] é um sistema de controle de versão gratuito e de código aberto de ampla popularidade atualmente, contando com suporte oficial do departamento de Tecnologia da Informação do CERN. Dentre suas vantangens, cabe destacar o excelente suporte para a manipulação de *branches*, ramos independentes de desenvolvimento que tornam o processo de integração de correções e novas funcionalidades muito mais organizado e intuitivo.

O desenvolvimento colaborativo do Fence é coordenado pelo estabelecimento de duas *branches* principais, mostradas em negrito na Figura 5.6a. Na chamada *stage*, as mais recentes funcionalidades são incorporadas após desenvolvimentos paralelos que se dão nas *feature branches*. Tipicamente, cada ramo é coordenado por um desenvolvedor que dispõe de um ambiente próprio normalmente hospedado em seu próprio computador (*localhost*) ou em diretórios designados no servidor dedicado `glance-stage.cern.ch`. Quando concluídas, as alterações são absorvidas pela *stage*, onde versões idealmente estáveis são disponibilizadas para permitir um primeiro contato dos usuários finais com as aplicações ainda em desenvolvimento.

Já a *master* é associada ao ambiente de produção, isto é, a versão que opera sobre os bancos de dados oficiais e acessíveis a toda colaboração através do domínio `glance.cern.ch`. Adições a esta *branch* estabelecem obrigatoriamente uma nova

Repositório	Endereço
Fence	https://git.cern.ch/reps/fence.git
ATLAS	https://git.cern.ch/reps/fence-atlas.git
ALICE	https://git.cern.ch/reps/fence-alice.git
LHCb	https://git.cern.ch/reps/fence-alice.git

Tabela 5.1: Repositórios Git hospedados em servidores do CERN.

versão do *framework*, identificada no Git através das chamadas *tags*. Dependendo da natureza das alterações, incrementos são aplicados em uma das três partições semânticas (`MAJOR.MINOR.PATCH`) que compõem a versão final [49].

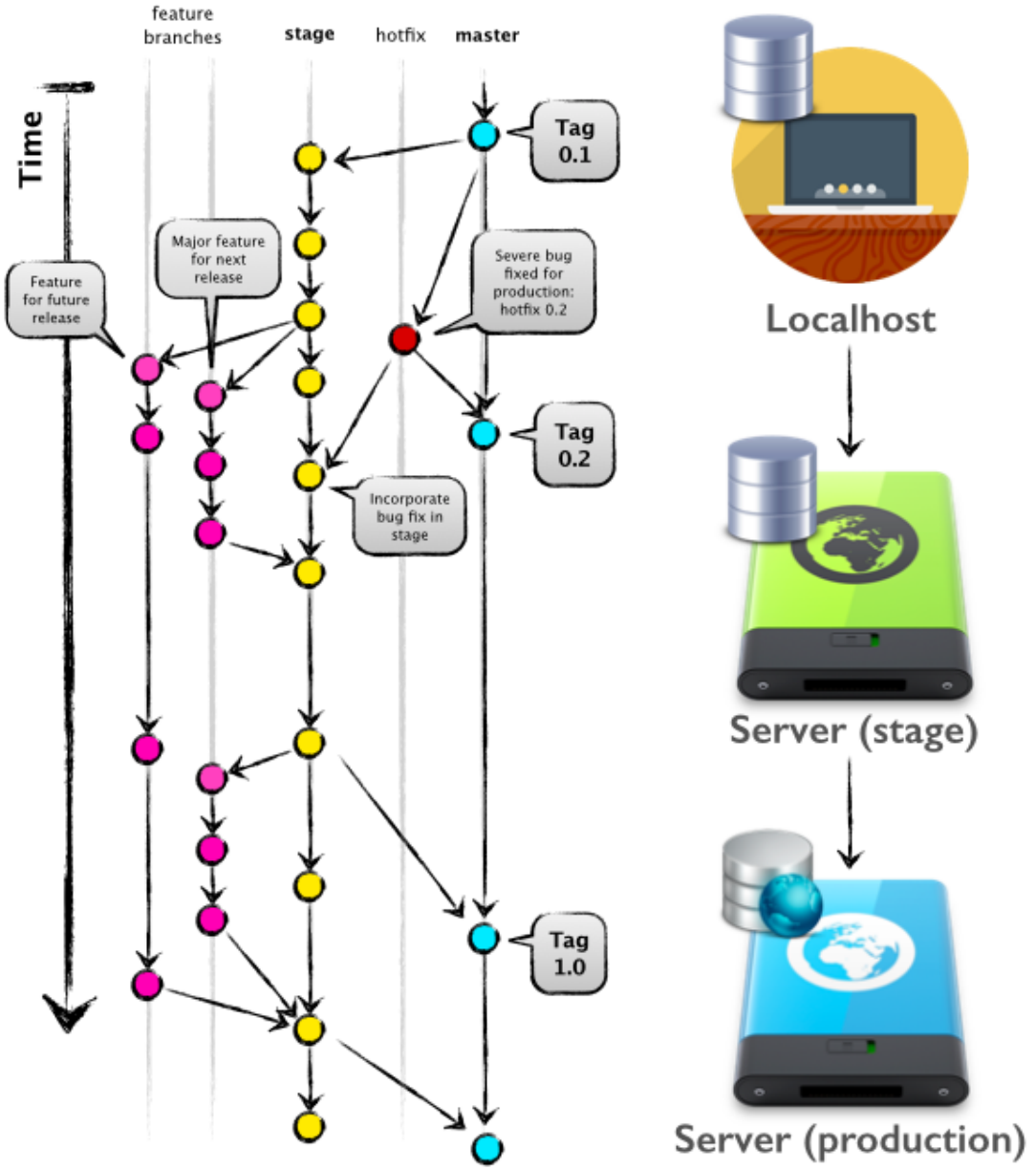
O padrão implementado pelo Fence adota, ainda, uma outra *branch* chamada *hotfix*, uma espécie de atalho à versão de produção para que correções urgentes sejam incorporadas. A dinâmica entre as *branches* do controle de versão e os ambientes de desenvolvimento é ilustrada na Figura 5.6.

Há um total de 4 repositórios que compõem a solução completa proposta neste projeto: um para o Fence e um para cada um dos três experimentos que se utilizam do *framework* na geração de seus sistemas de suporte. Os endereços de cada um dos repositórios hospedados nos servidores do CERN são encontrados na Tabela 5.1.

5.4.2 Configuração dos servidores

Os servidores *Web* que hospedam as aplicações geradas pelo Fence são máquinas virtuais criadas com a ferramenta de virtualização *OpenStack*, para o qual o CERN provê suporte oficial, e rodam *Apache* sobre o sistema operacional *Scientific Linux 6*. Conexão a estes servidores é feita via SSH (*Secure Shell*), uma implementação do protocolo de mesmo nome que permite o estabelecimento de um canal criptografado de interação entre dois nós de uma rede.

Os servidores Apache permitem que diretórios sejam localmente configurados por meio de um arquivo chamado *.htaccess* (*hypertext access*). Este recurso é explorado pelo Fence para impor diferentes escopos de forma análoga ao emprego de seus



(a) Modelo de *branches*. Inspirado em [50].

(b) Estágios de desenvolvimento.

Figura 5.6: Modelo de desenvolvimento adotado no Fence.

próprios arquivos de configuração. Como o acesso a interfaces é restrito a membros com contas ativas no CERN, este requisito pode ser imposto globalmente por um arquivo *.htaccess* na raiz do servidor - como o mostrado no Código 5.6. As linhas 2 a 6 ativam a chamada SSO (*Single Sign-On*), autenticação centralizada mediada pelo departamento de TI do CERN através de tecnologia *Shibboleth*.

Além disto, estes arquivos são utilizados para definir variáveis através do comando `SetEnv` para serem captadas pelo Fence durante sua inicialização. A variável `ENVIRONMENT` identifica o ambiente de execução dos sistemas, podendo ser utilizada para controlar o nível de detalhamento do relatório de erros ou credenciais de conexão ao banco de dados, por exemplo.

Código 5.6: Arquivo .htaccess na raiz do servidor glance-stage.cern.ch

```
1  ############################################################
2  SSLRequireSSL
3  AuthType shibboleth
4  ShibRequireSession On
5  ShibExportAssertion Off
6  Require valid-user
7  ############################################################
8  SetEnv ENVIRONMENT      DEVELOPMENT
9  SetEnv COMMON_FOLDER    "/srv/fence"
10 SetEnv LOG_PATH         "/srv/logs/fence"
```

5.4.3 Documentação

Para que o *framework* seja empregado de forma eficaz na construção de novas interfaces, é vital que os desenvolvedores disponham de uma compreensiva e correta descrição do conjunto de classes existentes. Minimamente, todos os membros públicos de uma classe tem de ser propriamente documentados. Atributos públicos devem ser brevemente descritos e a apresentação de métodos públicos deve incluir a natureza de cada um de seus argumentos de entrada e da saída.

Praticamente todas as linguagens de programação fornecem caracteres especiais que delimitam comentários, isto é, anotações associadas trechos de código para

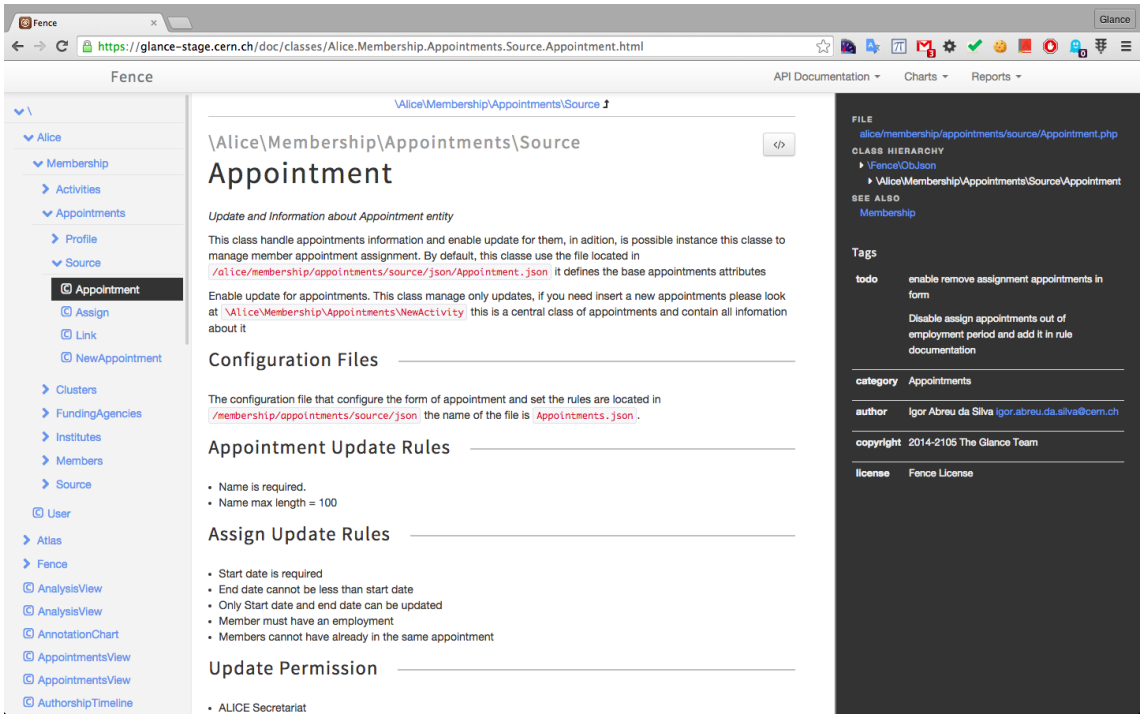


Figura 5.7: Documentação gerada pelo *phpDocumentor* para a classe *Appointment* do ALICE.

torná-los mais compreensíveis. Embora sejam normalmente ignorados por compiladores e interpretadores, comentários especialmente formatados nos chamados *Docblocks* podem ser capturados por ferramentas específicas para gerarem interfaces *Web* que exponham a documentação de todas as classes que compõem um projeto. No Fence, esta tarefa é atribuída ao *phpDocumentor* [51]. A Figura 5.7 mostra a página gerada por esta ferramenta através da captura de *Docblocks* no arquivo que hospeda a classe *Appointment* do ALICE.

Dada a particular preocupação do *framework* com a eficiência na manutenção dos sistemas, o estabelecimento de um estilo de codificação torna-se imprescindível para tornar os códigos mais fáceis de serem lidos, compreendidos e mantidos coletivamente por toda a equipe de desenvolvedores. O Fence adota o popular padrão PSR-2 [52] com este objetivo.

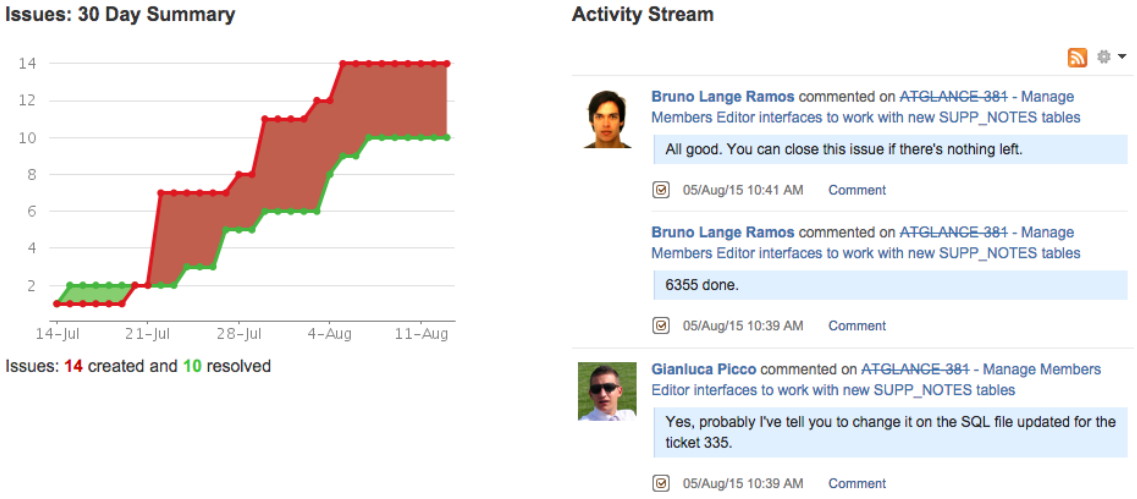
5.4.4 Registro e acompanhamento de incidentes

A partir da identificação da natureza volátil dos requisitos (discutida na Seção 4.1), a solução proposta buscou dar suporte à geração de interfaces flexíveis que pudessem ser facilmente adaptadas a mudanças em suas especificações. Para que esta dinâmica seja corretamente gerenciada tanto pelos *stakeholders* quanto pelos desenvolvedores, a utilização de ferramentas para o registro e acompanhamento dos incidentes que implicam em alterações nos sistemas de suporte se faz igualmente indispensável.

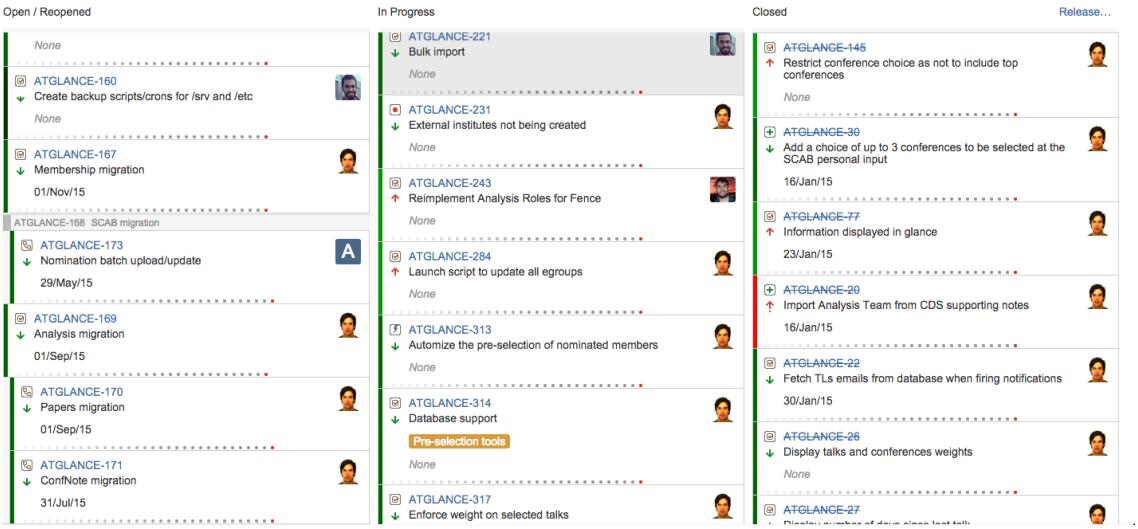
O registro de *bugs*, sugestões de melhorias, novas funcionalidades e tarefas em geral é atribuído ao JIRA [53]. Trata-se de um sistema de gestão e planejamento de tarefas e incidentes que acompanha todas as etapas do ciclo de vida de desenvolvimento e manutenção de sistemas. Usuários podem acompanhar o andamento de suas requisições através de uma interface simples e intuitiva, sendo notificados por email quando qualquer atualização é realizada pelo desenvolvedor encarregado da tarefa.

O time de desenvolvedores tem acesso a sínteses do balanço de incidentes abertos e fechados em qualquer período, além de poder facilmente inspecionar as mais recentes modificações, como mostrados na Figura 5.8a. Através de variados relatórios de eficiência, é possível identificar-se gargalos e padrões de reincidência a serem abordados.

O JIRA provê suporte a metodologias de desenvolvimento ágeis, onde soluções evoluem pela coordenada colaboração de um time de programadores. Na estratégia conhecida como Kanban [54], ilustrada na Figura 5.8b, a força de trabalho se concentra nas atividades em progresso (listadas na coluna central). A conclusão de uma atividade implica na sua remoção da coluna central, dando lugar àquela no topo da coluna da esquerda, onde tarefas em "estoque" são ordenadas de acordo com suas prioridades.



(a) Balanço de incidentes nos últimos 30 dias e modificações recentes.



(b) Estratégia de desenvolvimento ágil Kanban.

Figura 5.8: Sistema JIRA para a gestão e planejamento de tarefas e incidentes.

5.5 Geração de sistemas

Um sistema é composto normalmente por uma série de *interfaces* que, em uma definição um tanto ampla, proporcionam a interação dos usuários com um ou mais bancos de dados. As informações armazenadas são capturadas, processadas e formatadas em um documento formatado na linguagem HTML (*Hypertext Markup Language*) capaz de ser renderizado por um navegador. Por outro lado, o navegador é utilizado também como o mediador da inserção de novos dados através campos de entrada agrupados em formulários.

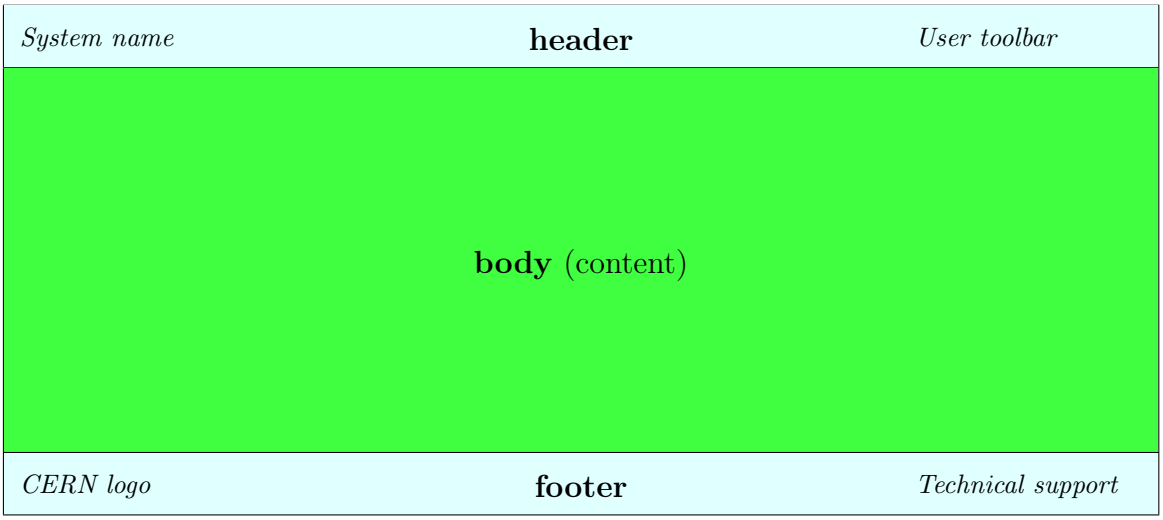


Figura 5.9: Estrutura das interfaces geradas com o Fence.

O Fence prevê um suporte nativo tanto para as operações de **leitura** quanto de **escrita** que um sistema deve, minimamente, fornecer. Para tal, o *framework* disponibiliza uma série de classes que implementam interfaces de busca e formulários de inserção. Estas classes representam implementações de uma classe abstrata base chamada `Content`, cujo contrato de herança impõe o emprego de um arquivo de configuração onde as regras intrínsecas de cada classe possam ser abstraídas e facilmente manipuladas quando necessário.

Para compreendermos como o *framework* incorpora estes conteúdos em suas interfaces, deve-se primeiro compreender a estrutura comum por trás de cada uma delas.

5.5.1 Estrutura das interfaces

Cada interface gerada com a plataforma proposta é semântica e espacialmente segmentada em três componentes: um cabeçalho (*header*), um corpo (*body*) e um roda-pé (*footer*), como mostra a estrutura ilustrada na Figura 5.9.

O cabeçalho contém as informações sobre a interface em si e do usuário logado. O nome do sistema é exposto no canto superior esquerdo e o nome e as opções do usuário são listadas no lado oposto.

O roda-pé, por sua vez, contém um *link* para a página central do CERN e, à

direita, endereços de contato e um botão para que problemas possam ser reportados diretamente no JIRA são disponibilizados. O usuário está, em qualquer momento, a um botão de sugerir melhorias ou reportar erros. A grande vantagem desta integração é que informações adicionais a respeito do navegador do usuário são automaticamente incorporadas à descrição do incidente. A identificação do ambiente do usuário permite que eventuais erros sejam reproduzidos mais facilmente.

A funcionalidade primária das interfaces é reservada à área central cuja convenção estabelecida pelo *framework* conhece como **corpo**. A geração de conteúdos no corpo das interfaces é subsidiada por uma classe base abstrata chamada `Content`. A classe não pode ser diretamente instanciada pois define um método público abstrato chamado `render`, que deve ser implementado pelas classes herdeiras para definirem o conteúdo HTML que encapsula as funcionalidades básicas de cada aplicação.

A adoção desta convenção implica que os conteúdos assimilados pelo Fence no corpo das interfaces originam-se sempre em instâncias de classes filhas de `Content`. Internamente, a plataforma armazena uma lista de instâncias deste tipo e simplesmente invoca o método `render` de cada uma delas para gerar o corpo da interface requisitada. O principal diferencial do Fence em relação a outros *frameworks* que se utilizam de estratégias similares reside na incorporação de um arquivo de configuração no construtor das classes de conteúdo, encorajando a externalização de regras e reduzindo o impacto causado pela alteração de requisitos.

No exemplo mostrado no Código 5.7, a classe `HelloWorld` é declarada herdeira de `Content`, qualificando-a para a geração de conteúdo no corpo de uma interface gerada com o Fence. O contrato de herança dos membros da classe base é concluído pela implementação do método público `render` que, neste exemplo, simplesmente imprime a mensagem "*Hello, World!*". A mensagem em si, no entanto, vem de uma fonte externa, sendo declarada no arquivo de configuração `helloWorld.json`, incorporado no instante em que a classe é instanciada.

```php
<?php
require_once "autoload.php";
use \Fence\Fence;
class HelloWorld extends \Fence\Content {
    public function render() {
        $msg = $this->data["message"];
        echo "<h4>". $msg . "</h4>";
    }
}
try {
    $fence = new Fence();
    $fence->addContent(
        new HelloWorld("helloWorld.json")
    );
    $fence->render();
} catch (\Exception $exception) {
    Fence::handleException($exception);
}
```

Código 5.7: Simples exemplo do emprego de classe herdeira de `Content` na geração de conteúdo com o Fence.

```
{
    "egroups": [
        ["atlas-physicists"]
    ],
    "data": {
        "message": "Hello, World!"
    }
}
```

Código 5.8: Exemplo de arquivo de configuração associado a uma filha de `Content`.

Neste simples porém ilustrativo exemplo, a eventual alteração da mensagem exibida não requer qualquer intervenção na implementação da classe de conteúdo, bastando editá-la diretamente no arquivo de configuração. Na prática, o arquivo de configuração codifica em alto nível a descrição de cada um dos elementos estruturados que compõem as interfaces como campos de entrada (*inputs*), formulários e tabelas.

5.6 Tratamento de exceções

Não existem, na prática, sistemas isentos de falhas. No entanto, a identificação de comportamentos anômalos pode ser acompanhada de medidas de contenção que atenuem uma possível percepção negativa dos usuários em relação as aplicações. O Fence disponibiliza uma série de ferramentas para alertar a equipe de desenvolvimento quando situações atípicas se desenvolvem, fornecendo pistas para a elucidação de suas origens e facilitando sua reprodução no ambiente dos programadores.

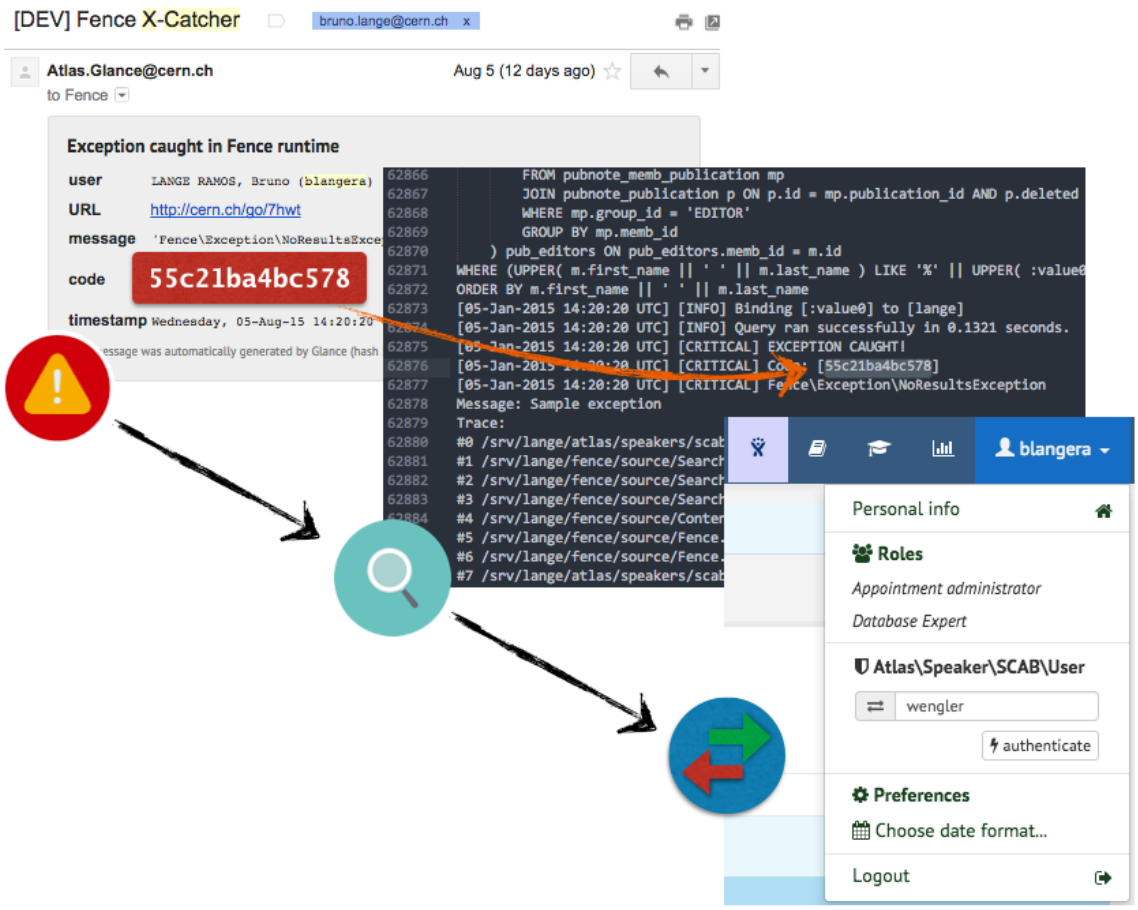


Figura 5.10: Tratamento de exceções no Fence: e-mails de alerta, inspeção de arquivos *logs* e personificação de usuários.

O bloco `try-catch` mostrado no Código 5.7 é adotado como padrão nas chamadas interfaces de entrada, isto é, os *scripts* PHP que são diretamente invocados pela requisição de uma URL. Exceções não captadas em classes internas são tratadas pelo método estático `handleException` da classe `Fence`.

A primeira medida tomada nestes casos é informar o usuário através de uma mensagem amigável e informativa a respeito do imprevisto ocorrido, ressaltando que o suporte técnico já foi alertado. De fato, como mostra a Figura 5.10, um e-mail é automaticamente disparado por uma instância da classe `Mailer`, alertando os desenvolvedores sobre a ocorrência de uma irregularidade. No corpo do email, um identificador do tipo `55c21ba4bc578` é apresentado. Esta sequência única de caracteres permite identificar nos arquivos de *log* o momento exato em que a exceção foi levantada.

Através do minucioso registro de transações nos *logs* pela classe `Logger`, o desenvolvedor consegue investigar as causas dos problemas e reproduzi-los com muito mais facilidade. Como consequência, o tempo investido na correção de *bugs* é reduzido.

Todo este aparato é ainda complementado pela capacidade de desenvolvedores em assumirem as credenciais de outros usuários. Para isto, o Fence disponibiliza, somente a desenvolvedores, um campo de entrada na lista de opções do campo superior direito das interfaces que pode ser preenchido com o login do membro da colaboração a ser personificado. Com isto, eventuais erros podem ser reproduzidos diretamente na interface, o que é particularmente importante para a captura de exceções no lado do cliente. O procedimento completo adotado pelo Fence na resposta a incidentes é ilustrado na Figura 5.10.

5.7 Campos de entrada

Campos de entrada configuram os elementos básicos de interação dos usuários com as aplicações *Web*. Estes elementos podem ser áreas de texto, caixas de seleção de itens pré-listados (*select boxes*), botões, campos selecionáveis múltiplos (*checkboxes*) ou mutualmente exclusivos (*radiobuttons*), ou mesmo arquivos a serem transferidos para o servidor.

Esta multiplicidade de formatos é explorada para o aprimoramento da experiência do usuário ao melhor condicionar as informações submetidas. Com avanços recentes e a ubiquidade de novas mídias em aplicações modernas, o mais recente padrão definido pela *World Wide Web Consortium* (W3C), chamado de HTML5, prevê ainda diversos outros tipos de *inputs*. O formato especifica o suporte nativo de navegadores para campos de tipo data, email, cor e número, para citar alguns. No entanto, nem todos os navegadores suportam estes novos formatos, o que infere a necessidade da aplicação de mecanismos de compatibilidade que garantam o funcionamento das interfaces em diferentes clientes.

Atributo	Tipo	Obrigatório	Descrição
`name`	*string*	✓	Identificador do campo.
`type`	*string*	✓	Tipo do campo. Cada tipo é mapeado em uma classe filha de `BaseInput`.
`label`	*string*	✗	Legenda que acompanha o campo e elucida sua função.
`about`	*string*	✗	Descrição detalhada do campo. Acessível através de um ícone de informação renderizado junto à legenda.
`rules`	*object*	✗	Descrição das regras associadas ao campo.

Tabela 5.2: Atributos do objeto JSON que descreve campos de entrada.

5.7.1 `BaseInput`

O suporte do Fence para a geração de campos de entrada é formulado na simples abstração de *inputs* em objetos chave-valor. De fato, apesar dos muitos formatos disponíveis, a função básica compartilhada por todos estes elementos reside na transferência de dados do cliente para o servidor. A chave identifica o campo e o valor associado é capturado para ser então processado ou armazenado. O *framework* encapsula este modelo através de uma classe base abstrata chamada de `BaseInput`.

Ciente de que muitos dos requisitos atingem essencialmente as especificações dos campos de entrada, a classe externaliza a descrição de suas propriedades ao exigir em seu construtor a incorporação de um objeto que está, tipicamente, hospedado em um arquivo de configuração. A Tabela 5.2 lista alguns dos atributos que estes objetos podem incorporar. Entre eles, dois são mandatórios: um identificador único definido em `name` e o tipo de *input* a ser renderizado, determinado por `type`.

No *framework* proposto, campos de entrada são instanciados por intermédio de um objeto especialmente projetado para a criação de outros objetos - uma **fábrica**. A classe `InputFactory` define o método público e estático `build` exatamente para este propósito. Em um processo ilustrado na Figura 5.11, o método recebe a descrição de um *input* em um formato JSON e retorna um novo objeto filho de *BaseInput*

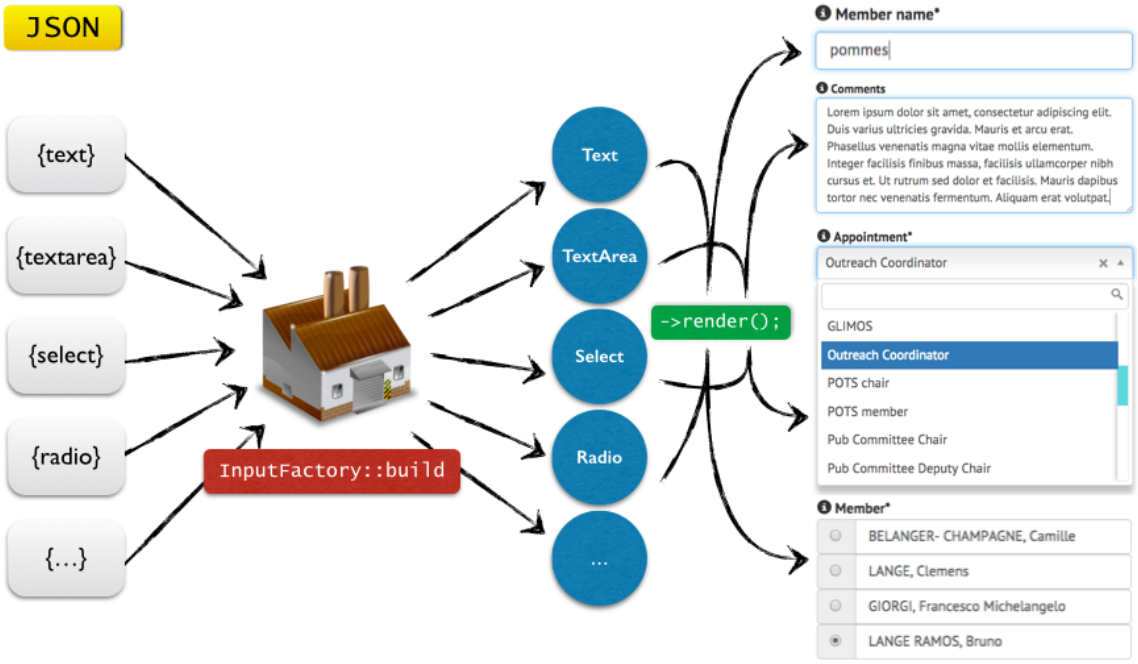


Figura 5.11: Fábrica de *inputs*: instâncias são criadas a partir da descrição de campos de entrada contida nos arquivos de configuração.

baseado no tipo de campo apresentado. Os mais importantes tipos suportados e as classes associadas são mostrados no diagrama de herança simplificado da Figura 5.12. Como cada *input* tem sua própria estrutura de marcação, o método `render` é sobrescrito em cada classe para gerar os respectivos códigos HTML.

5.7.2 Validação de dados de entrada

O Código 5.9 mostra um objeto JSON empregado na construção de um campo de entrada aplicado no formulário de inserção de novos cargos no sistema *Appointment*, já na plataforma proposta. No contexto deste projeto, o atributo `rules` torna-se especialmente importante, visto que as diretrizes ali codificadas frequentemente representam implementações diretas dos requisitos de um sistema. A gerência do ATLAS, por exemplo, impõe que todos os cargos devam ter uma data de término definida e que tenham uma duração mínima de 1 mês. Estas determinações são conjuntamente enforçadas pela ativação da regra `required` e definição do atributo `minDate`, onde a data mínima permitida é calculada somando-se o número de dias especificado em `offset` a uma referência definida em `ref`. Esta referência, por sua

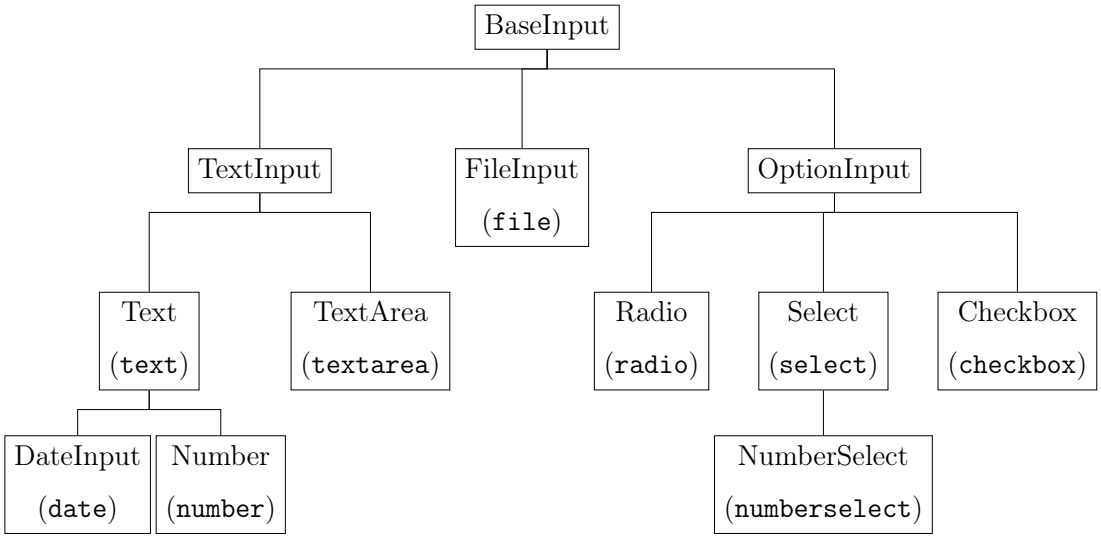


Figura 5.12: Diagrama de classes para a família de *inputs* suportados pelo Fence.

vez, corresponde à data de início escolhida pelo usuário e é, portanto, incorporada dinamicamente ao arquivo de configuração por caracteres especiais.

```
{
  "name": "end_date",
  "type": "date",
  "label": "End date",
  "about": "The appointment end date",
  "rules": {
    "required": true,
    "minDate": {
      "ref": "{%start_date%}",
      "offset": 30
    }
  },
  "edit": {
    "usergroups": [
        ["APPOINTMENT_ADMIN"]
    ]
  }
}
```

Código 5.9: Configuração do campo data término de um *Appointment*.

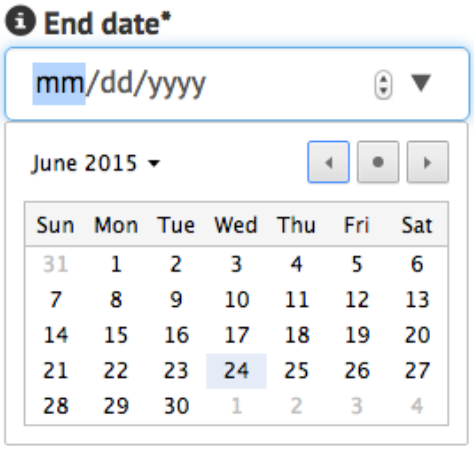


Figura 5.13: Campo de entrada renderizado pelo navegador.

A Tabela 5.3 contém a lista dos possíveis atributos de regras que podem ser associados a um campo de entrada. Uma vez preenchidos pelos usuários, qualquer *input* pode ser validado frente as especificações que os caracterizam por meio de

chamadas ao método público `validate`. Erros de validação são armazenados no atributo público `errors`. Estas rotinas são de vital importância para a garantia da conformidade dos dados inseridos pelos usuários com as regras estabelecidas.

5.7.3 Permissões associadas

Os mesmos atributos de acesso discutidos na Seção 5.3.4 podem ser empregados na descrição de campos de entrada para condicionar permissões de leitura e escrita às credenciais de cada usuário. No Código 5.9, o atributo `edit` define que somente administradores do sistema *Appointment* podem definir a data de término dos cargos gerenciais. Quando esta condição não é verificada, o *input* é bloqueado.

5.7.4 Integração com o Glance

Um último aspecto que cabe mencionar diz respeito à natureza peculiar dos filhos de `OptionInput`, já que os objetos representam um tipo de campo cujos valores estão restritos a uma lista de opções pré-estabelecidas. A população destas opções se dá por um atributo adicional mandatório chamado de `options`. Trata-se de uma lista de objetos que encapsulam as chaves `label` e `value`, podendo ser declarados diretamente nos arquivos de configuração ou invocados indiretamente por meio de chamadas ao Glance. As duas opções são mostradas nos Códigos 5.10 e 5.11, respectivamente.

```
"options": [{
        "label": "Barcelona",
        "value": 17
    },{
        "label": "Berkeley LBNL",
        "value": 21
    },
    ...
]
```

Código 5.10: Listagem direta das opções.

```
"options": [{
        "label": "Select an institute",
        "value": ""
    },{
        "type": "glance",
        "siid": "ATLAS_INSTITUTE_SELECT",
        "cacheable": true
    }
]
```

Código 5.11: População das opções via Glance.

Neste ponto, podemos observar o primeiro canal de integração direta do Fence e

Atributo	Tipo	Situação de falha
`required`	*boolean*	Quando o usuário não submete qualquer valor.
`allowed`	*array*	Quando o valor submetido não se encontra na lista de valores permitidos.
`length`	*number*	Quando o comprimento do texto submetido é diferente do definido.
`minlength`	*number*	Quando o comprimento do texto submetido é menor do que o mínimo definido.
`maxlength`	*integer*	Quando o comprimento do texto submetido é maior do que o máximo definido.
`number`	*boolean*	Quando o servidor recebe uma *string* que não pode ser traduzida em um número. Por exemplo, "42" passa pela verificação, enquanto que "<script>" ou "1.2.3" falham.
`min`	*number*	Aplica-se a instâncias de `Number`, falha quando o valor submetido é menor do que o limite inferior estabelecido.
`max`	*number*	Aplica-se a instâncias de `Number`, falha quando o valor submetido é maior do que o limite superior estabelecido.
`date`	*boolean*	Falha quando o valor recebido pelo servidor não corresponde a uma data válida no formato ISO 8601 (YYYY-MM-DD).
`minDate`	*string* (ISO 8601)	Aplicado a instâncias de `DateInput`, falha quando a data inserida é anteior à data mínima estipulada.
`maxDate`	*string* (ISO 8601)	Aplicado a instâncias de `DateInput`, falha quando a data é posterior à data máxima estipulada.

Tabela 5.3: Atributos do objeto `rules` que pode ser incluído na descrição de campos de entrada.

seus arquivos de configuração com o Glance. De fato, o objeto descrito nas linhas 6 a 10 do Código 5.11 invoca uma instância da classe `Glance`, responsável pela comunicação direta entre as duas plataformas. O atributo `siid` é uma referência nominal a uma interface de busca (*Search Interface*, ou SI), utilizada pelo Glance para, neste caso, recuperar todos os institutos do ATLAS.

Através da ativação da chave `cacheable`, os resultados incorporados são armazenados na sessão do usuário, descartando a necessidade de uma nova chamada ao Glance quando o usuário recarrega a mesma página ou acessa qualquer outra interface que invoque a mesma SI. Esta estratégia em muito melhora a performance das interfaces ao reduzir o número de solicitações ao Glance.

5.8 Formulários

Formulários HTML são agrupamentos de campos de entradas conjuntamente enviados ao servidor para serem normalmente processados e armazenados.

O Fence impõe a tarefa de coordenar a geração de formulários a uma classe dedicada, convenientemente chamada de `Form`. Como é de se esperar, a classe é filha de `Content`, uma vez que representa um conteúdo a ser exibido no corpo de uma interface. No entanto, uma única classe não poderia facilmente se ocupar do processamento dos dados submetidos uma vez que cada formulário tem um propósito distinto. Por este motivo, instâncias diretas de `Form` seriam de pouca utilidade e, portanto, a classe é declarada abstrata. Seu contrato de herança prevê a implementação de um método chamado `post`, onde explicita-se o que deve ser feito com as informações submetidas.

Como herdeiros de `Content`, formulários incorporam arquivos de configuração na geração de seu conteúdo. A modelagem destes arquivos emerge naturalmente da própria definição de um formulário: uma lista ordenada de campos de entrada, podendo, portanto, ser descrito por uma lista de objetos no formato JSON. Apesar desta abordagem funcionar perfeitamente para um reduzido número de *inputs*, formulários longos, com dezenas de campos apresentados em uma única interface, se

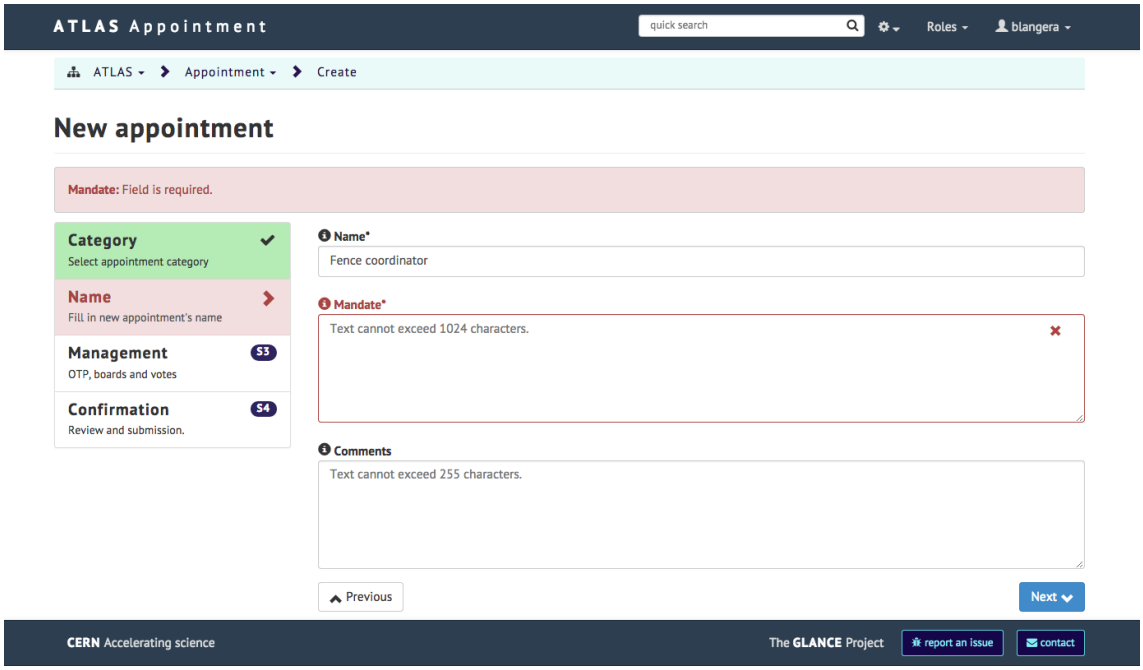


Figura 5.14: Criação de um novo cargo em um formulário de múltiplos passos no sistema *Appointment* gerado com o Fence. O usuário é impedido de avançar ao próximo passo a menos que todos os campos mandatórios sejam preenchidos.

tornam inconvenientes para os usuários. Para estes casos, o Fence permite que o processo de submissão de um formulário seja fracionado em passos, ou *steps*, onde cada passo é formado por um agrupamento lógico ou semântico de um número qualquer de *inputs*.

A Figura 5.14 ilustra o formulário de inserção de novos cargos gerenciais no ATLAS, já no sistema criado com o Fence. À esquerda, pode-se observar que o formulário é composto de três passos definidos no arquivo de configuração e mais um passo de confirmação que é automaticamente incorporado à interface pelo próprio sistema. O usuário é impedido de avançar ao passo seguinte a menos que os valores submetidos observem as regras associadas a cada um dos *inputs* que compõem aquela etapa. Quando qualquer regra é infrigida, o usuário é notificado dos erros em uma barra de alerta no topo da interface, com elementos visuais que reinforçam quais campos necessitam revisão.

Desta forma, o *framework* garante que os dados submetidos através dos formulários estarão sempre de acordo com as regras estabelecidas nos arquivos de configu-

ração. Portanto, requisitos que se expressem nestas diretrizes podem ser facilmente alterados quando necessários, sem quaisquer intervenções no código-fonte das aplicações.

5.8.1 O papel do JavaScript

O *JavaScript* é uma linguagem de programação interpretada, multi-plataforma e orientada a objetos. Todos os navegadores contém implementações de interpretadores especialmente equipados com bibliotecas de manipulação do chamado DOM (*Document Object Model*), convenção estabelecida pela W3C para a representação e interação de objetos em linguagens de marcação como HTML.

Aplicações *Web* modernas se utilizam deste recurso para atribuir grandes partes de suas funcionalidades ao lado do cliente através da execução de *scripts*. De fato, provido da capacidade de completa manipulação do DOM, o *JavaScript* dá bases para a criação de aplicações dinâmicas, onde o conteúdo das páginas pode ser controlado pelo cliente e manipuladores podem ser vinculados à resposta de eventos como o clique do mouse ou a submissão de um formulário, por exemplo, permitindo uma maior interatividade dos sistemas.

Um clássico exemplo da implementação de rotinas no lado do cliente é a validação de campos de um formulário. Normalmente, o processo de submissão de dados ao servidor é precedido por uma verificação das informações inseridas ainda no lado do cliente. Se um campo mandatório for deixado vazio, por exemplo, o *JavaScript* pode impedir a submissão do formulário, informando o usuário acerca do problema detectado e solicitando uma ação corretiva. O usuário se beneficia de um retorno imediato proporcionado pelo próprio lado do cliente e cargas desnecessárias no servidor são evitadas pela ação desta linha de frente.

No entanto, o *JavaScript* pode ser contornado por usuários maliciosos ou mesmo completamente desabilitado nas configurações do navegador. Por esta razão, a validação dos dados submetidos é, primariamente, atribuída ao lado do servidor. O lado do cliente, embora extensivamente aplicado no Fence, é pautado pelo conceito de que seu papel é reservado ao aprimoramento da experiência do usuário somente,

privando os *scripts* de qualquer ofício que envolva funcionalidades centrais das aplicações ou a proteção de dados sensíveis. Embora os recursos adicionais possibilitados pelo *JavaScript* representem consideráveis melhorias na usabilidade das interfaces, sua aplicação não deve ser imprescindível para o cumprimento de requisitos primários. Esta abordagem, conhecida como *graceful degradation* é aplicada no projeto das interfaces sempre que possível.

5.9 Interfaces de busca

Complementando a capacidade de incorpoação de novas informações, a recuperação de dados é de fundamental importância para a gerência dos experimentos do LHC e, portanto, o Fence desenvolveu mecanismos para dar suporte à geração de interfaces de busca através da classe `SearchInterface`. Mais uma vez, por representar um canal de inclusão de conteúdos ao corpo das interfaces, a classe é declarada filha de `Content`.

A particularidade desta abordagem fica por conta do emprego de um **controlador**, um objeto dedicado a abstrair o processo de aplicação de filtros de pesquisa e captação dos resultados. A classe base texttttBaseSearch abre caminho para a implementação de dois controladores aplicados em dois tipos de busca diferentes, a serem explorados com mais detalhes em seguida.

5.9.1 GlanceSearch

O `GlanceSearch` é um controlador que se utiliza do canal de comunicação estabelecido por uma instância da classe `Glance` para a recuperação de dados condicionados a filtros definidos pelos usuários. O arquivo de configuração atrelado a esta classe contém uma descrição dos campos de entrada que, neste contexto, são interpretados como parâmetros de busca. Por isto, a renderização dos *inputs* inclui também os operadores, que dependem do tipo de parâmetro. A descrição de campos de entrada em formato JSON e suas traduções em parâmetros de busca são mostrados na Figura 5.15.

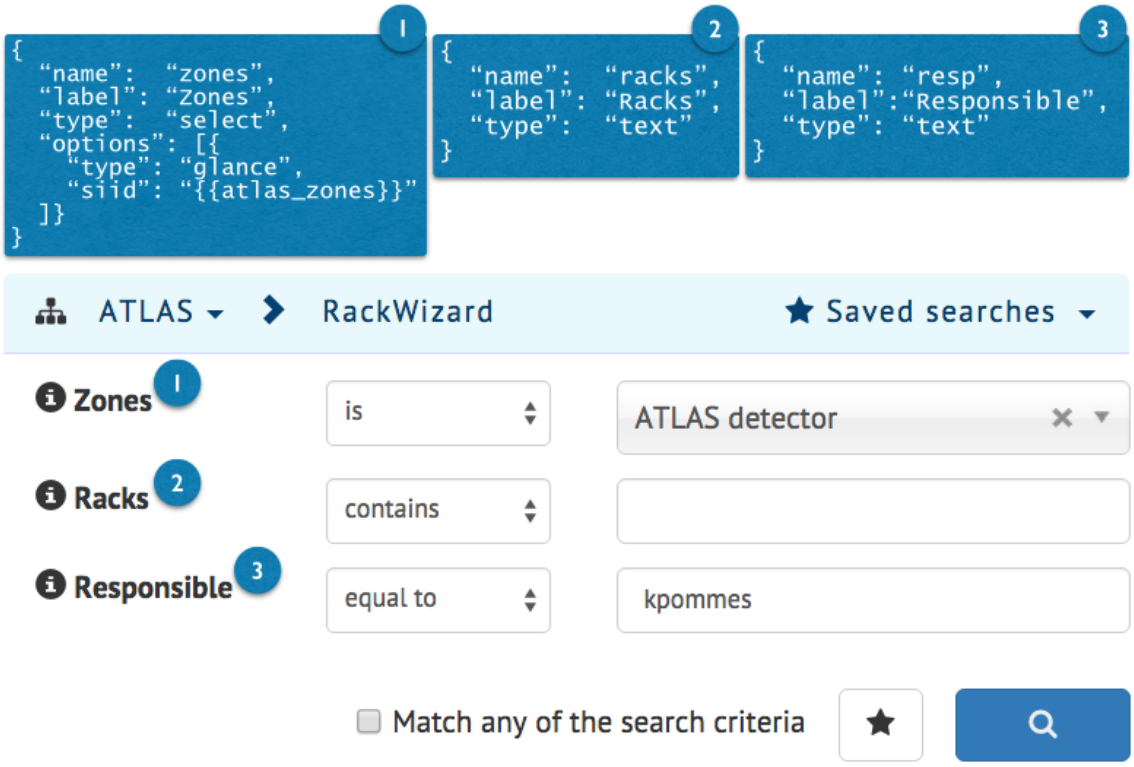


Figura 5.15: Campos de entrada transformados em parâmetros de busca.

Os valores inseridos pelo usuário, bem como os respectivos operadores, são enviados de volta ao servidor e, capturados por `GlanceSearch`, são transformados em filtros e transmitidos a uma instância de `Glance`. A requisição à API do Glance é disparada e os resultados são então transmitidos de volta à `GlanceSearch`, onde são processados e armazenados.

A assimilação dos resultados de uma busca no corpo da interface é incumbida à classe `GlanceSearchInterface`. Seu arquivo de configuração contém uma completa descrição de cada coluna que compõe a tabela onde os resultados são apresentados. Dados vindos do Glance podem ser incorporados à descrição das colunas pelo encapsulamento de identificadores únicos com colchetes duplos. A Figura 5.16 destaca duas colunas na tabela gerada após uma busca realizada no sistema *Papers* do ATLAS. O atributo `header` é transmitido ao cabeçalho da coluna enquanto que `value` pode ser uma *string*, sendo, neste caso, diretamente trasmitido para uma célula, ou um objeto, que pode ser empregado na composição de estruturas mais complexas como links ou botões.

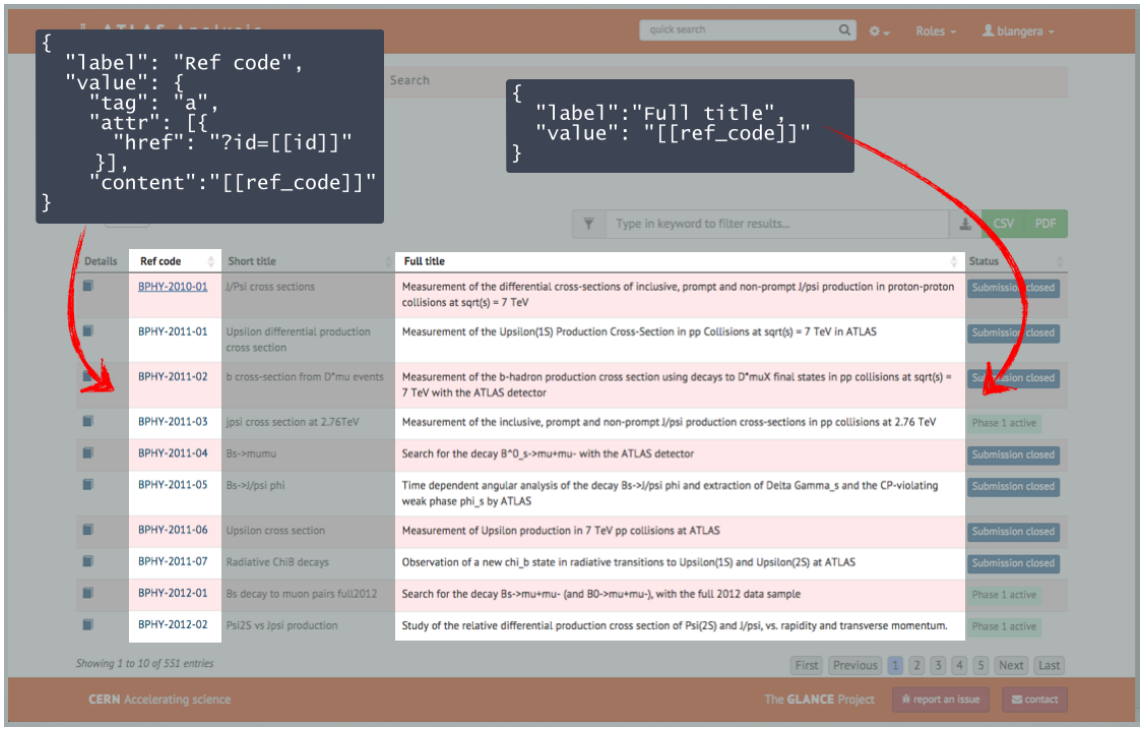


Figura 5.16: *GlanceSearch*: arquivos de configuração descrevem os parâmetros de busca e a tabela resultante.

5.9.2 SuperSearch

O Glance permite que um número qualquer de filtros sejam definidos na composição do critério de busca. A partir deste ponto, usuários podem optar pela recuperação dos dados baseada na observação de todos os parâmetros estabelecidos (o comportamento padrão) ou, alternativamente, na conformidade com ao menos um deles. Isto é, se os filtros devam ser logicamente conectados pelo operador de conjunção `AND` ($\wedge$) ou de disjunção `OR` ($\vee$). A classe `SuperSearch` expande este conceito para permitir a combinação de parâmetros de busca em qualquer arranjo lógico.

Neste contexto, uma **cláusula** é definida pela tripla P.O.V., isto é, um parâmetro, um operador e um valor, selecionados pelo usuário através de três campos de entrada na parte superior do corpo da interface, como ilustrado na Figura 5.17. Cláusulas adicionadas são representadas por nós na chamada *logic workspace*, onde podem ser rearranjadas através da funcionalidade *drag-and-drop* do HTML5 para comporem o critério de busca desejado.

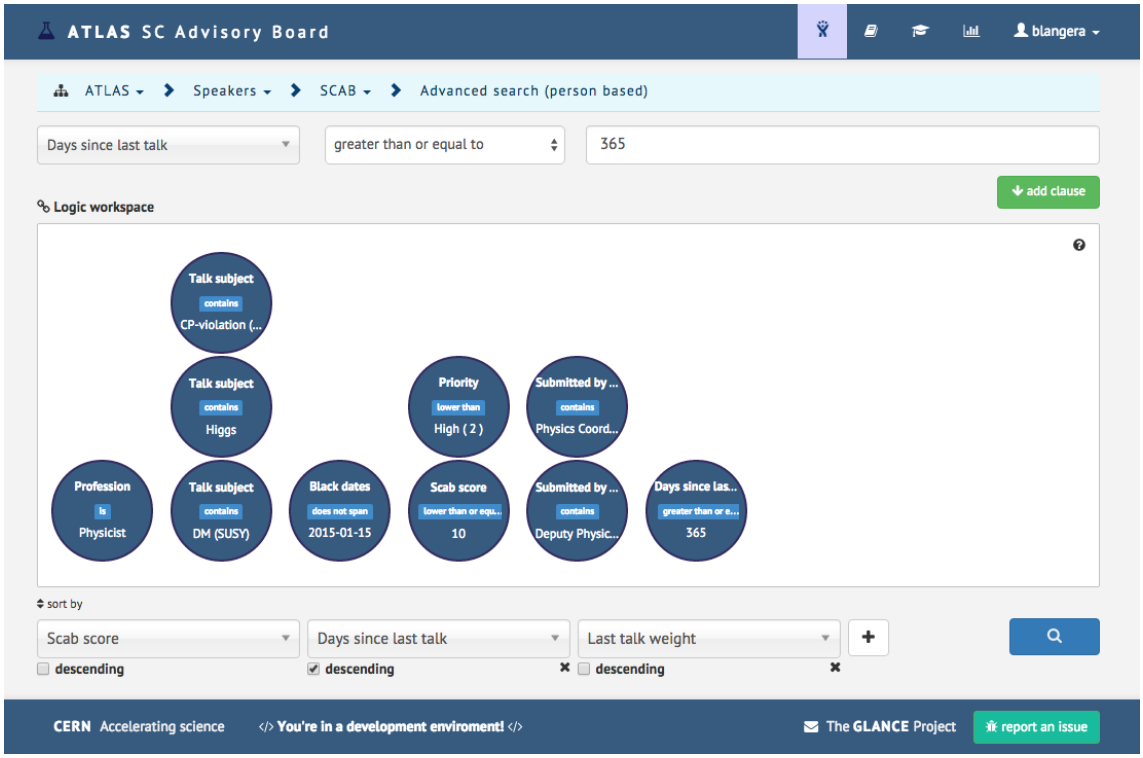


Figura 5.17: *SuperSearch*: solução gráfica para a composição de critérios de busca complexos.

Esta abordagem gráfica proporciona uma visualização simples porém efetiva da composição lógica por trás do critério de busca. Cada pilha (*stack*) representa a conjunção das cláusulas que a compõe. Em outras palavras, nós na mesma coluna são logicamente conectados pelo operador `OR`. Pilhas, por sua vez, conectam-se umas às outras pelo operador `AND`. Esta dinâmica é interativamente apresentada a novos usuários no primeiro acesso, guiando-os na criação de uma lógica simples porém ilustrativa da capacidade desta ferramenta de busca.

O estabelecimento de uma modelagem que pudesse ser utilizada para a transmissão da lógica codificada no *workspace* ao servidor provou ser um grande desafio. Idealmente, o critério deveria ser codificado por completo na URL, de forma que usuários pudessem salvar e compartilhar resultados de buscas sem ter de compor e rearranjar novamente todas as cláusulas envolvidas.

A solução proposta envolve a codificação das cláusulas em uma estrutura hierárquica, onde cada cláusula pode ter zero ou mais pais. À i-ésima cláusula atribui-se

	Parâmetro	Operador	Valor	Pais
c_1	*Profession*	*is*	*Physicist*	-
c_2	*Talk subject*	*contains*	*CP-violation (Bphys)*	c_1
c_3	*Talk subject*	*contains*	*Higgs*	c_1
c_4	*Talk subject*	*contains*	*Dark Matter (SUSY)*	c_1
c_5	*Black dates*	*does not span*	2015-01-15	c_2, c_3, c_4
c_6	*Priority*	$<$	*High* (2)	c_5
c_7	*SCAB score*	$\leq$	10	c_5
c_8	*Submitted by*	*contains*	*Physics Coordinator*	c_6, c_7
c_9	*Submitted by*	*contains*	*Dep. Physics Coordinator*	c_6, c_7
c_{10}	*Days since last talk*	$\geq$	365	c_8, c_9

Tabela 5.4: Arranjo de cláusulas para o examplo mostrado na Figura 5.17.

o identificador c_i. A interconexão entre os nós pode ser, simples o bastante, reconstruída pela definição de seus pais como o conjunto de cláusulas que compõem a pilha encontrada à esquerda do nó em questão. Para o exemplo mostrado na Figura 5.17, a descrição correspondente pode ser encontrada na Tabela 5.4.

Os componentes POV e os pais de cada cláusula são facilmente serializados em uma URL. A classe `SuperSearch` se encarrega da reconstrução da lógica no lado do servidor, com isso, aplica os filtros necessários. O exemplo mostrado na Figura 5.17 traduz-se na seguinte expressão:

$$c_1 \wedge (c_2 \vee c_3 \vee c_4) \wedge c_5 \wedge (c_6 \vee c_7) \wedge (c_8 \vee c_9) \wedge c_{10}$$

O emprego da *SuperSearch* é justificado quando é preciso combinar-se parâmetros de busca em arranjos lógicos mais complexos. Esta necessidade foi inicialmente detectada pela observação do modelo de trabalho do SCAB (*Speakers Committee Advisory Board*), onde o resultado de diversas interfaces de busca era combinado para chegar-se a uma lista de membros qualificados para representar o ATLAS em conferências internacionais.

Neste contexto, a ordem em que os resultados são dispostos na página de resul-

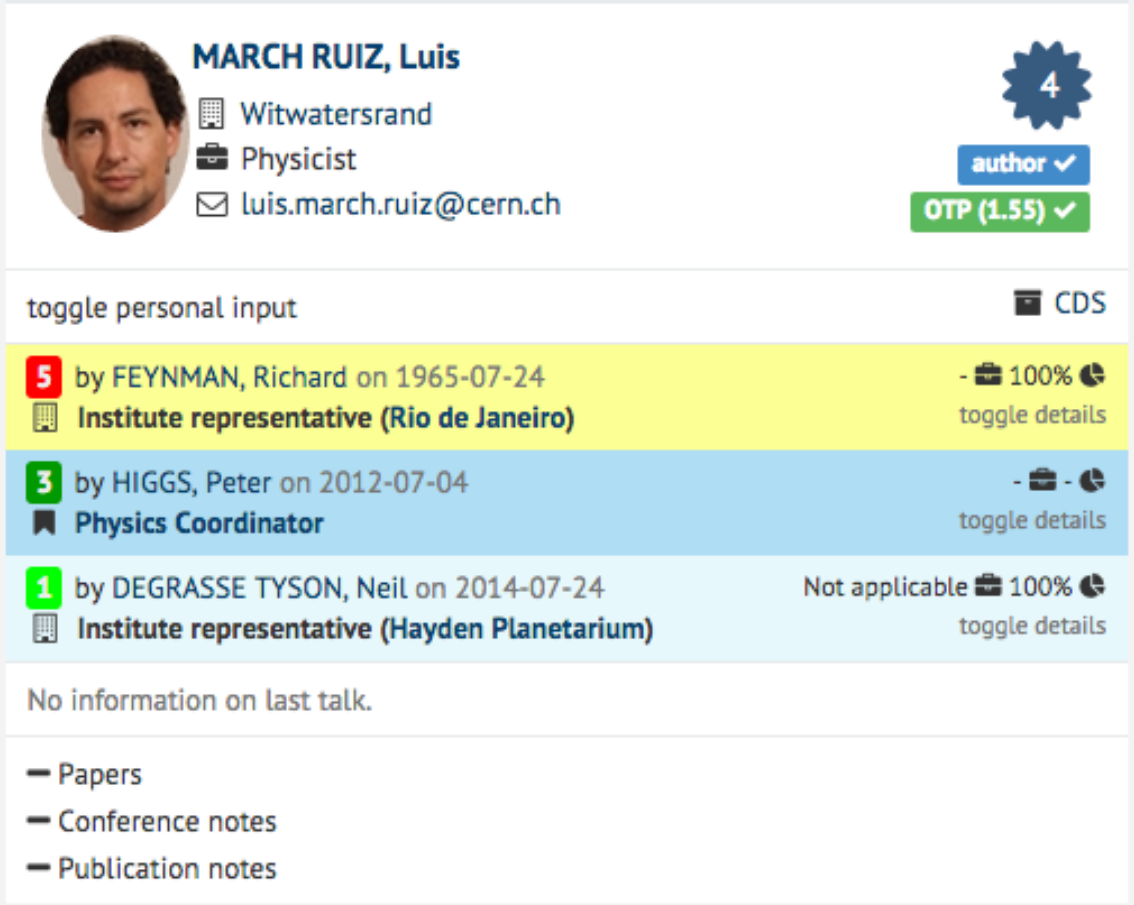


Figura 5.18: Candidato hipotético recuperado após o estabelecimento de um critério complexo de busca com a *SuperSearch*.

tados se faz de vital importância pois é de interesse do SCAB organizar os possíveis candidatos de acordo com critérios pré-estabelecidos, fazendo com que membros de maior preferência apareçam primeiro. Para isto, a *SuperSearch* permite a definição de parâmetros de ordenamento. No exemplo ilustrado na Figura 5.17, estabeleceu-se que o chamado SCAB *score* deveria ser avaliado como primeiro critério de classificação. Caso haja empates neste quesito, o número de dias desde a última palestra deve ser utilizado, priorizando aqueles que passaram mais tempo sem participar de uma conferência, e assim por diante.

A Figura 5.18 mostra um candidato hipotético recuperado para a avaliação do SCAB. A rica estrtutura na qual este resultado é exibido, em contraste com a tabela padrão mostrada na Figura 5.15, é resultado da incorporação de uma tecnologia que reforça a modelagem de desenvolvimento MVC (*Model-View-Controller*). O

Twig [55] é um *framework* recentemente incorporado ao Fence que encoraja o uso de *templates*, arquivos dedicados que estruturam a resposta HTML das aplicações. Além de facilitar a manutenção dos sistemas, esta abordagem também melhora suas performances. De fato, as buscas realizadas pelo sistema SCAB desenvolvido com a *SuperSearch* mostram um fator de redução no tempo de resposta de, em média, 78%.

5.10 Sistemas migrados

Pelo fato de ser a mais simples das aplicações, o sistema *Appointment* do ATLAS foi o primeiro a ser migrado para a nova plataforma. A investida foi tratada como uma espécie de teste, uma prova de conceito das capacidades do *framework*. Passados 5 mêses desde a conclusão do sistema e sua disponibilização para a colaboração, a experiência mostrou resultados que confirmaram as expectativas. Subsidiado pelo Fence, o novo sistema foi concebido em pouco mais de 3 mil linhas de código, o que representa uma redução de 61% comparada à aplicação antiga (ver Figura 3.7). Dos incidentes, um único *bug* foi registrado no JIRA e as eventuais mudanças de requisito, notavelmente motivadas pela alteração das regras de acesso a certas interfaces, foram capazes de ser prontamente atendidas sem grandes intervenções no código-fonte. A Figura 5.19 mostra a interface de gerência dos cargos disponibilizada para os administradores do sistema.

Paralelamente ao *Appointment*, o Fence foi comissionado para a substituição do *RackWizard*. Embora de vital importância para a coordenação técnica do ATLAS, especialmente em períodos de manutenção do detector, a aplicação apresentava problemas de compatibilidade com versões modernas do Java e, com isso, encontrava-se em um estado praticamente disfuncional para grande parte da colaboração. No momento da escrita deste projeto, uma nova versão do sistema desenvolvido com o Fence estava em sua versão beta, sendo amplamente testado por uma ação coordenada de usuários designados. A previsão de lançamento em produção é para Outubro de 2015.

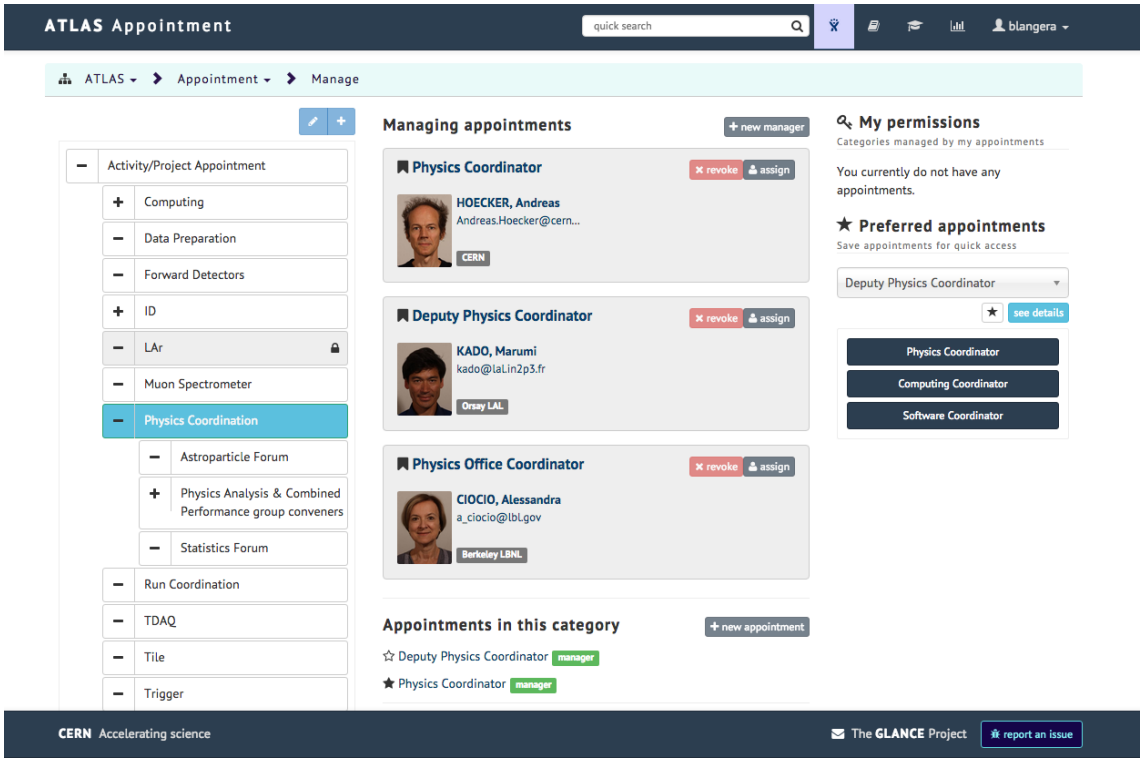


Figura 5.19: Sistema *Appointment* desenvolvido com o Fence.

O desenvolvimento do *RackWizard* contribuiu mutuamente para a evolução do próprio Fence, uma vez que o *framework* foi expandido para atender as particulares especificações do sistema. Destaca-se as significativas contribuições na incorporação de tecnologias de suporte a soluções gráficas, como as ilustradas na Figura 5.20. Desenvolvidas inicialmente para a renderização e manipulação de equipamentos como *racks* e *crates*, a absorção das classes pelo Fence permite que as inovações introduzidas sejam imediatamente disponibilizadas para seus empregos em outras aplicações por toda a equipe de desenvolvedores.

No ALICE, o novo sistema *Membership* está em fase final de testes. À exemplo do *RackWizard*, seu desenvolvimento agregou diversas inovações ao Fence que poderão ser utilizadas nas novas versões do sistema para o ATLAS e o LHCb. A Figura 5.21 mostra a página do perfil de um membro do ALICE renderizado por intermédio de uma classe dedicada que, por sua vez, é controlada por diretrizes armazenadas em arquivos de configuração. Neste tipo de interface, os arquivos de configuração tipicamente controlam o posicionamento e a visibilidade de cada seção de acordo com as credenciais do usuário.

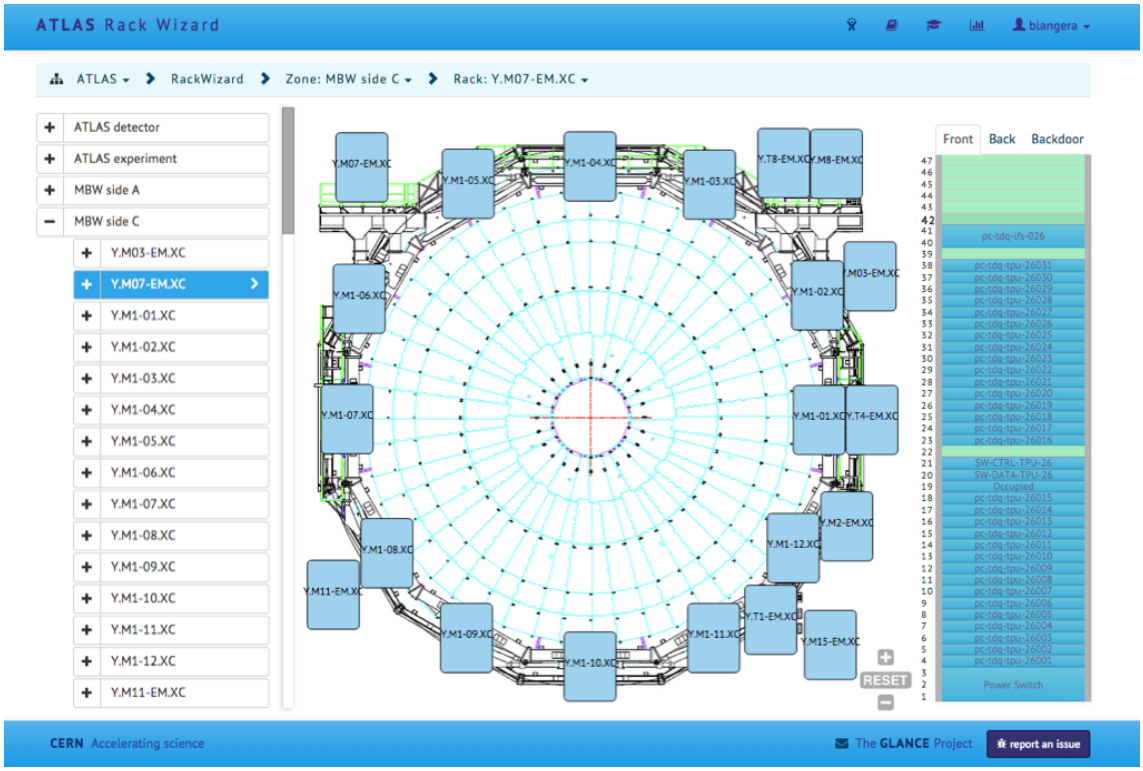


Figura 5.20: Soluções gráficas desenvolvidas para o *RackWizard* absorvidas pelo Fence para o uso em outras aplicações.

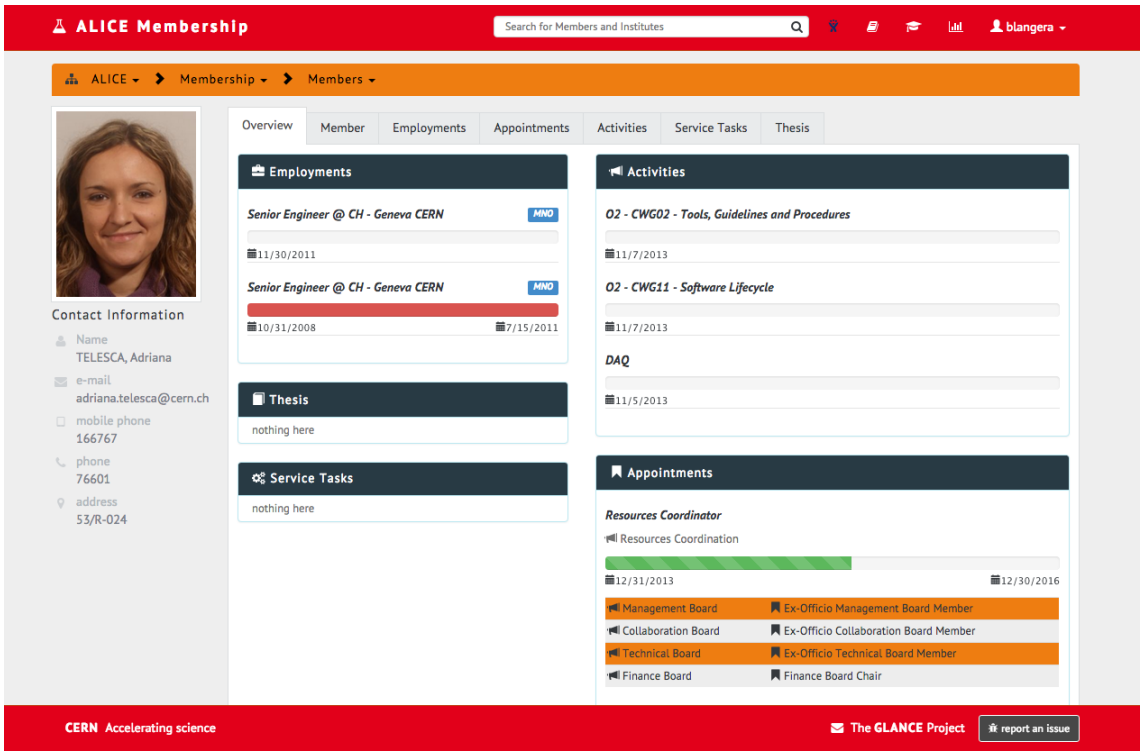


Figura 5.21: Sistema *Membership* desenvolvido para o ALICE.

Capítulo 6

Conclusão

No que representa o mais ambicioso experimento científico já realizado, o LHC e seus experimentos representam uma colaboração internacional de mais de 10 mil físicos, engenheiros, técnicos e estudantes. A heterogeneidade de países, instituições e especialidades responde por grande parte do sucesso das colaborações mas também representa desafios em suas gestões pela grande diversidade de práticas, necessidades e tecnologias empregadas.

Para que dados de diferentes fontes pudessem ser integrados, o Glance foi proposto em 2003 e incorporado na concepção de 21 sistemas cujas aplicações variam do monitoramento de níveis de radiação dos equipamentos na caverna do LHCb, passando pela gerência de membros do ALICE até o controle do processo de publicação de artigos científicos no ATLAS.

Este projeto de graduação consiste na concepção e implementação de uma plataforma de suporte ao desenvolvimento conjunto e eficiente destes sistemas. Um *framework* que disponibilize ferramentas para o compartilhamento de funcionalidades e que tenha em sua estrutura mecanismos de rápida resposta à natureza dinâmica dos requisitos das aplicações. O **Fence** foi proposto a partir de um estudo de tecnologias e estratégias de reuso de código e minimização de impacto de mudanças, dando origem a uma abordagem que combina a flexibilidade proporcionada por arquivos de configuração com a solidez e eficiência da orientação a objetos.

A solicitação de uma grande reestruturação do *Appointment* pela gerência do ATLAS foi vista como a oportunidade ideal para o desenvolvimento por completo do sistema já sobre a nova plataforma. Em pouco menos de um mês investido, a aplicação foi a primeira a ser migrada por completo. Como resultado, o novo sistema não só absorveu com facilidade os novos requisitos como o fez em 1/3 das linhas de código necessárias na implementação do antigo. A baixa incidência de *bugs* e a capacidade de alterar regras do sistema sem qualquer intervenção no código-fonte deu suporte ao emprego do Fence na geração dos demais sistemas.

Em especial, o *framework* teve a oportunidade de ser testado na concepção de uma aplicação mais complexa. Uma nova versão do sistema *RackWizard* foi proposta e implementada, sendo atualmente avaliada por usuários designados para ser lançada em produção dentro dos próximos meses.

Novas classes foram incorporadas a estrutura do Fence para que a plataforma pudesse dar suporte a soluções gráficas impostas pelo *RackWizard*, o que até então não era imperativo para o cumprimento dos requisitos nos demais sistemas. Em todo caso, as funcionalidades desenvolvidas encontram-se à disposição do time de desenvolvedores para ser incorporado a qualquer aplicação quando for necessário. Com a adoção de um padrão de codificação e de ferramentas para a geração da documentação a partir de comentários especialmente formatados, a leitura e compreensão do código é facilitada, o que aumenta a eficiência dos esforços de manutenção e implementação de melhorias pelos desenvolvedores.

Um outro exemplo da incorporação de novas tecnologias ao *framework* teve suas origens nas dificuldades encontradas pelo SCAB na combinação de mais de 40 parâmetros de busca para classificar e garantir uma justa seleção de membros do ATLAS para representar a colaboração em conferências internacionais. Como resposta, introduziu-se a chamada *SuperSearch*, uma ferramenta de busca onde os parâmetros podem ser dispostos em qualquer arranjo lógico, permitindo que complexos critérios sejam definidos na recuperação de dados. Tão logo disponibilizada uma versão beta para a avaliação de alguns membros do SCAB, a *SuperSearch* revelou novos modelos de trabalho que trouxeram consigo uma série de requisitos conse-

quentes. Conclui-se que a instituição de soluções inovadoras desencadeia um ciclo de realimentação positiva que mantém os sistemas em constante evolução.

Dentre as novas funcionalidades que já estão em processo de desenvolvimento para serem incorporadas ao Fence, destacam-se o chamado *Notification Center* - um sistema centralizado para a gerência das notificações enviadas pelo Fence onde usuários poderão acessar o histórico de alertas enviados, bem como inspecionar quaisquer ações pendentes, e o *WebEditor*, uma interface de edição de arquivos de configuração que permitirá que usuários credenciados alterem o comportamento dos sistemas sem o intermédio de um desenvolvedor.

As preocupações com a segurança dos dados exibidos e a regulação das informações submetidas foram também abordadas na implementação do *framework*. O estabelecimento de atributos de acesso garantiu que campos de entrada fossem manipulados somente por usuários autorizados de acordo com seus grupos (ou *e-groups*). Quando a edição de *inputs* foi permitida, os dados enviados foram corretamente submetidos a rotinas de validação de acordo com as regras definidas na descrição de cada campo em um arquivo de configuração. As verificações no lado do cliente se mostraram de grande conveniência mas a garantia da conformidade dos dados submetidos se manteve mesmo com o *JavaScript* desabilitado.

Além disso, a designação do lado do servidor como mediador da comunicação com o Glance tornou a incorporação de dados em campos de entrada e em interfaces de busca consideravelmente mais rápida, quando comparada a utilização de solicitações via AJAX realizadas no lado do cliente. Mais além, a capacidade de armazenamento temporário das chamadas à API do Glance na sessão do usuário foi responsável por consideráveis melhoras no tempo de geração das interfaces.

Os mecanismos de reação do Fence a eventuais anomalias nos sistemas tornaram possível a rápida reprodução dos erros, passo crucial na resolução de incidentes. Embora não tivesse sido concebida necessariamente para este propósito, a classe `Mailer` foi incorporada no tratamento de erros do *framework* para a imediata notificação da equipe de desenvolvimento, com informações úteis já imbutidas no corpo do email. O detalhado registro da execução de códigos pela classe `Logger` se mos-

trou de grande valia para a reconstrução de passos do usuário até o instante em que a mensagem de erro lhe é apresentada.

Pelos fatos apresentados, conclui-se que o Fence observa os requisitos levantados e pode ser utilizado como uma plataforma de integração dos sistemas dos experimentos do LHC. De fato, com 3 sistemas já migrados e a proximidade na conclusão de outros 2, a expectativa é de que todos os 16 sistemas restantes possam ser migrados em curto e médio prazos.

Referências

[1] The birth of the web.
Acessado em Fevereiro de 2015.
`http://home.web.cern.ch/topics/birth-web/`.

[2] James D Herbsleb and Deependra Moitra. Global software development. *Software, IEEE*, 18(2):16–20, 2001.

[3] Audris Mockus and James Herbsleb. Challenges of global software development. In *Software Metrics Symposium, 2001. METRICS 2001. Proceedings. Seventh International*, pages 182–184. IEEE, 2001.

[4] Karl Wiegers and Joy Beatty. *Software requirements*. Pearson Education, 2013.

[5] Susan DP Harker, Ken D Eason, and John E Dobson. The change and evolution of requirements as a challenge to the practice of software engineering. In *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on*, pages 266–272. IEEE, 1993.

[6] Matthias Jarke and Kalle Lyytinen. Editorial: "complexity of systems evolution: Requirements engineering perspective". *ACM Transactions on Management Information Systems (TMIS)*, 5(3):11, 2015.

[7] Johann Rost and Robert L Glass. *The dark side of software engineering: evil on computing projects*. John Wiley & Sons, 2011.

[8] The Large Hadron Collider. Acessado em Fevereiro de 2015.
`http://home.web.cern.ch/topics/large-hadron-collider`.

[9] LHC Season 2: facts & figures. Acessado em Julho de 2015. `http://press.web.cern.ch/backgrounders/lhc-season-2-facts-figures`.

[10] THE ATLAS Collaboration. The ATLAS experiment at the CERN Large Hadron Collider. *J. Instrum*, 3:S08003, 2008.

[11] ATLAS. Acessado em Janeiro de 2015. `http://home.web.cern.ch/about/experiments/atlas`.

[12] S Chatrchyan, G Hmayakyan, V Khachatryan, AM Sirunyan, W Adam, T Bauer, T Bergauer, H Bergauer, M Dragicevic, J Erö, et al. The CMS experiment at the CERN LHC. *Journal of Instrumentation*, 3(08):S08004, 2008.

[13] Georges Aad, T Abajyan, B Abbott, J Abdallah, S Abdel Khalek, AA Abdelalim, O Abdinov, R Aben, B Abi, M Abolins, et al. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Physics Letters B*, 716(1):1–29, 2012.

[14] Serguei Chatrchyan, Vardan Khachatryan, Albert M Sirunyan, A Tumasyan, W Adam, E Aguilo, T Bergauer, M Dragicevic, J Erö, C Fabjan, et al. Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Physics Letters B*, 716(1):30–61, 2012.

[15] ALICE Collaboration, K Aamodt, et al. The ALICE experiment at the CERN LHC. *Jinst*, 3(420):S08002, 2008.

[16] ALICE. Acessado em Fevereiro de 2015. `http://home.web.cern.ch/about/experiments/alice`.

[17] A Augusto Alves Jr, LM Andrade Filho, AF Barbosa, I Bediaga, G Cernicchiaro, G Guerrer, HP Lima Jr, AA Machado, J Magnin, F Marujo, et al. The LHCb detector at the LHC. *Journal of instrumentation*, 3(08):S08005, 2008.

[18] K K Galvao C Maidantchik, F F Grael and K Pommes. Glance project: a database retrieval mechanism for the ATLAS detector. *J. Phys.: Conf. Ser*, 2008.

[19] Shawn A Bohner. Software change impact analysis. 1996.

[20] Shawn Bohner et al. Extending software change impact analysis into cots components. In *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*, pages 175–182. IEEE, 2002.

[21] Keith H Bennett and Václav T Rajlich. Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 73–87. ACM, 2000.

[22] Greg Wilson, DA Aruliah, C Titus Brown, Neil P Chue Hong, Matt Davis, Richard T Guy, Steven HD Haddock, Katy Huff, Ian M Mitchell, Mark D Plumbley, et al. Best practices for scientific computing. *PLoS biology*, 12(1):e1001745, 2014.

[23] Ben Smith. Object-Oriented Programming. In *Advanced ActionScript 3*, pages 1–23. Springer, 2015.

[24] Convention for the Establishment of a European Organization for Nuclear Research. Acessado em Dezembro de 2014. `http://council.web.cern.ch/council/en/Governance/Convention.html`.

[25] The Standard Model of Particle Physics. Acessado em Dezembro de 2014. `http://www.symmetrymagazine.org/standard-model/`.

[26] CERN mug summarizes Standard Model, but is off by a factor of 2. Acessado em Março de 2015. `http://www.quantumdiaries.org/2011/06/26/cern-mug-summarizes-standard-model-but-is-off-by-a-factor-of-2`.

[27] The accelerator complex. Accessado em Março de 2015. `http://home.web.cern.ch/about/accelerators`.

[28] Christiane Lefevre. LHC: the guide (English version). Technical report, 2009.

[29] Werner Herr and Bruno Muratori. Concept of luminosity. 2006.

[30] Martin Hilbert and Priscila López. The world's technological capacity to store, communicate, and compute information. *science*, 332(6025):60–65, 2011.

[31] Jamie Shiers. The worldwide LHC computing grid (worldwide LCG). *Computer physics communications*, 177(1):219–223, 2007.

[32] ATLAS Photos. Accessado em Março de 2015. `http://www.atlas.ch/photos/`.

[33] CMS. Accessado em Março de 2015. `http://home.web.cern.ch/about/experiments/cms`.

[34] ALICE experiment sets record for the hottest spot in the Universe. Acessado em Setembro de 2014. `http://alicematters.web.cern.ch/?q=QM12_temp_rec`.

[35] The LHCb Upgrade. Accessado em Março de 2015. `http://ph-news.web.cern.ch/content/lhcb-upgrade`.

[36] Chase Econometrics Associates. Relative impact of nasa expenditure on the economy. Technical report, University of Wisconsin - Madison Computer Sciences Department, March 1975.

[37] The Burning Question. Accessado em Fevereiro de 2015. `http://atlas-service-enews.web.cern.ch/atlas-service-enews/2009/news_09/news_excitement.php`.

[38] David Ferraiolo, Janet Cugini, and D Richard Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings of 11th annual computer security application conference*, pages 241–48, 1995.

[39] O Beltramello, HJ Burckhart, S Franz, M Jaekel, M Jeckel, S Lüders, G Morpurgo, F dos Santos Pedrosa, K Pommes, and H Sandaker. The detector safety system of the ATLAS experiment. *Journal of Instrumentation*, 4(09):P09012, 2009.

[40] F Glege. The Rack Wizard, a graphical database interface for electronics configuration. 2003.

[41] Kaio Karam Galvão, Kathy Pommès, Jorge Molina-Pérez, Carmen Maidantchik, and Felipe Fink Grael. Management of Equipment Databases at CERN for the Atlas Experiment. In *Astroparticle, Particle and Space Physics, Detectors and Medical Physics Applications*, volume 1, pages 781–785. World Scientific, 2008.

[42] Susan Harker. Requirements specification and the role of prototyping in current practice. In *Taking software design seriously*, pages 339–354. Academic Press Professional, Inc., 1991.

[43] HJ Bullinger and J Niemeier. Computer Integrated Business: Corporate and Information System Management in Competitive Environments. *EURINFO*, 88.

[44] Judith Ramsay, Alessandro Barbesi, and Jenny Preece. A psychological investigation of long retrieval times on the World Wide Web. *Interacting with computers*, 10(1):77–86, 1998.

[45] Introducing JSON. Accessado em Fevereiro de 2015. `http://json.org`.

[46] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of JSON and XML Data Interchange Formats: A Case Study. *Caine*, 2009:157–162, 2009.

[47] Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format. 2014.

[48] Git - distributed is the new centralized. Accessado em Junho de 2015. `https://git-scm.com/`.

[49] Semantic Versioning. Accessado em Fevereiro de 2015. `http://semver.org/`.

[50] A Successful Git Workflow. Accessado em Outubro de 2013. `http://nvie.com`.

[51] phpDocumentor analyzes your code to create great documentation. Accessado em Junho de 2015.
`http://www.phpdoc.org/`.

[52] Coding Style Guide. Accessado em Julho de 2015.
`https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-2-coding-style-guide.md`.

[53] Atlassian. *JIRA - Issue & Project Tracking Software*. 2015. https://www.atlassian.com/software/jira.

[54] Y Sugimori, K Kusunoki, F Cho, and S Uchikawa. Toyota production system and kanban system materialization of just-in-time and respect-for-human system. *The International Journal of Production Research*, 15(6):553–564, 1977.

[55] Twig: The flexible, fast, and secure template engine for PHP. Accessado em Maio de 2015.
`http://twig.sensiolabs.org/`.