UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

ALYSSON GOMES DE SOUSA

DESENVOLVIMENTO DE UMA FERRAMENTA WEB PARA CONSTRUÇÃO DE APLICAÇÕES NA PLATAFORMA DE COMPUTAÇÃO DE ALTO DESEMPENHO ORIENTADA A COMPONENTES HPC-*SHELF*

QUIXADÁ

2017

ALYSSON GOMES DE SOUSA

DESENVOLVIMENTO DE UMA FERRAMENTA WEB PARA CONSTRUÇÃO DE APLICAÇÕES NA PLATAFORMA DE COMPUTAÇÃO DE ALTO DESEMPENHO ORIENTADA A COMPONENTES HPC-*SHELF*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Orientador: Prof. Me. Wagner Guimarães Al-Alam

Coorientador: Prof. Dr. Jefferson de Carvalho Silva

QUIXADÁ

2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S696d Sousa, Alysson Gomes de.
Desenvolvimento de uma ferramenta web para construção de aplicações na plataforma de computação de alto desempenho orientada a componentes HPC-Shelf / Alysson Gomes de Sousa. – 2017.
97 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2017.
Orientação: Prof. Me. Wagner Guimarães Al-Alam.
Coorientação: Prof. Dr. Jefferson de Carvalho Silva.

1. Sites da Web-Desenvolvimento. 2. Computação de alto desempenho. 3. Interfaces de usuários (Sistemas de computação). 4. Componente de software. 5. Fluxo de trabalho. I. Título.

CDD 005

ALYSSON GOMES DE SOUSA

DESENVOLVIMENTO DE UMA FERRAMENTA WEB PARA CONSTRUÇÃO DE APLICAÇÕES NA PLATAFORMA DE COMPUTAÇÃO DE ALTO DESEMPENHO ORIENTADA A COMPONENTES HPC-*SHELF*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Aprovada em: ___/___/_____

BANCA EXAMINADORA

Prof. Me. Wagner Guimarães Al-Alam (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Jefferson de Carvalho Silva (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Paulo Victor Barbosa de Sousa
Universidade Federal do Ceará (UFC)

Este trabalho é dedicado especialmente a minha mãe Raquel Moraes da Silva Sousa que já descansa no Senhor. Também é dedicado a meu pai Almir Gomes de Sousa e meu irmão Arley Gomes de Sousa.

AGRADECIMENTOS

Agradeço primeiramente a Deus, porque até aqui Ele tem me sustentado. Agradeço aos meus pais Almir Gomes e Raquel Moraes, que me educaram e me instruíram, me permitindo ser a pessoa que sou hoje.

Agradeço aos professores Wagner G. Al-Alam e Jefferson de C. Silva pela orientação durante a realização do projeto de iniciação científica, e durante o trabalho de conclusão de curso.

Agradeço ao professor Davi Romero, por me aceitar como bolsista do Programa de Ensino Tutorial (PET) do curso de Sistemas de Informação, além de seus excelentes conselhos para carreira acadêmica e profissional.

Agradeço aos demais professores pelo vasto conhecimento que me foi passado, e sem dúvida enriqueceram minha formação. Agradeço aos meus amigos do conjunto de jovens da igreja a qual faço parte, que me receberam e me ajudaram direta ou indiretamente durante a minha trajetória.

Agradeço aos meus amigos da turma a qual faço parte e do grupo de bolsista da qual estou inserido, que contribuíram direta ou indiretamente para o resultado deste trabalho.

"O temor do Senhor é o princípio da sabedoria, e
o conhecimento do Santo é entendimento."

(Bíblia Sagrada, Provérbios 9, 10)

RESUMO

Tomando como plano de fundo a plataforma de computação de alto desempenho orientada a componentes HPC-*Shelf*, foi identificado uma necessidade inerente a um dos objetivos da HPC-*Shelf*, que é fornecer um ambiente em que desenvolvedores possam construir aplicações computacionalmente intensivas e usuários possam usufruir destas aplicações. Para atender esta necessidade foi concebida a ferramenta *web* I*Shelf* que provê um conjunto de interfaces que auxiliam os desenvolvedores a construírem suas aplicações exibindo graficamente os principais conceitos da HPC-*Shelf*, como componentes e *workflows*, permitindo que os mesmos sejam manipulados dinamicamente nas interfaces e registrados na plataforma. Para tanto foi definido um processo de desenvolvimento que inclui as fases de levantamento e análise dos requisitos, prototipação, modelagem, implementação, teste e validação da solução. Na ultima fase do desenvolvimento foi realizada uma avaliação da qualidade da ferramenta para determinar o nível de esforço necessário para realizar tarefas comuns aos contexto da I*Shelf*. Por fim, este trabalho apresenta, além do relato de experiência sobre o desenvolvimento, a descrição dos resultados obtidos pela execução do processo.

Palavras-chave: Sites da *Web*-Desenvolvimento. Computação de alto desempenho. Interfaces de usuários (Sistemas de computação). Componente de software. Fluxo de trabalho.

ABSTRACT

Taking as background the component-oriented high-performance computing platform HPC-Shelf, was identified a need to one of HPC-Shelf's objectives, which is to provide an environment in which developers can build highly computational applications and user can get the best of this applications. To attend this demand was conceived the web tool IShelf, that provides a set of interfaces to aid the developers to build theirs applications graphically showing the core concepts of HPC-Shelf, such as components and workflows, allowing that this elements are dynamically manipulated on interface and registered on the plataform. To do that was defined a development process that contains the phases of gathering and analysis of requeriments, prototyping, modeling, implementation, testing, and validation of the solution. On the last development phase was used a quality evaluation to measure how much effort was necessary to accomplish common tasks on IShelf context. Finally, this paper presents, in addition to the experience report on development, the description of the results obtained on the execution of the process.

Keywords: Websites-Development. High Performance Computing. User interface (Computer Systems). Software Component. Workflow.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos a computação tem se deparado com problemas computacionais extremamente dispendiosos, muitos deles têm origem nos avanços científicos, da qual as plataformas de execução disponíveis não tem poder de processamento necessário para obter soluções em tempo satisfatório. Por conta disso surgiu a computação de alto desempenho (CAD), a mesma consiste em uma subárea da computação que busca ter uma melhor compreensão sobre a natureza destes problemas, além de desenvolver técnicas para otimizar os recursos computacionais (sejam eles *softwares* ou *hardwares*) que também irão tratar destes problemas.

Atualmente diversas tecnologias têm dado suporte a resolução de problemas computacionalmente intensivos, dentre elas a *computação em nuvem* tem se mostrado muito promissora e vem ganhando espaço no mercado. A Computação em nuvem pode ser definida como um modo que permite de forma ubíqua, e conveniente, o acesso sob demanda a um conjunto de recursos configuráveis de computação compartilhada, fornecendo diversas formas de abstrair os recursos conforme o modo de utilização, por meio dos diversos tipos de perfis de serviços que se adequam às mais diversas necessidades (MELL *et al.*, 2011). Assim, as nuvens computacionais vem ocupando um espaço considerável no contexto de computação distribuída, despertando interesse nos especialistas da CAD.

Além da dispendiosidade, outro fator importante é a complexidade das aplicações que tratam de problemas com alta demanda computacional. O crescente volume de dados tem exigido aplicações cada vez mais robustas e eficientes, o que torna ainda difícil a construção destas aplicações. Dessa forma, para lidar com a complexidade destas aplicações, os especialistas da CAD têm usado o paradigma de *programação orientada a componentes* (POC), este paradigma surgiu na indústria de *hardware* e tem conquistado espaço nas fábricas de *software*. No contexto de *softwares*, este paradigma fragmenta a aplicação em componentes, sendo eles desenvolvidos e validados de forma independente, que ao final se unem para compor a solução computacional; por serem componentes autocontidos e independentes, o reuso é estimulado, o que proporciona um ganho de produtividade, diminuindo o custo de desenvolvimento (WANG; QIAN, 2005).

Outro recurso que permitiu a integração entre as técnicas citadas anteriormente, são os *workflows*. Apesar de seu surgimento ter se dado no meio corporativo, no contexto da ciência da computação *workflows* podem ser definidos como uma sequência de etapas ou atividades que descrevem o processo de um experimento computacional (BARKER; HEMERT, 2007). No contexto da CAD, o conceito de *workflow* tem sido muito útil em experimentos computacionais

que utilizam recursos distribuídos, onde estes recursos podem ser abstraídos como componentes que serão orquestrados para realizarem o experimento (SILVA, 2016).

Dada a diversidade de técnicas e recursos baseado em componentes, existem *frameworks* que integram alguns destes recursos para auxiliar os especialistas da CAD a desenvolverem suas aplicações, como: *Common Component Architecture*[1] (CCA), *Fractal*[2], *Grid Component Model*[3] (GCM) e *Hash Programming Environment*[4] (HPE), etc. Porém, dentre estas plataforma, a HPC-*Shelf* vem com o objetivo de integrar os conceitos já citados e fornecer uma ambiente computacional que permita que os especialistas da CAD construam suas aplicações, abstraindo os detalhes de implementação do *hardware*. Além de permitir que usuários finais (também chamados de usuários especialistas no contexto da HPC-*Shelf*) também possam usufruir destas aplicações em suas áreas de interesse, abstraindo os detalhes implementação do *software*.

A plataforma HPC-*Shelf* é um ambiente de nuvem computacional que possui tipos diferentes de usuários, também chamados de *atores*, onde cada ator possui um papel diferente, seja provendo recursos ou os consumindo. Por isso, este ambiente precisa de interfaces que sejam apropriadas para cada tipo de ator, levando em conta seu papel na plataforma, seja criando ou registrando novas aplicações, ou disponibilizando recursos computacionais (*clusters*, servidores, etc), ou apenas utilizando-os. Porém, a mesma não possui estas interfaces, o que caracteriza o principal problema tratado neste trabalho. Por isso, este trabalho tem como objetivo o desenvolvimento de uma ferramenta gráfica de acesso baseado na *web*, com o intuito de atender a necessidade da plataforma HPC-*Shelf*. Esta ferramenta deve prover funcionalidades que permita que (i) os provedores de recursos possam registrar seus recursos (aplicações e recursos computacionais) na plataforma, e (ii) os consumidores destes recursos possam utilizá-los de forma conveniente.

Diversos trabalhos já foram desenvolvidos visando tratar deste mesmo problema, porém, são aplicados a outras plataformas. Dentre estes trabalhos estão os de (SHEN *et al.*, 2001), que criou um ambiente de apoio ao desenvolvimento de aplicações distribuídas; (BRAMLEY *et al.*, 2000), (BAUDE *et al.*, 2001) e (BAUDE *et al.*, 2002), que criaram um ambiente para desenvolvimento de aplicações distribuídas baseadas em componentes; (HARRISON *et al.*, 2008) e (OINN *et al.*, 2004) criaram um ambiente para construção de aplicações distribuídas gerenciadas por *workflows*. Porém, nenhum destes utilizou os conceitos de *workflows* e POC de

[1] http://www.cca-forum.org
[2] http://fractal.ow2.org
[3] http://gridcomp.ercim.eu
[4] $http://www.scidac.gov/March2004/ascr_isic_3.html$

forma integrada.

Este trabalho está organizado da seguinte forma: a Seção 2 contém cinco trabalhos relacionados. Serão discutidas na seção as propostas dos trabalhos e como eles se assemelham e se diferenciam da proposta aqui apresentada. A Seção 3 apresenta o objetivo geral e os objetivos específicos do trabalho. A Seção 4 apresenta com mais detalhes os conceitos técnicos e teóricos necessários para a realização do trabalho. A Seção 5 mostra a descrição do procedimento metodológico que será realizado no trabalho. A Seção 6 mostra os resultados obtidos com implementação da ferramenta, e por fim as considerações finais e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, apresentaremos o que constitui como principal pano de fundo deste trabalho, bem como os principais conceitos utilizados. Nas seções subsequentes, iremos definir individualmente cada conceito. Na Seção 2.4 mostraremos como estes conceitos são integrados na plataforma, além disso, nesta mesma seção também será mostrado como a plataforma HPC-*Shelf* está estruturada, e como a estrutura se relaciona com seus atores.

2.1 Nuvens Computacionais

Desde os anos de 1980, o mundo corporativo e a comunidade acadêmica tem se deparado com um aumento expressivo da carga de trabalho gerada pelos problemas computacionais de seus domínios, esses problemas têm sido explorados pela computação com o intuito de resolvê-los da forma mais eficiente possível. Para atingir tal objetivo, diversos profissionais da computação têm criado diversas técnicas e ferramentas para auxiliar o desenvolvimento de soluções mais eficientes, dentre essas técnicas surgiu a computação paralela, que busca extrair maior aproveitamento das plataformas de computação por meio da execução simultânea de diversas instruções (STALLINGS, 2010).

O computador CDC6600, desenvolvido pela *Control Data Corporation* (CDC), foi o primeiro com a capacidade de computação paralela. Além dele, o Cray-1, desenvolvido pela empresa *Cray Research*, também foi considerado um marco inicial para supercomputação, também inaugurando o uso efetivo da computação vetorial na supercomputação. Na década de 1980, surgiram as arquiteturas MPPs (*Massive Parallel Processors*), que permitiram a criação de supercomputadores com memória distribuída e processadores de propósito geral, interligados por meio de interconexão de alta velocidade, que no fim da mesma década já dominavam o mercado de supercomputação. No mesmo período surgiram os *clusters*, estes formados por processadores e interconexões de prateleiras (STALLINGS, 2010), que desde o final dos anos 1990 vêm liderando o ranque de arquiteturas de computação paralela. Essas arquiteturas serviram de base para as grades computacionais, que também utilizam o mesmo princípio de divisão da carga de trabalho entre as diversas máquinas (STALLINGS, 2010). Posteriormente os aceleradores computacionais como GPUs (*Graphic Processor Units*) e MICs (*Many Integrated Core*), que constituem-se como unidades de processamento gráfico, vêm sendo amplamente utilizados para computação numérica (AL-ALAM, 2013). No entanto, um outro conceito vem ganhando espaço

na computação moderna, amplamente conhecido como *computação em nuvem*.

Atualmente, usufruímos de certos recursos, como energia elétrica, oferecidos por meio de serviços prestados pelas empresas ou órgãos especializados de forma abstrata, ou seja, toda a complexidade envolvida no processo de geração e distribuição destes recursos é abstraída[1], permitindo que nós consumidores possamos utilizar esses recursos de forma transparente. Este mesmo conceito foi estendido para as tecnologias de informação, e emergiu rapidamente ao longo dos últimos anos. Segundo o Instituto americano de Padrões e Tecnologia - *National Institute of Standards and Technology* (NIST) - a computação em nuvem (*Cloud Computing*) pode ser definidas da seguinte forma:

> Computação em nuvem é um modo que permite de forma ubíqua, e conveniente, o acesso sob demanda a um conjunto de recursos configuráveis de computação compartilhada (por exemplo, redes, servidores, armazenamento, aplicativos e serviços) que podem ser rapidamente provisionados e liberados com o mínimo esforço de gerenciamento ou interação do provedor de serviços (MELL *et al.*, 2011).

A princípio, computação em nuvem foi cunhado como um termo abrangente para descrever uma categoria de sofisticados serviços de computação sob demanda, inicialmente oferecidos por provedores comerciais, como Amazon[2], Google[3] e Microsoft[4] (BUYYA *et al.*, 2010), mas nos últimos anos, outras empresas também têm adotado este modelo.

Segundo Mell e Grance (2011), as nuvens computacionais possuem certas características essenciais, e a primeira delas é o *self-service* sob demanda. Esta característica nos diz que um consumidor pode fornecer ou consumir recursos conforme sua necessidade, sem requerer a interação humana com o provedor destes recursos.

As nuvens computacionais também permitem que seus recursos sejam acessados pela rede, principalmente pela *internet*, possibilitando que uma grande variedade de dispositivos (por exemplo, telefones celulares, *tablets*, *laptops* e estações de trabalho) possam usufruir de sua capacidade.

Outra característica importante das nuvens computacionais é o agrupamento de recursos, pois estes recursos são reunidos para atender múltiplos consumidores e podem ser alocados e realocados dinamicamente conforme a necessidade do consumidor; para os consumidores,

[1] Neste contexto, o termo *abstrair* refere-se a separação daquilo que é mais importante para o usuário, expondo apenas aquilo que é de interesse do mesmo, neste caso, a própria energia elétrica apta para o consumo doméstico.
[2] https://aws.amazon.com/pt/
[3] https://cloud.google.com/?hl=pt-br
[4] https://azure.microsoft.com/pt-br/

geralmente os detalhes da localização geográfica destes recursos computacionais são escondidos, permitindo que os usuários utilizem esses recursos de forma transparente[5] (MELL *et al.*, 2011).

Em nuvens computacionais, os recursos podem ser concedidos de forma elástica, ajustando-se proporcionalmente a demanda do consumidor, dando a impressão de que os recursos são ilimitados e apropriados às necessidades do consumidor; vale ressaltar que, plataformas que implementam o conceito de nuvem possuem mecanismos de medição de serviços, logo, a provisão elástica de recursos é controlada e monitorada pela mesma, proporcionando transparência para consumidor dos serviços (MELL *et al.*, 2011).

Tecnologias que implementam o conceito de nuvem tem crescido rapidamente e ocupado o espaço antes dominado por *clusters*. Esta ocupação é justificada pelas atraentes classes de serviços oferecidos, dividas em três categorias: *Software as a Service* (SaaS), *Platform as a Service* (PaaS) e *Infrastructure as a Service* (IaaS). SaaS defini-se como serviços que estão no topo da pilha de uma nuvem, estes são oferecidos aos usuários comuns[6] e podem ser acessados por meio de *browsers*. PaaS constitui-se como ambientes de desenvolvimento e implantação que permitem aos usuários criar e executar suas aplicações com pouca ou nenhuma preocupação com detalhes de baixo nível da plataforma. Por fim, IaaS são serviços que oferecem plataformas de computação, como servidores e *clusters*, executando diversas opções de sistemas operacionais (BUYYA *et al.*, 2010).

Além dos serviços citados anteriormente, as nuvens computacionais têm proporcionado o surgimento de novos serviços, como: *Storage as a Service* (StaaS), que fornece um serviço de armazenamento sob demanda; *Communication as a Service* (CaaS), que provê recursos para serviços que usam mensagens instantâneas e aplicações de videoconferência (KUNDU *et al.*, 2010); *Development as a Service* (DaaS), que concede ferramentas de desenvolvimento principalmente baseadas na web; dentre muitos outros serviços.

Podemos classificar as nuvens computacionais em 4 classes de modelos: *privada*, *comunitária*, *pública* e *híbrida*. Nuvens privadas, são tipos de nuvens desenvolvidas para uso exclusivo de uma organização, esta organização pode disponibilizar os seus recursos para cliente e parceiros, porém, o uso é restrito e controlado pela organização detentora da plataforma (MELL *et al.*, 2011). Nuvens comunitárias estendem o conceito de nuvem privada, pois não restringe apenas à uma organização, mas a uma comunidade de consumidores ou até mesmo de

[5] O usuário enxerga apenas aquilo que é importante para ele, e todos os detalhes de criação e gerenciamento destes recursos estão ocultos.

[6] Usuários que não necessariamente têm conhecimento em computação, e estão apenas interessados em usar aplicações como editores de texto, editores de imagens, etc.

organizações com interesses em comum (MELL *et al.*, 2011). Nuvens públicas são uma classe de nuvens que não restringem o seu uso, pois o mesmo é aberto ao público geral, podendo ser gerida e operada por uma organização ou instituição (MELL *et al.*, 2011). Além das classes já citadas, existem nuvens que integram características de outras classes, este modelo é chamado de híbrido, pois a mesma pode particionar sua infra-estrutura e adotar diferentes modelos (privada, comunitária ou pública) para cada partição (MELL *et al.*, 2011).

Apesar das vantagens, as nuvens computacionais possuem limitações. A primeira delas é a própria rede onde os dados trafegam, considerando que plataformas em nuvem são altamente dependentes da *internet*, esta dependência pode resultar em um grande gargalo para aplicações que demandam de grandes quantidades de dados e não dispõem de uma boa conexão com a *internet* (ABADI, 2009). Além das limitações geradas pela *internet*, as nuvens podem oferecer recursos distribuídos geograficamente, possivelmente em locais muito distantes, podendo prejudicar o desempenho da nuvem.

No contexto da CAD, a computação em nuvem tem trazido benefícios, principalmente pelas classes de serviços, pois os usuários de CAD podem usufruir de recursos computacionais sem a necessidade de mantê-los localmente (AHUJA; MANI, 2012). Porém, a sobrecarga inserida pela virtualização de diversos sistemas operacionais e a transparência em relação a localização geográficas das plataformas são características que motivaram o afastamento dos usuários de CAD das nuvens computacionais (AHUJA; MANI, 2012).

Como será descrito na Seção 2.4, este trabalho tem como plano de fundo uma plataforma de computação em nuvem, logo a mesma fornece níveis de abstrações diferentes para cada tipo de usuário. Assim, o conceito de nuvem é relevante para este trabalho pelo fato do mesmo ter como objetivo fornecer interfaces gráficas adequadas a cada tipo de usuário conforme seu interesse na plataformas.

2.2 Programação Orientada a Componentes

A indústria, de uma forma geral, tem tido grande preocupação com o custo, a qualidade e a velocidade na produção de seus produtos, e na indústria de *software* isso não é diferente. Por isso, nos últimos anos, têm surgido diversos tipos de processos e paradigmas que buscam facilitar o processo de produção; diminuindo o custo e aumentando a produtividade, e dentre eles estão: a programação orientada a procedimentos, também conhecida como programação estruturada, que incentiva os programadores a pensar em termos de procedimento e funções; a

programação orientada a objetos (POO), que estimula os desenvolvedores a pensar em termos de tipos de dados e hierarquias de tipos de dados; e a programação orientada a aspectos (POA), que introduz uma nova tecnologia de separação de preocupações no desenvolvimento de *software* (WANG; QIAN, 2005); temos também a arquitetura orientada a serviço (AOS), que prega que as funcionalidades devem ser implementadas como serviços que se comunicam por meio de interfaces acessadas remotamente (KRAFZIG *et al.*, 2005). Além dos paradigmas citados, outro que se destaca é a programação orientada a componentes (POC).

Inicialmente, a abordagem orientada a componentes obteve reconhecimento na indústria de *hardware*, pois permitiu que novos produtos fossem criados a partir da montagem de componentes pré-fabricados por empresas especializadas na elaboração destes componentes, proporcionando-lhes um ganho de produtividade (WANG; QIAN, 2005).

Na indústria de *software*, a POC é uma abordagem que permite o desenvolvimento de aplicações a partir da composição de componentes de *software* reutilizáveis e auto-contidos, estes componentes de *software* seguem um padrão predefinido que permite a interação com outros componentes (WANG; QIAN, 2005). Estes componentes também podem ser definidos da seguinte forma:

> Um componente de *software* é um bloco de construção de *software* que está em conformidade com um modelo de componente. Um Modelo de Componente define padrões para propriedades que componentes individuais devem satisfazer e métodos, e possivelmente mecanismos, para compor componentes. (CRNKOVIC *et al.*, 2007)

De acordo com Crnkovic et al. (2011), um componente está diretamente relacionado a seu modelo, por isso, existe uma grande diversidade de modelos de componentes no mercado, com as mais diversas características e comportamentos. Mas além de seus modelos, componentes de *software* podem ser definidos como unidades que possuem funcionalidades específicas. Estas funcionalidades são servidas a usuários ou a outros componentes por meio de interfaces, a partir destas interfaces o componente interage com o seu contexto, onde o mesmo está implantado; além disso, os componentes *software* são auto-contidos e auto-implantáveis, ou seja, podem ser executados e testados independentemente de outras partes do sistema, facilitando a sua utilização ou até mesmo sua substituição (WANG; QIAN, 2005).

Em um contexto geral, a POC oferece uma série de benefícios, como flexibilidade, adaptabilidade, manutenibilidade, reusabilidade, interoperabilidade e escalabilidade (WANG; QIAN, 2005). No desenvolvimento de *software*, mudanças de requisitos e modificação ou

inclusão de novas funcionalidades são muito frequentes, por isso os sistemas devem ser construídos visando essas mudanças. Tendo em vista estas necessidades, a POC fornece agilidade no processo de adição e remoção de funcionalidades, evitando longas interrupções no processo de desenvolvimento (WANG; QIAN, 2005).

No que diz respeito à adaptabilidade, a POC permite que seus componentes sejam modificados para torná-los compatíveis com as necessidades do usuário, causando um impacto mínimo no núcleo da aplicação (WANG; QIAN, 2005), e como veremos nos próximos parágrafos, esta característica influencia as demais.

Considerando a natureza auto-contida dos componentes, os mesmos podem ser facilmente substituídos por outros componentes ou versões melhores, esta facilidade permite que a aplicação seja manutenível, pois novas implementações de funcionalidades podem ser incorporadas ao sistema sem longas interrupções (WANG; QIAN, 2005).

A interoperabilidade é resultante da possibilidade de integrar diversas aplicações de forma padronizada, para isso, as interfaces disponibilizadas pelos componentes têm papel importante, porque por mais distintas que sejam as aplicações, a comunicação entre os componentes e as aplicações se torna padronizada, e consequentemente, há um ganho de escalabilidade, pois a integração de aplicações que estão em diferentes plataformas computacionais permite que o sistema continue operando com diferentes demandas (AL-ALAM, 2010).

Em particular na característica de reuso a POC tem se destacado muito, pois esta característica permite que uma funcionalidade ou aplicação seja projetada e implementada apenas uma vez, e usada diversas vezes, promovendo um ganho substancial de produtividade (WANG; QIAN, 2005). Existem diversos níveis de reuso, e para cada nível é fornecido uma camada de abstração ao usuário; no nível mais baixo de reuso residem as bibliotecas baseadas em programação procedural onde os programadores podem implementar procedimentos e usá-los diversas vezes em diferentes locais do código do programa, porém não são extensíveis. Bibliotecas baseadas em orientação a objetos fornecem um nível mais alto de abstração em relação a bibliotecas implementadas sobre o paradigma procedural, porém, ela exige que o usuário compreenda a estrutura e o comportamento interno desta biblioteca, pois qualquer mudança nas classes de base podem afetar diretamente o comportamento da biblioteca, além de estar fortemente acoplada ao tipo de linguagem usada pela própria biblioteca, dificultando a interação com outras linguagens. Em contraste a outros paradigmas, a POC permite que o componente possa ser modificado internamente, desde que as interfaces permaneçam inalteradas,

ou até mesmo usá-lo descartando os detalhes de implementação, utilizando apenas as próprias interfaces providas pelo componente, fornecendo um alto nível de abstração ao usuário (WANG; QIAN, 2005).

No contexto de computação de alto desempenho, existem diversos modelos de componentes que foram criados especificamente para atender aos requisitos das aplicações da CAD, alguns deles são: o *Common Component Architecture* (CCA), Fractal, *Grid Component Model* (GCM) e *Hash*; além destes, existem outros modelos que são implementados como serviços remotos, disponibilizados por meio da *internet* na forma de *web services* (SILVA, 2016).

Para este trabalho será utilizado o modelo de componentes *Hash*, pois este é o modelo utilizado na plataforma HPC-*Shelf*, que será descrito na Seção 2.4. Além disso, parte deste trabalho destina-se a representação gráfica deste modelo.

2.3 *Workflows* Científicos

No mundo globalizado, as organizações estão cada vez mais competitivas, exigindo extrema eficiência de seus processos internos. Devido a estas exigências, constatou-se que era necessário uma maior integração entre os processos e a automatização das atividades operacionais, e a partir destas necessidades surgiu o conceito de *workflows*.

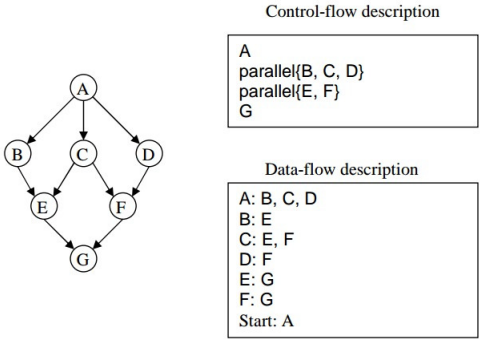
Segundo a definição oficial dada pela *Workflow Management Coalition* (WfMC) - organização global de profissionais e pesquisadores envolvidos em *workflows*, *Business Process Management* (BPM) e *Adaptive Case Management* - *workflow* é definido como:

> O *workflow* está relacionado com a automatização de procedimentos em que documentos, informações ou tarefas são passados entre os participantes de acordo com um conjunto definido de regras para alcançar ou contribuir para um objetivo comercial global. (HOLLINGSWORTH; HAMPSHIRE, 1995)

A definição dada pela WfMC foi concebida para o contexto empresarial, porém, o conceito foi explorado pela comunidade científica, e levou a uma ligeira adaptação. No contexto científico, *workflows* são uma sequência de etapas ou atividades que descrevem o processo de um experimento computacional (BARKER; HEMERT, 2007). Estes experimentos computacionais podem ser elaborados através da composição de componentes de *software* e sistemas distribuídos através da *internet*. Dentro desse contexto, *workflows* podem ser usados para orquestrar componentes e sistemas a fim de integrar os recursos para resolução do experimento (SILVA, 2016).

Workflows podem ser definidos basicamente por dois estilos, estilo de controle de fluxo e o estilo de fluxo de dados. No estilo de controle de fluxo, a ordem de execução de tarefas do *workflow* é explicitamente especificada, podendo ser declarados através de diretivas e comandos semelhantes aos da programação como: "*if*", "*while*", "*fork*", "*join*", "*parallel*" (YAN, 2007). No estilo de fluxo de dados, um *workflow* consiste em um conjunto de tarefas e um conjunto de relações de dependência, estas relações de dependência determinam a ordem de execução das tarefas do *workflow* (YAN, 2007). Como exemplo para ambos os estilos, a Figura 1 nos mostra como um *workflow* é especificado: no estilo de controle de fluxo, a primeira a ser executada é tarefa A, seguida das tarefas B, C e D que serão executadas em paralelo, após a execução das três tarefas anteriores segue-se E e F para serem também executadas em paralelo e por fim a tarefa G; no estilo de fluxo de dados, apenas é necessário especificar as relações de dependência (as tarefas B, C e D são dependentes da tarefa A, e assim por diante). Para grandes *workflows*, o estilo de controle de fluxo pode tornar-se complexo e propenso a erros por parte do desenvolvedor, em compensação, a ordem de execução das tarefas podem ser facilmente determinada pelo desenvolvedor por meio do estilo de fluxo de dados (YAN, 2007).

Figura 1 – Exemplo de especificação de *workflows*



Fonte: Yan (2007)

Devido a abertura de um leque de possibilidades, proporcionado pelo uso de *workflows* no contexto científico, surgiram uma grande variedade de tecnologias e *frameworks* para gerenciamento de *workflows*, que permitiram os engenheiros e cientistas usufruírem de seus benefícios de forma transparente e em alto nível. São exemplos: o *Taverna*, que consiste em um sistema de gerenciamento de *workflows* de código aberto, voltado para computação de alto desempenho, este provê um conjunto de *middlewares* com baixo acoplamento e semanticamente apto para apoiar os experimentos de cientistas (BARKER; HEMERT, 2007); o *Kepler*, é um

framework de código aberto com contribuintes de uma variedade de projetos de pesquisa, construído sobre o Ptolomeu II, que consiste em sistema criado na Universidade da Califórnia em *Berkeley*, que adota o estilo de fluxo de dados para definir seus *workflows* (BARKER; HEMERT, 2007); o *Triana*, é um ambiente projetado para definição, processamento, análise, gerenciamento, execução e monitoramento de *workflows*, este também fornece ferramentas gráficas para automatizar estas atividades; dentre outros *frameworks* (BARKER; HEMERT, 2007). Apesar de distintos, todos estes *frameworks* possuem um objetivo em comum, que consiste na busca de uma forma de gerenciar aspectos comuns aos *workflows*; estes aspectos compreendem as fases de *projeto*, *instanciação*, *execução* e *análise* desses *workflows*.

Apesar da diversidade de *frameworks* para lidar com as estruturas de *workflows*, estas ferramentas normalmente apresentam limitações. Estas limitações ocorrem principalmente na reutilização de *workflows* previamente definidos, pois cada modificação é considerado um *workflow* diferente, o que acaba dificultando a escalabilidade e a manutenibilidade (MATTOSO *et al.*, 2009).

Neste trabalho, será necessário que fluxos de execução entre componentes sejam representados graficamente, adotando o estilo de controle de fluxo utilizado na plataforma que descreveremos na Seção 2.4.

2.4 HPC-*Shelf*

A HPC-*Shelf* é uma plataforma em nuvem para computação de alto desempenho baseada em componentes, que provê um ambiente de desenvolvimento e execução de aplicações computacionalmente intensivas, e procura promover a aproximação dos usuário de CAD as nuvens computacionais (SILVA, 2016). Essa plataforma foi proposta pelo grupo de Computação de Alto Desempenho do programa de Mestrado e Doutorado em Ciência da Computação (MDCC) da Universidade Federal do Ceará e constitui o plano de fundo deste trabalho.

A plataforma HPC-*Shelf* usa como base o modelo *Hash* de componentes, este modelo descreve as características de componentes que possuem propriedades específicas da programação paralela. Estes componentes serão utilizados para compor as aplicações, sendo orquestrados por *workflows* a fim de executar um determinado conjunto de operações contidas nos próprios componentes.

A plataforma HPC-*Shelf* possui tipos específicos de usuários, esses usuários também chamados de *atores*, atuam sobre um conjunto de elementos arquiteturais que compõem a

plataforma, estes elementos são: *Front-End*, *Core* e *Back-End*. Os atores e os elementos arquiteturais serão descritos com mais detalhes nas Seções 2.4.1 e 2.4.2 respectivamente.

2.4.1 Atores da HPC-Shelf

Os atores da HPC-*Shelf* são os usuários que provêem e consomem recursos da nuvem, estes atores estão divididos nas seguintes classes: *usuários especialistas*, *montadores de aplicações*, *desenvolvedores de componentes* e *mantenedores de plataformas*. Cada classe de atores possui uma visão diferente dos serviços da nuvem, impactando em suas responsabilidades em relação a mesma.

O usuário especialista, ou *especialista*, é o tipo de usuário que está interessado apenas em consumir os recursos, pois ele apenas utiliza as aplicações que atendem às suas demandas. O montador de aplicações, ou *montador*, é o tipo de usuário que provê recursos na forma de aplicações, estas aplicações são construídas por meio da composição de componentes registrados na plataforma HPC-*Shelf*. Os componentes registrados na plataforma são desenvolvidos por um desenvolvedor de componentes, ou *desenvolvedor*, pois ele possui conhecimento em métodos e técnicas computacionais para construir funcionalidades que podem usufruir eficientemente as plataformas de execução. Para gerenciar estas plataformas de execução, existe o mantenedor de plataforma, ou apenas *mantenedor*, pois ele é responsável pela infraestrutura da nuvem.

Os especialistas, são atores que dominam alguma área do conhecimento (química, física, biologia, geologia, etc), não necessariamente computação, estes são tidos como usuários finais da plataforma, pois são os principais consumidores dos serviços disponíveis na HPC-*Shelf*, isso implica dizer que, os especialistas vêem a plataforma na visão de *SaaS* (SILVA, 2016). Pelo fato do especialista enxergar a nuvem como um conjunto de aplicações, a plataforma deve prover um ambiente que descreve os problemas de seu interesse, contendo as abstrações e linguagens específicas de seu domínio, e da mesma forma, lhe dê a capacidade de inserir os dados a serem computados, e por fim lhe fornecer os resultados destas computações em tempo hábil, permitindo que estes possam ser visualizados na própria interface ou transferidos para o especialista na forma de arquivos.

O montador é um ator que possui conhecimento em métodos computacionais para resolução de problemas de domínios específicos das ciências e engenharias. Ele é responsável por construir as aplicações utilizadas pelos especialistas, logo, o montador vê a plataforma como *PaaS* (SILVA, 2016). Para atuar na plataforma, o montador precisa ter a capacidade de

listar, pesquisar e reutilizar os componentes disponíveis em suas aplicações. Os montadores de aplicação também são responsáveis por traduzir os problemas descritos pelos especialistas para a linguagem de *workflows*, e através desta linguagem os componentes serão orquestrados para resolver o problema. Naturalmente, é responsabilidade do montador, definir as funcionalidades e controles para cada aplicação, bem como registrá-las e atualizá-las no repositório de aplicações da HPC-*Shelf*.

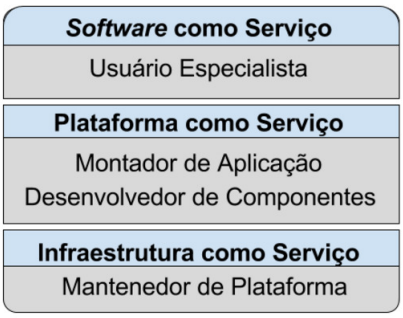
Da mesma forma que o montador, o desenvolvedor também possui conhecimento em métodos computacionais e arquiteturas de computação, pois ele deve prover recursos para nuvem na forma de componentes (SILVA, 2016). Por isso, ele também deve possuir um ambiente que possibilite desenvolver seus componentes, registrá-los e atualizá-los no catálogo de componentes. Outra responsabilidade inerente ao desenvolvedor é a definição do modelo de desempenho, neste modelo é determinado o comportamento do componente em função do tempo e do espaço de memória; além do modelo de desempenho, é necessário especificar os requisitos não-funcionais, como por exemplo, a necessidade de aceleradores computacionais; é de responsabilidade de desenvolvedor a especificação de parâmetros a respeito da plataforma de computação paralela, e as dependências necessárias à execução do componente, como: bibliotecas, compiladores e etc.

Por fim, o mantenedor é o ator responsável por criar componentes que representam plataformas de computação paralela, onde estes componentes correspondem a espécie plataforma; e por terem a responsabilidade de manter as plataformas físicas da HPC-*Shelf*, estes usuários possuem a visão de *IaaS* (SILVA, 2016). Contratos contextuais podem ser definidos como instâncias de componentes criados por desenvolvedores, para os mantenedores os contratos contextuais de componentes da espécie plataforma são chamados de perfis de plataformas virtuais, representando um procedimento de instanciação de uma plataforma virtual, que atende pelo perfil associado a plataforma física do mantenedor. Cada mantenedor gerencia um parque de recursos computacionais, dessa forma, apenas ele pode cadastrar no catálogo de componentes os perfis de suas plataformas.

A Figura 2 mostra a relação entre os atores da plataforma HPC-*Shelf* e suas respectivas visões dos serviços da nuvem.

Neste trabalho daremos foco a visão de plataforma como serviço (*Platform as a Service*), que engloba o papel dos montadores de aplicação e dos desenvolvedores de componente.

Figura 2 – Atores e suas respectivas visões dos serviços da nuvem



Fonte: Próprio autor

2.4.2 Elementos Arquiteturais

A HPC-*Shelf* é composta por três elementos arquiteturais: *Front-End*, *Core* e *Back-End*, estes elementos relacionam-se uns com os outros, a fim cumprir com o objetivo global da HPC-*Shelf*. As seções a seguir, descrevem em detalhes as características e responsabilidades de cada elemento arquitetural.

2.4.2.1 Front-End

Front-End é o nome dado aos conjunto aplicações que cada ator utiliza para interagir com a plataforma, estas aplicações variam conforme os interesses e responsabilidades de cada ator, ou seja, cada classe de atores possui uma forma específica de interagir com a nuvem (AL-ALAM, 2013). Neste elemento arquitetural, existem quatro tipos de conjunto de aplicações que fornecem funcionalidades específicas para cada ator, sendo elas:

- as aplicações;
- o ambiente de montagem de aplicações;
- o ambiente de desenvolvimento dos componentes;
- e o ambiente de configuração de plataformas.

Aplicações

As aplicações representam a interface que os especialistas utilizam para interagir com a plataforma. O especialista escolhe a aplicação que descreve seu problema de interesse, e por meio dela que o usuário irá obter seus resultados (AL-ALAM, 2013).

O especialista enxerga a aplicação através de uma linguagem de domínio específico, esta linguagem pode ser textual ou gráfica, a mesma deve ser acessível por um navegador. Compondo a aplicação existe o *framework*, chamado *Shelf Application Framework* (SAFe), que diz respeito a visão do montador; o SAFe permite que a aplicação tenha acesso aos componentes da nuvem, de forma que os mesmos possam ser orquestrados para resolver o problema de interesse do especialista. A descrição do problema do usuário deve ser traduzida para uma linguagem de *workflows* suportada pelo *framework*, pela qual ele irá executá-lo (AL-ALAM, 2013).

Nesse contexto, a linguagem de domínio específico funciona como uma espécie de linguagem de programação, pois através dela o usuário descreve seus problemas e a aplicação traduzirá para uma linguagem compreensível à plataforma, a fim de construir uma solução computacional, agindo como uma espécie de "compilador" para o problema descrito pelo especialista.

Ambiente de Montagem de aplicações

O ambiente de montagem de aplicação é uma interface criada para os montadores interagirem com a plataforma HPC-*Shelf*, por meio dela os montadores constroem aplicações a partir dos componentes catalogados na nuvem, assim eles podem propor soluções para os problemas descritos pelos especialistas, tendo também a capacidade de desenvolver componentes próprios a partir dos componentes existentes.

Através deste ambiente os montadores poderão construir a arquitetura das aplicações, além da definição do fluxo de execução das aplicações, esta definição será feita por meio de *workflows* que orquestrarão o comportamento dos componentes que constituem as aplicações. Além do comportamento, o montador também poderá definir interfaces amigáveis que irão conter abstrações adequadas ao domínio dos especialistas.

Naturalmente fica a cargo dos montadores, o registro, a remoção e atualização das aplicações, estes serviços são análogos aos que são prestados aos desenvolvedores (AL-ALAM, 2013).

Ambiente de Desenvolvimento de Componentes

O ambiente de Desenvolvimento de componentes funciona como uma espécie de

IDE (*integrated development environment*), dando suporte ao desenvolvimento de componentes e especificação de suas características. Estes componentes serão dispostos no catálogo da nuvem para serem utilizados pelas aplicações. Neste ambiente é oferecido para o desenvolvedor uma interface visual para criação dos componentes, bem como a definição de seus contratos que representam instâncias destes componentes (AL-ALAM, 2013).

Ambiente de Configuração de Plataformas

O ambiente de configuração de plataformas é uma interface criada para que os mantenedores possam interagir com a nuvem, de forma semelhante aos desenvolvedores, mas diferenciados pelo fato de criarem componentes de tipos específicos, estes são responsáveis por configurar a infraestrutura da nuvem, que também são representados por componentes (AL-ALAM, 2013).

O ambiente de configuração de plataformas possibilita ao mantenedor, definir os perfis das plataformas virtuais contidas em seu conjunto de plataformas, intitulado como parque de recursos, essas definições irão auxiliar o processo de instanciação de plataformas. O mesmo conjunto de funcionalidades, a saber, o registro e atualização de componentes, disponíveis aos montadores e desenvolvedores de componentes são disponibilizados aos mantenedores.

2.4.2.2 Core

O Core é o elemento arquitetural que possui as seguintes responsabilidades: prover a manutenção do catálogo de componentes, controlar o desenvolvimento e a configuração de plataformas; implementar a estratégia de resolução dos contratos dos componentes e por fim instanciar os componentes de diferentes espécies. Também é responsabilidade do Core, manter os serviços de catalogação de componentes servidos aos atores (com exceção do especialista), esses serviços são: registro, remoção e atualização de versões dos componentes e contratos (AL-ALAM, 2013). O Core também se comunica com o Back-End, agindo apenas como consumidor de seus recursos, permitindo a instanciação e monitoramento das plataformas, e a implantação de componentes.

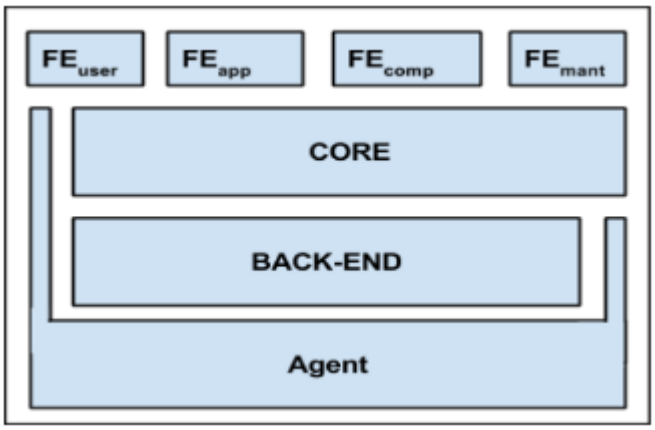
2.4.2.3 Back-End

O Back-End é o elemento que tem como responsabilidade mapear os perfis de plataforma criados pelos mantenedores para as plataformas de computação paralela reais; também é cargo deste elemento, fornecer serviços de monitoramento das plataformas de computação, tanto para o Core como para a aplicação. O perfil de plataforma já possui as informações necessárias para que o Back-End faça a instanciação da mesma, assim, ele irá alocar recursos do parque de recursos do mantenedor (AL-ALAM, 2013). Caso não seja possível instanciar o perfil, o Back-End informa ao Core que não foi possível criar a plataforma virtual, gerando uma exceção. A partir daí, fica a cargo do Core escolher outro componente ou até mesmo abortar a execução da aplicação.

Integrado ao Back-End, existe um elemento chamada Agente de plataforma, que é um serviço responsável pela comunicação entre uma plataforma virtual e o Core, que a invoca, para os propósitos de instanciação de componentes e monitoração do estado de execução (AL-ALAM, 2013). Um Agente é criado em tempo de instanciação da plataforma e possui o mesmo tempo de vida da plataforma virtual que o hospeda, assim, permitindo o reaproveitamento da plataforma virtual por componentes de diferentes aplicações.

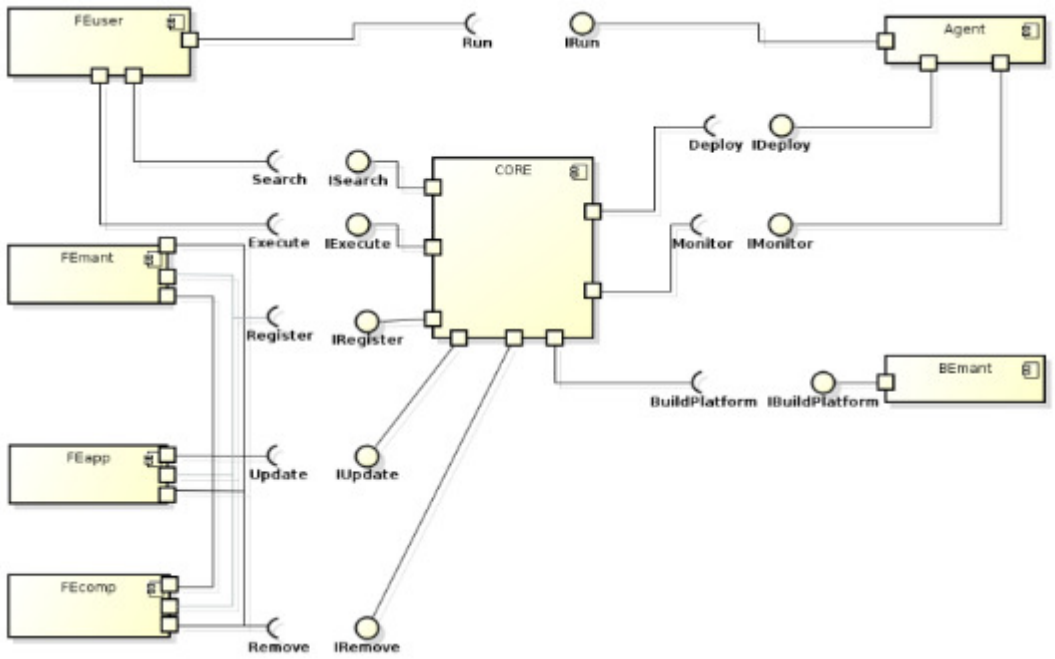
A Figura 3 mostra a relação entres os componentes arquiteturais, nela é possível visualizar os tipos de aplicações fornecidas pelo Front-End, sendo intermediadas pelo Core, que por sua vez se comunica com o Back-End e o Agente. A Figura 4 ilustra a comunicação entre os elementos arquiteturais, mostrando os serviços disponíveis em cada elemento e quem utiliza estes serviços, além da interação com o ambiente de cada ator da plataforma.

Figura 3 – Arquitetura HPC-*Shelf*



Fonte: Al-Alam (2013)

Figura 4 – Diagrama de Componentes da Arquitetura HPC-*Shelf*



Fonte: Al-Alam (2013)

2.5 Interfaces de Usuário

Desde a década de 1970, os usuários de computadores tiveram contato com uma evolução vertiginosa de seus ambientes de interação com plataformas de computação. Com os avanços dos sistemas operacionais, diversos modelos de interfaces foram propostos, e alguns deles tiveram grande aceitação pelos os usuários, permitindo que eles pudessem se desenvolver a ponto de chegar ao estado que testemunhamos hoje. Mas com o advento da *internet*, muitos dos conceitos utilizados nestas interfaces de usuário convencionais foram reaproveitados em páginas da *web*, levando a um novo universo de possibilidades.

Inicialmente, as primeiras páginas eram baseadas apenas em hipertextos, com links que redirecionavam o usuário para outras páginas, assim, o esforço para construir interfaces *web* tinha como foco a estrutura das informações, para permitir que os usuários navegassem entre as informações de forma rápida e fácil (ROGERS *et al.*, 2013). Mas desde a década de 1990, o esforço de desenvolvimento foi direcionado para estética destas interfaces, focando em ambientes agradáveis, usáveis, atrativos e de fácil manutenção. No início de 2000, muitas linguagens de programação e ferramentas de desenvolvimento foram criadas para dar apoio a criação destes ambientes gráficos, centrados na experiência do usuário. As novas linguagens,

técnicas e ferramentas que surgiram, permitiram que diversas aplicações, similares aos aplicativos usados em *desktops*, ganhassem força na *internet* (ROGERS *et al.*, 2013).

Desde então, podemos testemunhar a evolução destas linguagens e ferramentas, que ao longo dos anos vêm incorporando novos recursos, como HTML semântico e Ajax, que permitiu que houvesse mais interatividade nas interfaces de usuário, assim, as aplicações na *internet* vêm ganhando espaço entre os usuários, desde os mais leigos aos especialistas.

Neste trabalho as interfaces de usuário são de fundamental importância, porque por meio delas traduziremos os principais conceitos da plataforma HPC-*Shelf*, como componentes e arquiteturas, que até então são entidades conceituais importantes para os desenvolvedores e montadores, respectivamente.

2.5.1 Linguagens para Desenvolvimento

Como vimos na seção anterior, as aplicações na *web* tem ganhado força ao longo dos anos, e para apoiar este crescimento surgiram diversas linguagens e ferramentas que tratam das mais variadas questões inerentes a estas aplicações. Dentre as muitas ferramentas e linguagens existentes, algumas tornaram-se muito populares ao longo dos anos, como Java, Python, PHP e JavaScript (DIAKOPOULOS; CASS, 2016).

O Java é uma linguagem projetada para permitir a criação de programas que podem ser executados em diferentes plataformas com pouca ou nenhuma modificação, por isso ele ganhou muito espaço, tanto no âmbito da internet, quanto em aplicações *desktop* e *mobile* (DIAKOPOULOS; CASS, 2016). Além da linguagem em si, existem diversos complementos, também chamados de *frameworks*, que auxiliam no desenvolvimento das aplicações, fornecendo novos recursos e padrões que promovem o ganho de produtividade no processo de desenvolvimento, destes *frameworks* podemos citar: o JavaServer Pages (JSP), que permite o gerenciamento dinâmico de conteúdo em páginas *web*; o JavaServer Faces (JSF), que foca na interfaces de usuários e fornece componentes de interfaces para criar telas mais atraentes; já o JavaFX, que constitui-se como uma poderosa ferramenta para processamento gráfico e fornece controles para interfaces de usuário; e por fim o *Spring Framework*, que consiste em um *container*, que gerencia um conjunto de serviços para desenvolvimento *web*, com suporte a transações e a realização de armazenamento (GERASIMOV V.V.; PONOMAREV; LISHCHENKO, 2015); por fim, o framework Maven é uma ferramenta que tem auxiliado os desenvolvedores Java no processo de compilação dos projetos e no controle das dependências (MAVEN, 2017).

A linguagem Python é uma linguagem interpretada de alto nível e que suporta múltiplos paradigmas de programação, a mesma é de uso gratuito e de código-fonte aberto, e tem mostrado um grande potencial para construção de aplicações *web* (CRUZ, 2015). O Python também possui uma ampla gama de *frameworks* para desenvolvimento de aplicações *web*, como o Django[7] e Pyramid[8], ambos possuem recursos para gerenciamento de rotas, autenticação e administração de bases de dados, além de possuírem diversos *plugins* e extensões para os mais diversos propósitos; além destes, existe também o Flask, normalmente utilizado para projeto menores.

O PHP é uma linguagem de programação de *script server-side* (ou seja, é executada do lado do servidor) de código aberto, especialmente adequada para o desenvolvimento *web* e que pode ser embutida dentro do HTML (*HyperText Markup Language*) (PHP, 2017), similares a outras tecnologias como JSP. Assim como as demais tecnologias, o PHP dispõe de recursos complementares que auxiliam o processo de desenvolvimento, dentre estes recursos estão: o CodeIgniter[9], que é formado por um conjunto de bibliotecas para tarefas comuns combinado com interfaces simples e estruturas lógicas para acessar estas bibliotecas; Yii[10], é um *framework* de alto desempenho com arquitetura baseada em componentes, adequado para aplicações de larga escala como sistemas de comércio eletrônico e gerenciamento de conteúdo; por fim, o Zend[11] é um *framework* também baseado em componentes reutilizáveis e extensíveis, permitindo que os desenvolvedores usem qualquer componente com o mínimo de dependências (BAGESTAN, 2016).

Por fim, o JavaScript também é linguagem de programação interpretada de alto nível, dinâmica, dinâmicamente tipada que é bem adaptada aos paradigmas de programação orientada a objeto e funcional. Atualmente, grande parte das aplicações *web* e sites modernos são construídos com esta linguagem, pois toda a lógica destas aplicações é processada no próprio navegador, permitindo mais interatividade e dinamismo (FLANAGAN, 2011). Assim como as outras linguagens, o JavaScript também possui *frameworks* para otimizar o desenvolvimento, dentre estes *frameworks* estão: jQuery, que constitui-se como uma biblioteca que fornece um arcabouço de recursos que vão desde a interatividade com o usuário por meio de animações até requisições a recursos externos; MooTools, sendo esta uma biblioteca leve, modular e orientada

[7] https://www.djangoproject.com/
[8] https://trypyramid.com/
[9] https://www.codeigniter.com/
[10] http://www.yiiframework.com/
[11] http://www.zend.com/

a objetos, o que permite a construção de aplicações leves, pois poucos arquivos precisam ser carregados; YUI, criada por desenvolvedores da empresa Yahoo!, a mesma é composta por vários utilitários e componentes divididos em módulos que auxiliam o desenvolvimento de interfaces de usuário (BERNICHE; NERIS, 2013).

Dada a variedade de linguagens e *frameworks* disponíveis no mercado, para este trabalho foi escolhido as linguagens JavaScript e Java; além da experiência que o autor deste trabalho possui em relação a estas linguagens, cada uma delas foi escolhida para um propósito específico. O JavaScript foi escolhido por que o mesmo permite a construção de interfaces dinâmicas e interativas, além possuir diversas extensões que maximizam a produtividade e tornam a interface *web* atraente. A linguagem Java foi escolhida para permitir o gerenciamento da comunicação entre os *web services* da plataforma HPC-*Shelf* e as interfaces dos atores, e também fazer o controle das sessões dos usuários, além disso, parte da lógica da aplicação foi implementada em Java, para que o lado do cliente (ou seja, no navegador) não fique sobrecarregado e prejudique realização das atividades dos atores.

3 TRABALHOS RELACIONADOS

A Tabela 1 destaca as principais diferenças e semelhanças entre os trabalhos citados e o trabalho proposto. A seguir é descrito o que cada um destes trabalhos aborda.

Tabela 1 – Diferenças e semelhanças dos trabalhos relacionados com o trabalho proposto

Trabalhos	Nuvens Computacionais	POC	Workflows Científicos	Interfaces de Usuário	Tema
Baude et al. (2001)	**Não**	**Sim**	**Não**	**Sim**	*Ic2d: Interactive control and debugging of distribution*
Bramley et al. (2000)	**Não**	**Sim**	**Não**	**Sim**	*A component based services architecture for building distributed applications*
Harrison et al. (2008)	**Não**	**Sim**	**Sim**	**Sim**	*Ws-rf workflow in triana*
Oinn et al. (2004)	**Não**	**Não**	**Sim**	**Sim**	*Taverna: a tool for the composition and enactment of bioinformatics workflows*
Presente Trabalho	**Sim**	**Sim**	**Sim**	**Sim**	*Solução web para desenvolvimento na plataforma HPC-Shelf*

Fonte: Próprio autor

Baude et al. (2001a) apresenta um ambiente gráfico de monitoramento e controle remoto de aplicações distribuídas chamada *Interactive Control and Debugging of Distribution* (IC2D). O ambiente auxilia o controle e o monitoramento de aplicações que utilizam o modelo ProActive destinada a infraestruturas de *clusters* e *grid*, e (BAUDE *et al.*, 2002) o autor mostra um modelo de descrição de aplicações na linguagem XML para esta mesma ferramenta. Algumas dessas estratégias também serão adotadas neste trabalho, dentre elas a descrição das aplicações na linguagem XML, e a composição de componentes por meio de uma interface gráfica. Este trabalho difere-se do nosso pelo fato de adotar o modelo de componentes ProActive, além de destinar-se a plataformas de *cluster* e *grid*, pois o nosso trabalho visa para plataformas em nuvens computacionais.

Bramley et al. (2000) descreve uma abordagem para a construção de um sistema de componentes de *software* distribuído para aplicações científicas e de engenharia, baseados no modelo CCA e destinado a infraestrutura de *Grid*. Este trabalho apresenta um *framework*, nomeado de CCA *Toolkit* (CCAT), que permite a criação, registro e execução de aplicações baseadas em componentes por meio de uma interface de usuário, além de permitir que os recursos

do *framework* sejam utilizados através da linguagem Python. Este trabalho assemelha-se ao nosso pelo fato de ser proposto um ambiente gráfico que dá suporte a criação, e execução de aplicações orientadas a componentes distribuídos. Diferindo-se, porém, pelo modelo de componentes, pois o nosso não irá adotar o modelo CCA; e o ambiente de execução, pois este trabalho é destinado a plataformas de execução em nuvem.

Harrison et al. (2008) descreve a integração de funcionalidades de comunicação com *web services* em uma ferramenta de representação gráfica de *workflows*. Esta ferramenta permite a criação e gerenciamento de fluxos de trabalhos, por meio da composição de serviços representados por componentes. Parte deste assemelha-se com o nosso, pois, parte deste trabalho procura desenvolver um ambiente que dê suporte ao desenvolvimento de *workflows*, por meio de uma interface de usuário que facilite a composição destes fluxos de trabalho. Diferindo-se, pelo fato da ferramenta ter um foco exclusivo em controle e gerenciamento de *workflows* que integram serviços distribuídos.

Outro trabalho que ainda objetiva *workflows* é Oinn et al. (2004), que também descreve uma ferramenta para composição e implementação de *workflows* focados em bioinformática. A ferramenta também possui um aplicativo que fornece uma interface gráfica para construção de *workflows*, além de possuir uma linguagem específica para descrição dos fluxos de trabalhos. O objetivo deste trabalho assemelha-se com nosso, pois, o objetivo é fornecer uma ferramenta que auxilie a construção de *workflows* em alto nível de abstração, o que motiva a criação de interfaces de usuário e a linguagem específica para criação de *workflows*. Este trabalho se difere com o nosso, pelo fato de ser uma ferramenta focada no desenvolvimento de *workflows* contidos no domínio das aplicações da bioinformática.

4 OBJETIVOS

Este trabalho tem como objetivo conceber e implementar uma ferramenta *web* que automatiza o processo de desenvolvimento e montagem de aplicações na plataforma de computação em nuvem HPC-*Shelf*.

4.1 Objetivos específicos

Para alcançar o objetivo geral, elencamos os seguintes objetivos específicos:

- Levantamento e análise dos requisitos de cada tipo de usuário da plataforma;
- Modelar interfaces gráficas que atendam as necessidades identificadas;
- Planejamento e execução de testes de acordo com os requisitos especificados;
- Realização da integração da solução a plataforma.

5 PROCEDIMENTOS METODOLÓGICOS

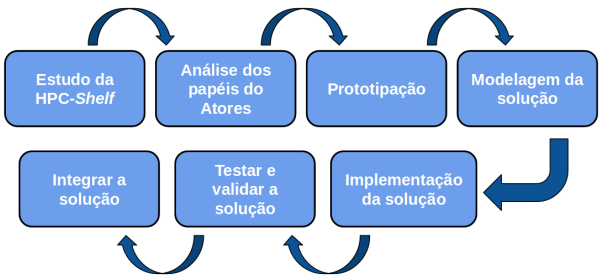
Nesta seção será descrito os procedimentos metodológicos que serão utilizados para elaboração deste trabalho. A metodologia adotada neste trabalho utilizará uma abordagem iterativa e incremental, após o estudo do ambiente (Seção 5.1.1) as etapas subsequentes serão realizadas repetidas vezes.

5.1 Organização da Metodologia

Inicialmente definimos os passos da metodologia como sendo as atividades de um processo de desenvolvimento de *software* iterativo e incremental, porém devido a complexidade da plataforma surgiu a necessidade de incluirmos a fase de estudo do ambiente.

Por isso, como mostra a Figura 5, começamos com o estudo da HPC-*Shelf*, para que tenhamos uma compreensão sólida do contexto onde este trabalho será inserido, na fase seguinte fizemos uma análise dos papéis do atores da HPC-*Shelf*, em seguida realizamos a prototipação das telas e a modelagem da solução, depois fizemos a implementação da mesma, e por fim os testes e a validação da ferramenta, culminando na integração com a HPC-*Shelf*, sendo este um dos objetivos específicos.

Figura 5 – Fluxo da Metodologia



Fonte: Próprio autor

5.1.1 Estudo da HPC-Shelf

A primeira fase do trabalho é o estudo da HPC-*Shelf* que consiste na leitura e compreensão das pesquisas relacionadas. Nesta fase foi estudado a sua organização arquitetural, bem como o relacionamento entre os elementos que a compõe e os atores que fazem uso do ambiente. Nesta fase foi compreendido as necessidades e requisitos do ambiente computacional e dos usuários que irão interagir com a mesma.

É importante ressaltar que esta fase se estende por todo projeto, pois o HPC-*Shelf* é uma plataforma ainda em desenvolvimento, por consequência, é necessário que estejamos sempre procurando compreender o ambiente computacional a medida que ele evolui, conforme as mudanças que ocorrerem.

5.1.2 Análise dos papéis de cada ator

De acordo com a Seção 2.4.1, cada ator desempenha um papel diferente na plataforma, por isso, nesta etapa foram analisadas as necessidades de cada tipo de ator do ambiente computacional, de acordo com as suas responsabilidades e conforme a visão que eles possuem da nuvem.

Assim, os requisitos foram elencados, detalhados, identificados, classificados segundo o tipo de requisito e ordenados conforme a prioridade.

5.1.3 Prototipação

O processo de prototipagem auxilia a compreensão das necessidades dos atores, pois esta técnica baseia-se num esboço inicial do layout do sistema, onde os principais objetivos são obter *feedback* do cliente com respeito a organização da interface, além de auxiliar na identificação de novas necessidades. Protótipos não necessitam ser muito detalhados e fieis a realidade, porém servem como guias para o desenvolvimento.

5.1.4 Modelagem da solução

Após terem sido compreendidas as atividades de cada tipo de ator, o conjunto de interfaces foi projetado para atender aos requisitos dos atores, isso serviu de estrutura base para o projeto como um todo.

Foi tomado como referência a arquitetura cliente-servidor, pois esta é uma arquitetura muito comum em aplicações *web*. A respeito das responsabilidades de cada parte da arquitetura (cliente e servidor), de modo geral, aspectos como interatividade ficaram a cargo do cliente, e aspectos como comunicação com a plataforma ficaram no lado servidor.

5.1.5 Implementação da solução

Após a modelagem da solução, a mesma foi implementada utilizando as duas linguagens de desenvolvimento, bem como as bibliotecas disponíveis para as mesmas. Para este trabalho foi utilizado a linguagem Java, para o desenvolvimento de controladores de informações dos atores e comunicação com os *web services* da nuvem, sendo auxiliado pelos *frameworks* Maven e Spring. Juntamente com a linguagem JavaScript, para exibição e controle interativo das informações provenientes dos controladores implementados em Java. O projeto foi colocado no Github[1] na forma de um repositório *online*, onde está disponível para comunidade e futuras atualizações.

5.1.6 Testar e validar a solução

Nesta fase os protótipos foram testados com o intuito de analisar o que foi implementado, avaliando se de fato o que foi desenvolvido atende aos requisitos do ambiente computacional. Nesta fase foi definidos os casos de teste, que foram utilizados para verificar a adequação da solução ao problema.

Estes casos de teste foram formulados com base em componentes com características e funcionalidades pré definidas, além de aplicações pré construídas a partir destes componentes. Também foi realizado uma avaliação da qualidade da solução, seguindo alguns critérios de usabilidade, para determinar se o que foi desenvolvido é minimamente usável.

5.1.7 Integrar a solução ao conjunto de ferramentas da plataforma.

A última fase do trabalho foi a integração da ferramenta, onde a mesma foi implantada em um servidor e o endereço da aplicação foi disponibilizado tanto no repositório da ferramenta quanto em outros repositórios da HPC-*Shelf*, para que os atores possam utilizá-la.

Assim, os atores da plataforma poderão interagir com a HPC-*Shelf* de uma maneira mais cômoda e amigável, através de um navegador comum.

[1] https://github.com/alyssongomes/HPC-Shelf-FrontEnd/

6 DESENVOLVIMENTO

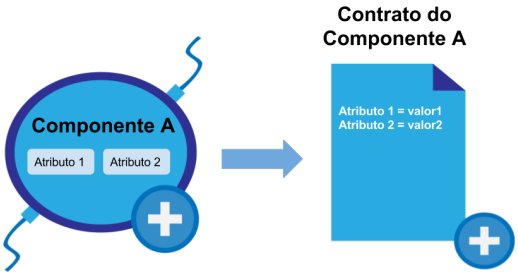
Nesta seção são apresentados em forma de relato pessoal, as atividades realizadas e os resultados alcançados durante todo o processo de desenvolvimento da solução *web* proposta.

6.1 Especificação dos papéis dos atores

De acordo com as primeiras fases dos procedimentos metodológicos foram: o estudo da plataforma e a análise dos papéis dos atores. A partir do estudo da mesma, foi possível compreender a estrutura base da plataforma, além do *web service* utilizado como mecanismo de comunicação entre os elementos arquiteturais, que também foi utilizado para alimentar a solução proposta.

A compreensão da plataforma foi vital para entender os papéis dos atores, além de compreender como os mesmos atuaram em nossa solução. Tendo em vista o foco deste trabalho, nos detivemos a atuação dos desenvolvedores de componentes e dos montadores de aplicação. Em nossa solução, os desenvolvedores de componentes devem atuar da seguinte forma: primeiramente os desenvolvedores deverão modelar os componentes abstratos, levando em consideração as características do modelo *Hash*; o segundo ato relaciona-se com o primeiro, pois o desenvolvedor deve criar um contrato contextual, onde este contrato pode ser entendido como uma instância do componente abstrato, e nele o desenvolvedor deverá especificar valores concretos às características do componente abstrato, o que podemos chamá-lo de componente concreto, estes dois atos são ilustrados na Figura 6.

Figura 6 – Relação entre o componente abstrato e o contrato



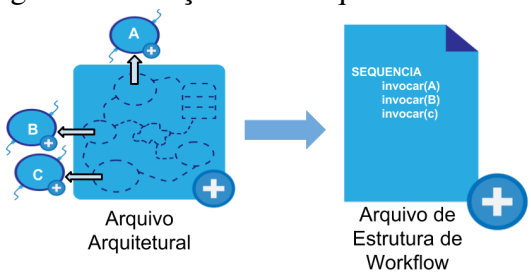
Fonte: Próprio Autor

O modelo *Hash*, adotado pela plataforma HPC-*Shelf*, permite que os componentes tenham unidades abstratas, essas unidades são funcionalidades que podem ser executadas pelo

componente. Esse aspecto do modelo influencia a nossa solução, pois os desenvolvedores devem associar arquivos de código-fonte a estas unidades. Por fim, o desenvolvedor deve ser capaz de executar seus componentes a partir de seus contratos, a fim de testá-los e avaliá-los.

A respeito dos montadores de aplicação, foi definido a seguinte atuação: os montadores deverão criar arquiteturas para suas aplicações, essas arquiteturas são compostas por componentes criados pelos desenvolvedores de componentes e mantidos no Core da plataforma; o segundo ato resume-se a especificação do comportamento da arquitetura, de forma que a aplicação se comporte da maneira desejada, essa especificação é feita por meio de *workflows*, constituído de diversos tipo de ações que regem a arquitetura. O relacionamento descrito é ilustrado na Figura 7. Portanto, ambas as estruturas (arquitetura e *workflow*) devem ser levadas em conta em nossa solução, além do relacionamento entre elas.

Figura 7 – Relação entre arquitetura e *workflow*



Fonte: Próprio Autor

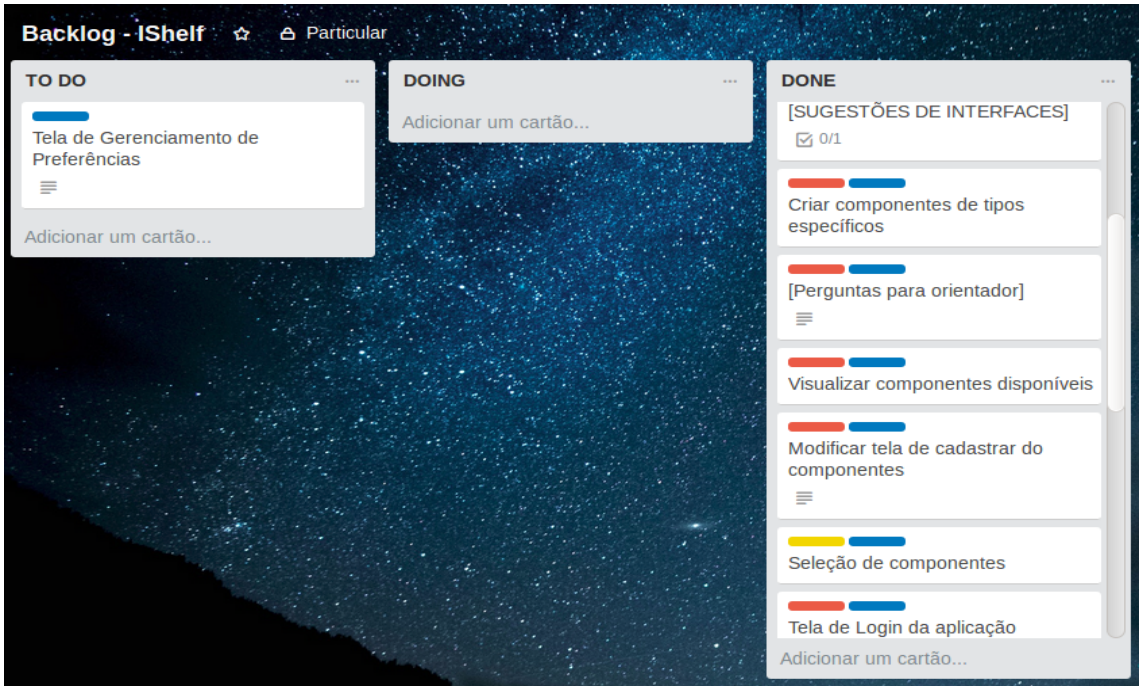
Para fins de organização, os requisitos estão definidos e descritos no Documento de Requisitos, Apêndice A. Para tornar o trabalho mais organizado foi utilizada a ferramenta Trello[1], pois possibilita que os requisitos sejam alocados em quadros, permitindo a adição de descrição, comentários e rótulos em forma de tags. A Figura 8 apresenta uma visão da ferramenta Trello, e como os requisitos dos atores são apresentados.

6.2 Projeto da Solução

O projeto da solução foi idealizado conforme a necessidade de consumir informações do *web service* da plataforma. Foi definida uma arquitetura do tipo cliente-servidor, onde o servidor é responsável por realizar a comunicação com o *web service* e repassar o conteúdo solicitado para o cliente, além disso, o servidor é responsável pela comunicação com o banco de dados da plataforma e gerenciar o mecanismo de autenticação da solução. O cliente é

[1] https://trello.com/

Figura 8 – Visão da ferramenta online Trello



Fonte: https://trello.com/

responsável por gerenciar a interação com o usuário, fornecendo elementos gráficos manipulados pelos usuários que permitirá a realização das tarefas elencadas na Seção 6.1.

O Apêndice B contém a versão final do Documento de Arquitetura, neste documento se encontra as visões lógicas, que incluem os pacote e as classes envolvidas na solução; a visão de processos que contemplam o fluxo dos dados entre as classes; e por fim a visão de implementação, que incluem relacionamento entre as classes em cenários específicos; este documento é útil para compreender de forma mais aprofundada a solução.

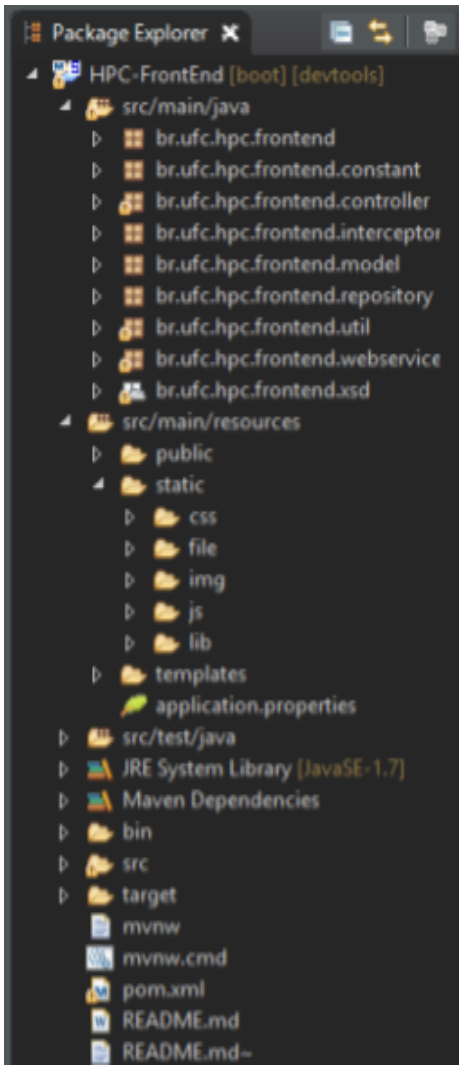
6.3 Implementação da Ferramenta

Esta seção trata da implementação da ferramenta de acordo com os requisitos definidos para a solução. A ferramenta foi denominada de *Interactive Shelf* (I*Shelf*), em referência ao objetivo da mesma, que é prover um ambiente interativo para atores da HPC-*Shelf*. Nas próximas subseções apresentarei detalhes sobre as etapas de configuração inicial do ambiente de desenvolvimento e da implementação de todas as funcionalidades definidas para a ferramenta.

6.3.1 Ambiente de desenvolvimento

A *Integrated Development Environment* (IDE), Eclipse Neon para desenvolvedores *web*, é a principal ferramenta de apoio ao desenvolvimento da solução *web* proposta. A IDE foi escolhida pelo fato de ser destinada ao desenvolvimento *web* com Java e amplamente utilizada, além da integração com a ferramenta Maven para gerenciar as dependências do projeto. Utilizei o Postgresql como Sistema de Gerenciamento de Banco de Dados (SGBD), e adotei o Apache Tomcat 7 como servidor de aplicação. A organização hierárquica do projeto é mostrada na Figura 9.

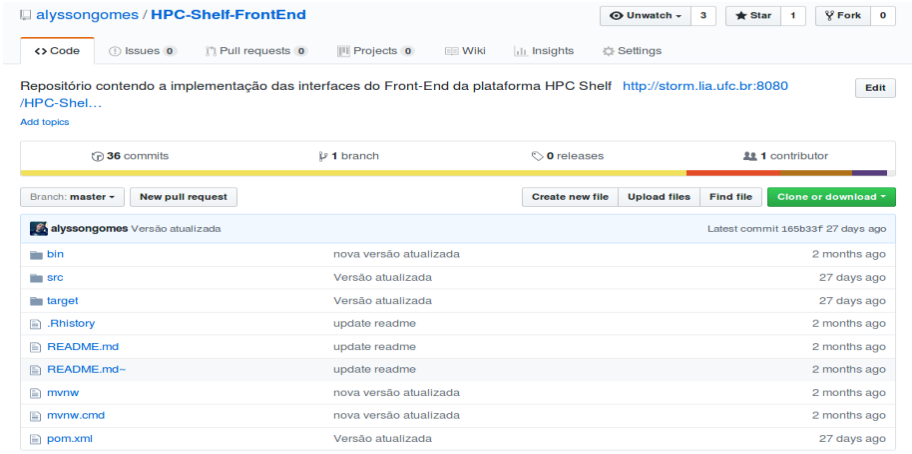
Figura 9 – Organização do projeto no Eclipse



Fonte: Próprio Autor

Criei um repositório[2] remoto no GitHub para a solução, onde a aplicação desenvolvida está disponível, como mostra a Figura 10. Ao passo que novas funcionalidades foram sendo desenvolvidas, por questões de segurança e integridade, foram sendo adicionadas ao repositório remoto, garantindo assim que caso algum problema ocorresse na máquina de trabalho o repositório remoto teria a última versão desenvolvida. O uso da ferramenta Git foi de grande valia para o controle de versões do sistema, onde juntamente com o GitHub pude sempre que necessário voltar o sistema para versões anteriores, de acordo com o histórico de mudanças registradas no GitHub.

Figura 10 – Repositório Remoto no GitHub



Fonte: https://github.com/alyssongomes/HPC-Shelf-FrontEnd

6.3.2 Implementação da Solução

Para implementação da solução inicialmente realizei a configuração dos *frameworks* adotados para o desenvolvimento. O Maven foi o primeiro *framework* a ser configurado, pois o mesmo gerenciou todas as dependências do projeto. Em seguida configurei o *framework* Spring para que pudesse fazer a conexão com o banco de dados, e desenvolver os controladores que requisitam informações ao *web service* da plataforma HPC-*Shelf* e os envia para as interfaces de usuário.

O desenvolvimento foi dividido em etapas, e em cada etapa foi desenvolvido uma interface que reunia um conjunto de funcionalidades oriundas dos requisitos levantados, que se encontram no Apêndice A. As interfaces são destinadas a tipos diferentes de atores, como este

[2] https://github.com/alyssongomes/HPC-Shelf-FrontEnd

projeto leva em conta apenas os desenvolvedores de componentes e os montadores de aplicações, dividimos as interfaces da seguinte forma:

- Desenvolvedores de componentes
  - Criação de componentes abstratos;
  - Criação de contratos contextuais;
  - Upload de códigos-fonte;
  - Execução de componentes;
- Montadores de Aplicações
  - Montagem de arquiteturas;
  - Montagem de workflows.

A seguir discutirei os resultados obtidos no desenvolvimento de cada uma das interfaces.

Criação de componentes abstratos

Nesta fase foram implementados os seguintes requisitos relativos a interface de criação de componentes abstratos:

- RF02: O sistema deve permitir que um usuário do tipo desenvolvedor, possa criar e editar um componente abstrato.
- RF03: O sistema deve listar os componentes disponíveis.
- RF04: Para cada componente selecionado, o sistema deve mostrar as informações do componente escolhido.
- RF05: O sistema deve permitir que o desenvolvedor inclua um ou mais funcionalidades no componente, representado por uma unidade abstrata.
- RF06: O sistema deve permitir que outros componentes possam ser associados ao componente criado.
- RF07: O sistema deve permitir que o desenvolvedor inclua características no componente criado, representadas por parâmetros de contexto, de qualidade, de custo e de ranking.
- RF11: O sistema deve permitir que componentes de tipos específicos possam ser criados (Ex.: componente da espécie plataforma).

Nesta fase nos concentramos no desenvolvimento de uma interface que permitisse que o desenvolvedor de componentes pudesse modelar um componente, onde este esteja em

conformidade com o modelo adotado pela plataforma HPC-*Shelf*. Nesta interface o desenvolvedor pode definir dados básicos a respeito do componente, como o nome, o tipo e o supertipo da qual o componente herdará determinadas características. O desenvolvedor também pode adicionar funções ao componente, denominadas unidades abstratas (como mostra o requisito RF05), essas funções definem o comportamento do componente, da mesma forma que um método define o comportamento de um objeto na POO.

Além disso, nesta interface é possível inserir características específicas do componente, chamadas de parâmetros (como mostra o requisito RF07), esses parâmetros podem ser de 4 tipos: contexto, qualidade, custo e ranking. Estes tipos de parâmetros possuem finalidades diferentes, pois são inerentes ao contexto da plataforma HPC-*Shelf*, no contexto da I*Shelf* esses parâmetros podem ser entendidos como atributos específicos do componente, assim como os atributos que descrevem um objetos na POO. Por fim, o desenvolvedor pode associar funções de outros componentes ao componente que está criando, ou seja, um componente pode conter componentes aninhados.

A Figura 11 mostra um exemplo de componente que está sendo modelado na I*Shelf*. O *Componente_A* é o componente principal, e está ligado a dois componentes aninhados *mlmgtbl* e *dirBindIntra*, onde o componente principal está usando as funções *unique* e *read_dir* dos respectivos componentes aninhados, no lado direito existe uma aba de propriedades onde podemos visualizar as unidades abstratas (representadas como quadrados no componente principal) e os parâmetros de contexto logo abaixo.

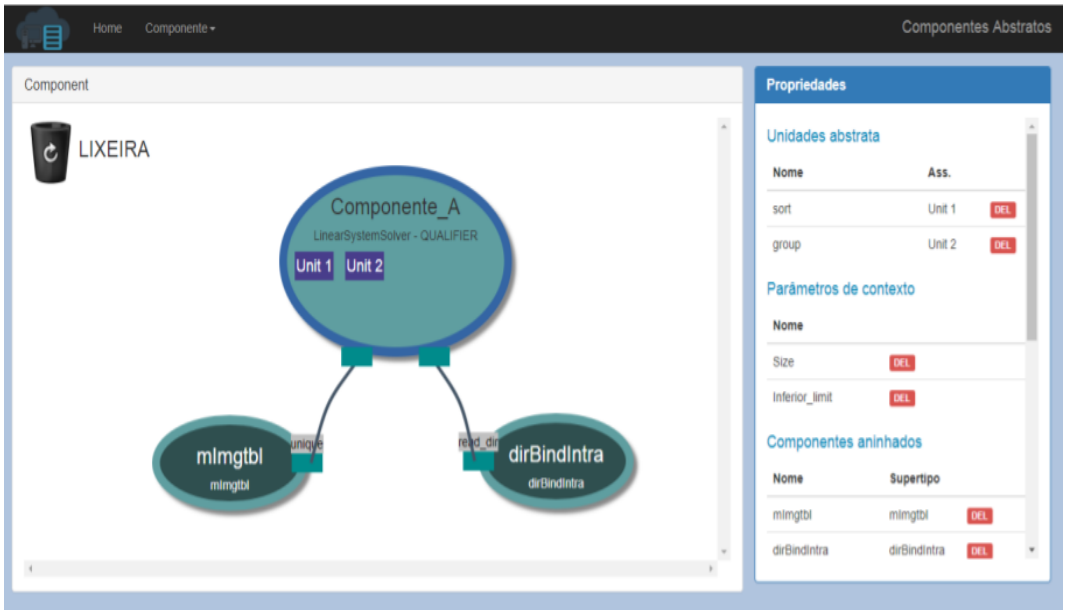
Criação de Contrato Contextual

Nesta fase foram implementados os seguintes requisitos relativos a interface de criação de contratos contextuais:

- RF09: O sistema deve permitir que instâncias (contratos contextuais) dos componentes possam ser criadas. Além de permitir que esta instância seja salva em um arquivo para o desenvolvedor.

Nesta fase me ocupei no desenvolvimento da interface de criação de contrato contextual, pois é nesta interface que o desenvolvedor poderá criar uma instância do componente abstrato criado na interface anterior. Nesta interface o desenvolvedor pode inserir informações básicas do contrato, como o componente abstrato que será a base do contrato, o nome e o tipo do

Figura 11 – Exemplo de Modelagem de um componente



Fonte: Próprio Autor

contrato. Um contrato contextual pode ser entendido como a instância de uma classe, pois nele podemos inserir valores aos atributos e argumentos nas funções, por isso, após a escolha do componente, a ferramenta lista os atributos e funções do componente, a partir disso o desenvolvedor pode informar valores aos diversos tipos de parâmetros.

Na Figura 12 podemos ver o que foi descrito anteriormente, onde o componente selecionado se chama *Software*, e logo abaixo é listado os parâmetros de contexto do componente, onde serão inseridos os valores.

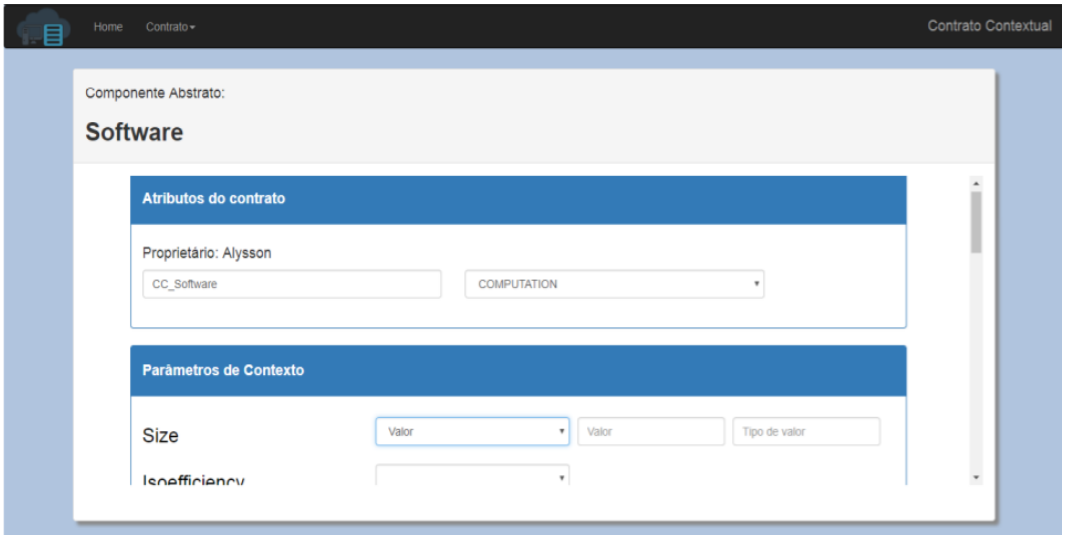
Upload de códigos-fonte

Nesta fase foram implementados os seguintes requisitos relativos a interface de *upload* de códigos-fonte:

- RF08: O sistema deve permitir que o desenvolvedor faça o *upload* de arquivos de código-fonte para uma determinada unidade.

Como já mencionei anteriormente, os componente possuem unidades abstratos que representam as funções do componente, por isso é necessário que o desenvolvedor associe um código-fonte à unidade do componente (como mostra o requisito RF08). Na Figura 13 é possível visualizar a interface de submissão de um arquivo de código, onde é possível informar a versão

Figura 12 – Exemplo de Contrato Contextual



Fonte: Próprio Autor

do arquivo, o tipo de arquivo, os parâmetros para execução do arquivo e o componente a qual será vinculado o arquivo.

Figura 13 – Interface de *Upload* de Arquivos

Fonte: Próprio Autor

Execução de componentes

Nesta fase foram implementados os seguintes requisitos relativos a interface de

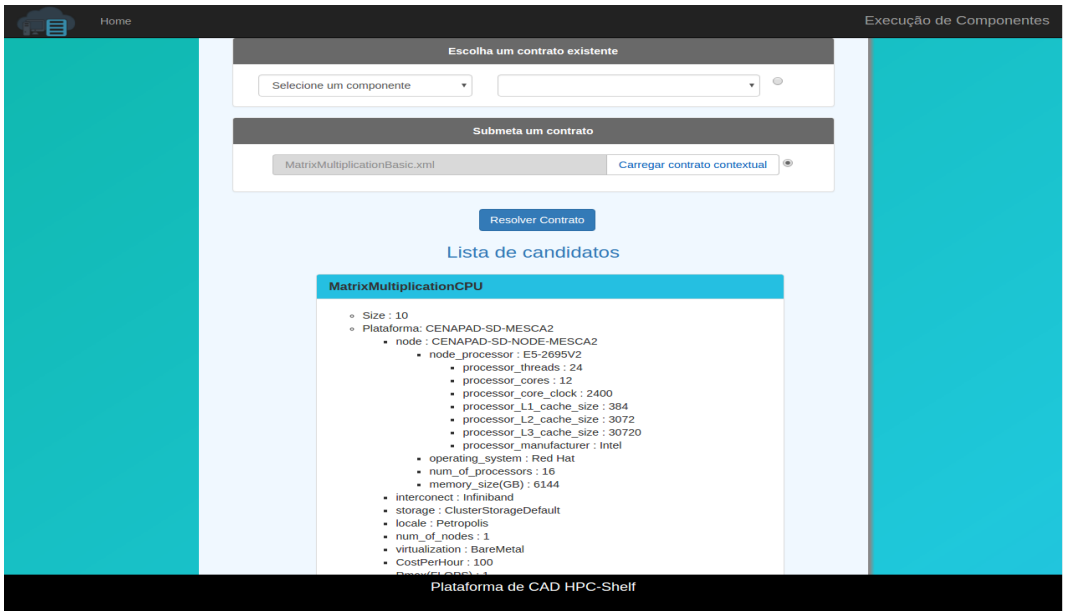
execução de componentes:

- RF10: O sistema deve permitir que as instâncias dos componentes possam ser executadas. Além disso, deve permitir que o resultado do processamento da instância seja salvo em um arquivo.

Após a especificação do contrato contextual, e a associação de códigos-fonte às unidades do componente, o desenvolvedor pode realizar a execução do componente. A execução é feita a partir do contrato especificado pelo desenvolvedor, e na I*Shelf* o desenvolvedor pode selecionar contratos já cadastrados na plataforma ou pode submeter um arquivo contendo o contrato, em seguida o contrato é resolvido. Na etapa de resolução o contrato é enviado para HPC-*Shelf*, onde a mesma calcula as possíveis plataformas computacionais (servidores, *clusters*, grades computacionais, etc.) que podem receber o componente para ser executado, sendo esta lista de candidatos o retorno da etapa de resolução.

Depois da resolução, o desenvolvedor pode realizar a implantação do componente em um dos candidatos listados, e ser de fato executado. A Figura 14 mostra o processo descrito anteriormente, onde o contrato denominado *MatrixMultiplicationBasic* foi resolvido e gerou uma lista de candidatos, um deles denominado de *MatrixMultiplicationCPU*, onde a lista de candidatos pode ser baixado, estando de acordo com o requisito RF10.

Figura 14 – Tela de execução



Fonte: Próprio Autor

Montagem de arquiteturas

Nesta fase foram implementados os seguintes requisitos relativos a interface de montagem de arquiteturas de aplicações:

- RF12: O sistema deve permitir que os montadores possam criar a arquitetura de aplicações baseadas em componentes.

Após o registro dos componentes abstratos e contratos contextuais especificados pelo desenvolvedor de componentes, o montador de aplicações pode desempenhar o seu papel. Uma das primeiras tarefas do montador de aplicações é a modelagem da arquitetura da aplicação, pois ele deve especificar quais elementos farão parte da solução que ele está modelando, bem como as fontes de dados que ele irá utilizar e quais computações irá aplicar sobre esses dados. Todas essas atividades são oriundas do requisito RF12, por isso, na I*Shelf* o montador pode modelar uma arquitetura a partir de componentes e contratos registrados na plataforma HPC-*Shelf*, podendo incluir na arquitetura: computações, plataformas de computação, estruturas de dados, componentes de conexão, etc, além de estabelecer um relacionamento entre esses componentes por meio de ligações representadas graficamente.

Na Figura 15 podemos visualizar um exemplo de arquitetura da implementação do modelo MapReduce montada na I*Shelf*, onde a mesma possui duas estruturas de dados denominadas de *DataSink* e *DataSource*, também possui três componentes de computação denominados *Reducer*, *Combiner* e *Mapper*, além de conter duas plataformas de computação *PLAT_A* e *PLAT_B*, os componentes que formam a arquitetura estão sendo sincronizados por um componente chamado *Workflow*, que por sua vez se conecta com os demais elementos arquiteturais. O relacionamento entre os elementos é representado por setas direcionadas e não direcionadas, de forma que os pares de elementos conectados estão descritos na aba de propriedades no lado direto.

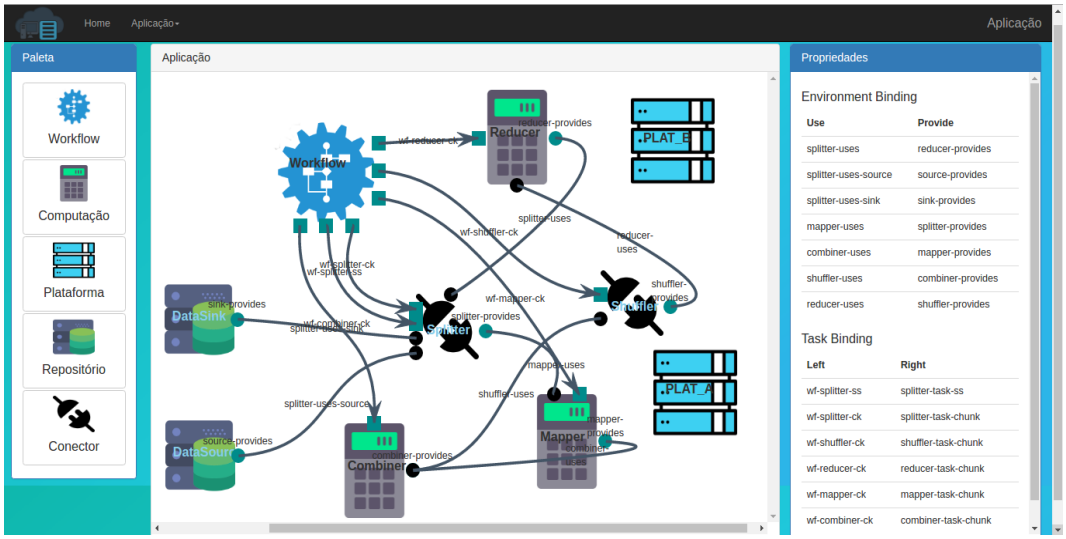
Montagem de *workflows*

Nesta fase foram implementados os seguintes requisitos relativos a interface de montagem de arquiteturas de *workflows*:

- RF13: O sistema deve permitir que o fluxo dos *workflows* possa ser definido a partir de sua arquitetura.

Após a modelagem da arquitetura, a próxima tarefa do montador de aplicações será

Figura 15 – Tela de montagem de arquitetura

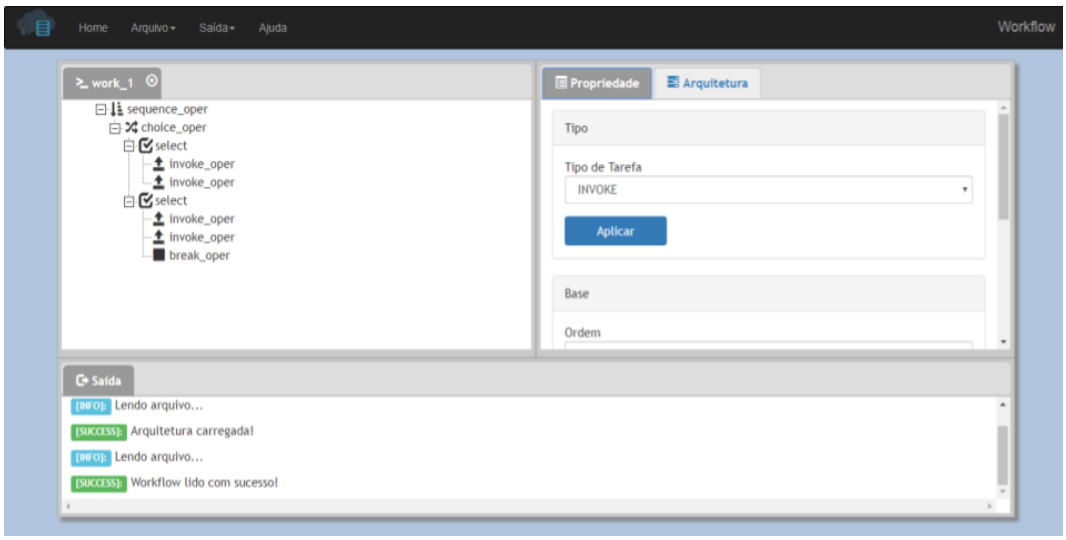


Fonte: Próprio Autor

a especificação do arquivo de *workflow* que orquestrará a aplicação, como destaca o requisito RF13. Na I*Shelf* o montador pode realizar a especificação do *workflow*, criando uma estrutura hierárquica que controlará o comportamento da aplicação. Como foi dito na Seção 2.3, um *workflow* pode ser definido por meio de diretivas e comandos semelhantes a uma linguagem de programação, a plataforma HPC-Shelf utiliza esta mesma noção para especificar um *workflow*, por isso, na interface de especificação de *workflows* da IShelf é utilizada a mesma idéia, pois a mesma possui comandos de iteração, seleção, escopos de execução sequencial e paralelo, onde todos este comandos estão definidos na linguagem do SAFe.

Na Figura 16 destaco o exemplo de um *workflow* especificado na I*Shelf*. No lado esquerdo, na aba denominada *work_1*, vemos o *workflow* sendo iniciado com um escopo de execução sequencial, dentro deste escopo temos as operações *choice_oper* e *select* que podem ser entendidas como os comandos *switch* e *case* de uma linguagem de programação.

Figura 16 – Tela de montagem de *Workflow*



Fonte: Próprio Autor

7 TESTES E MEDIÇÕES

Durante o desenvolvimento da ferramenta foram executados testes de desenvolvedor, à medida que novas funcionalidades foram desenvolvidas, porém esses testes são muito específicos e não abordam cenários complexos durante o uso do sistema.

Estes cenários descrevem estórias (situações de teste) que contextualizam determinadas circunstâncias e atividades que são comuns no contexto do usuário da ferramenta, e normalmente envolvem diversas funcionalidades. Por isso, foram definidos e executados casos de teste mais complexos baseados nos cenários mais típicos. Dentre os principais cenários, definimos uma situação em que envolve a modelagem de componentes abstratos, a definição de seus contratos e a execução dos componentes, além da situação relativa a especificação de uma aplicação por meio da modelagem de sua arquitetura e de seus respectivos *workflows*.

De acordo com os requisitos especificados para a solução, definimos e executamos casos de teste que foram organizados em 7 (sete) grupos, os quais contém testes específicos para cada tipo de interface, por exemplo: no primeiro grupo estão os testes relativos a interface de modelagem de componentes abstratos, no segundo estão os testes específicos para interface contratos contextuais e assim por diante. Nesta seção são apresentados os resultados obtidos com o processo de teste e avaliação da solução.

7.1 Especificação dos casos de teste

A partir dos requisitos levantados nas fases iniciais do projeto, identifiquei cenários que seriam relevantes para validação da ferramenta, com esses cenários definimos um conjunto de casos de teste. Os casos de teste foram especificados e agrupados de acordo com as funcionalidades que seriam avaliadas, buscando sempre abordar os testes de fronteira das funcionalidades. Ao todo foram especificados 42 (quarenta e dois) casos de teste distribuídos nos 7 grupos que correspondiam as principais interfaces, o documento que contém todos os casos de teste se encontra no repositório[1] remoto do projeto.

O Quadro 1 representa um caso de teste presente no documento de testes do projeto, neste caso está sendo testado a criação de um novo componente, como está descrito na seção de *objetivo do teste*, além das *pré-condições* que descrevem o estado anterior, ou *ações* anteriores a realização do mesmo, a partir disso são enumeradas uma *sequência de passos* a serem executados

[1] https://github.com/alyssongomes/HPC-Shelf-FrontEnd/tree/master/Documentos

Quadro 1 – Criar um novo componente

Caso de Teste 1: Criar componente - Criar um novo componente		
Autor: Alysson Gomes		
Objetivo do Teste: Verificar se o usuário logado consegue dar início a criação de um novo componente.		
Pré-condições: Usuário estar logado		
#	Ações do passo	Resultados esperados
1	Informe o “nome” do novo componente com o valor “MatrixMultiplication”;	
2	Selecione o “tipo de componente” com o valor “COMPUTATION”;	O sistema listará os tipos de componentes;
3	Selecione o “supertipo” do novo componente com o valor “MatrixOperation”.	O sistema lista os dados do componente selecionado.
4	Selecione a opção de criar componente	O sistema deve mostrar o workspace com componente criado.
Tipo de Execução: Manual		
Prioridade: Alta		

Fonte: Próprio Autor

bem como os *resultados esperados* na execução de cada passo.

7.2 Planejamento e Medição de Qualidade

Além dos casos de teste especificados na etapa anterior, também elaborei um plano de medição de qualidade da ferramenta, com o intuito de avaliar a usabilidade da I*Shelf*. Isso se tornou relevante para o projeto pelo fato de nos ter despertado um interesse em saber quanto esforço é necessário para realizar tarefas comuns ao contexto da ferramenta.

No plano de medição especifiquei o procedimento de medição, contendo o perfil dos usuários da ferramenta, em seguida defini um conjunto de tarefas que estão atreladas a cenários específicos, além de quais métricas seriam utilizadas para avaliar a ferramenta, e por fim o procedimento de interpretação dos dados coletados. Para mais detalhes em relação ao plano de medição, o mesmo se encontra no relatório de medição no Apêndice C.

7.3 Execução dos testes e Medição da qualidade

Ao executar os casos de teste pude perceber a necessidade de criar novos casos de testes e acrescentá-los nos seus respectivos conjuntos de teste, buscando garantir que determi-

Quadro 2 – Resolução de contrato contextual

Caso de Teste 1: Executar componente - Carregar e resolver um contrato				
Autor: Alysson Gomes				
Objetivo do Teste: Verificar se o usuário logado consegue carregar um contrato pré cadastrado				
Pré-condições: Usuário estar logado, lista dos componentes				
#	Ações do passo	Resultados esperados	Notas de Execução	Status de Execução
1	Selecionar um componente com o nome “MatrixOperation”	Listar os contratos		Passou
2	Selecionar um contrato com o nome "MatrixMultiplication"			Passou
3	Selecionar o “Radio Button” da aba “Escolha um contrato existente”			Passou
4	Clicar em “Resolver Contrato”	Listar candidatos		Passou
Tipo de Execução: Manual				
Prioridade: Alta				
Detalhes de Execução				
Testador: Alysson Gomes				
Resultado Exec.:Passou				
Modo Exec.:Manual				

Fonte: Próprio Autor

nados cenários do sistema estavam sendo verificados. Em termos de desenvolvimento e testes, uma clara vantagem surgiu, o testador também é o desenvolvedor das funcionalidades, com isso, corrigi funcionalidades à medida que erros surgiam.

O Quadro 2 apresenta a execução de um caso de teste, onde inclui informações adicionais como as notas de execução para levar em conta observações feitas durante a execução, e também o status da execução que indica se um determinado passo correspondeu aos resultados esperados ou não. Todos os planos de testes podem ser consultados no repositório remoto do projeto.

Como resultado dos testes, algumas funcionalidades apresentaram defeitos, o que mostra a importância da execução de testes em um projeto de *software*. Devido ao agrupamento dos testes, vimos que no grupo relativo aos componentes abstratos houve mais erros a serem corrigidos, seguido pelo grupo dos testes da interface de registro de contratos e da montagem de *workflows* e por fim a interface de execução de componentes.

Alguns dos defeitos encontrados durante a execução dos casos de teste de cada grupo foram os comportamentos inesperados decorrentes de campos vazios, pois os dados não podiam ser organizados da forma correta e assim serem submetidos ao servidor da aplicação, além da falta de *feedback* de algumas funcionalidades, como a falta de candidatos disponíveis após a resolução de um contrato. Outros defeitos encontrados nos grupos de teste, principalmente no grupo de modelagem de componentes e registro de contratos, estavam relacionados à compreensão equivocada de alguns aspectos, como o relacionamento entre tipos diferentes de parâmetros de um mesmo componente.

A respeito da medição de qualidade, foi realizado a medição com um grupo de 5 (cinco) voluntários que estavam dentro do perfil definido no plano, que considerou usuários que possuem alguma experiência com ferramentas de modelagem como Astah[2] e brModelo[3], ou com ferramentas de definição de fluxo como Pentaho[4]. Cada um dos voluntários realizou as 5 (cinco) tarefas definidas, e através de uma ferramenta de gravação da tela pude capturar as interações dos usuários com as interfaces, e assim pude obter valores para as métricas que escolhi. Os dados brutos dos experimentos de cada usuário estão descritos detalhadamente nos resultados no relatório de medição.

7.4 Validação da Solução e Resultados da Medição

Desde o início do projeto as funcionalidades foram revisadas junto aos principais interessados do projeto, a saber, os alunos do MDCC envolvidos no projeto da HPC-*Shelf*. Validar a solução com os interessados no projeto foi extremamente benéfico para o projeto. Foram realizadas reuniões durante o processo de desenvolvimento para mostrar as funcionalidades da ferramenta. Durante essas reuniões surgiram sugestões de melhorias que foram incorporadas ao projeto, e erros que precisavam ser corrigidos. Diversas funcionalidades foram verificadas, mas algumas tiveram uma atenção especial, como a criação de componentes abstratos e a especificação de contratos.

Em relação a medição de qualidade, a coleta e compilação dos dados, que também se encontram no Apêndice C, me permitiu chegar algumas conclusões a respeito da ferramenta. A primeira conclusão é que os elementos na interfaces ainda não estão bem dispostos, isso vale para as interfaces de criação de componentes e contratos, isso também pôde ser averiguado nas

[2] http://astah.net/
[3] http://www.sis4.com/brModelo/
[4] http://www.pentaho.com/

gravações das interações dos usuários, que influenciou diretamente no tempo de execução das tarefas e na quantidade de *clicks*, pois essa métrica mostra quantos *clicks* foram necessários para a realização de um tarefa, permitindo mensurar o esforço necessário para interagir com uma determinada interface, e juntamente com a gravação das interações estes fatores tornam-se importantes para compreensão do comportamento do usuário no manuseio da interface, indicando sua navegação na utilização da tela e o esforço necessário em tal navegação. A segunda conclusão é que a interface de montagem de *workflows* é mais complexa do que as demais, pois nos testes identifiquei que os usuários precisaram de mais tempo para compreender a interface, isso também foi visível na quantidade erros relativos a tarefa de montagem de *workflows*.

8 CONSIDERAÇÕES FINAIS

Este trabalho apresentou o processo de análise, desenvolvimento e validação da ferramenta *Interactive Shelf* (I*Shelf*), que se encontra no contexto do desenvolvimento do ambiente de nuvem HPC-*Shelf*. Tendo em vista um dos objetivos da HPC-*Shelf*, que consiste em fornecer uma ambiente de desenvolvimento de aplicações computacionalmente intensivas, que possa ser executado em um ambiente de nuvem utilizando a abstração de componentes, surgiu a necessidade de uma solução que permitisse criar estes componentes em alto nível, levando em conta o modelo pré-definido na plataforma. Como uma maneira de suprir esta necessidade, foi desenvolvida a ferramenta I*Shelf*.

Para que esta ferramenta fosse desenvolvida, inicialmente compreendemos o contexto onde ela seria inserida, identificando quais seriam os atores da plataforma que fariam uso da ferramenta, e ao final da análise chegamos à conclusão de que os principais atores seriam os desenvolvedores de componentes e o montadores de aplicações. Analisando o papel desses atores, identificamos um conjunto de necessidades e requisitos importantes que fariam parte da solução.

Durante o processo de desenvolvimento foram realizadas reuniões para avaliar o que foi desenvolvido, os alunos do programa MDCC envolvidos no projeto da HPC-*Shelf*, verificaram a adequação da ferramenta à necessidade a plataforma.

Além das reuniões, para garantir o bom funcionamento da ferramenta foram especificados e executados 42 (quarenta e dois) casos de teste, divididos em 7 (sete) grupos correspondentes as principais interfaces. Para avaliarmos a qualidade da ferramenta no que diz respeito a usabilidade, realizamos uma medição com um conjunto de 5 (cinco) voluntários, que realizaram 5 (cinco) tarefas que serão comuns no contexto da I*Shelf*. Por meio da avaliação conseguimos chegar a algumas conclusões, (i) a disposição dos elementos nas interfaces ainda não é a ideal, o que influenciou o tempo de realização de tarefas e a quantidade de *clicks*, (ii) também vimos que a interface de montagem de *workflows* ainda possui uma certa complexidade que torna difícil a sua compreensão. Contudo, apesar dos erros as atividades foram completadas, o que nos mostra que apesar das dificuldades é possível modelar as principais entidades da plataforma na ferramenta desenvolvida.

Para resolvermos os problemas identificados na avaliação, sugiro a adoção das seguintes medidas: tornar mais explícito as principais funcionalidade das interfaces, com foco nas telas de modelagem de componentes abstratos e registro de contratos; para auxiliar os atores

também podemos incluir tutoriais animados para exemplificar a utilização das interfaces e das principais funcionalidades.

No decorrer do processo foi possível perceber que as tecnologias adotadas facilitaram o desenvolvimento da solução *web*. A utilização do JavaScript juntamente com a biblioteca jQuery facilitou o desenvolvimento dos elementos gráficos interativos, e dos componentes arrastáveis presentes nas interfaces de modelagem de componentes e arquitetura. A linguagem Java facilitou a comunicação com o *web service* da plataforma HPC-*Shelf*, além do *framework* Spring que auxiliou o desenvolvimento dos controladores que alimentam as interfaces, também auxiliou o gerenciamento da sessão do usuário e a comunicação com o banco de dados.

Espera-se que a ferramenta supra a necessidade da HPC-*Shelf*, mas permitindo que novas funcionalidades possam ser incorporadas sem mudanças significativas na arquitetura da mesma. A I*Shelf* está disponível como código aberto.

8.1 Trabalhos Futuros

A avaliação da ferramenta foi importante para identificarmos aspectos que precisam ser melhorados, por isso conseguimos observar algumas possibilidades de trabalhos futuros, são eles:

- Melhorar a disposição dos elementos na interface e incorporar um *tour* pelas principais funcionalidades para os usuários que irão utilizar a ferramenta pela primeira vez;
- Incluir tutoriais animados para exemplificar a utilização das principais funcionalidades;
- Realizar uma pesquisa entre os atores da HPC-*Shelf* para identificar novas opções de interfaces;
- Por fim, incorporar como nova funcionalidade um serviço que permita submeter e executar a arquitetura e o workflow no SAFe da HPC-*Shelf*.

REFERÊNCIAS

ABADI, D. J. Data management in the cloud: Limitations and opportunities. **IEEE Data Eng. Bull.**, Citeseer, v. 32, n. 1, p. 3–12, 2009.

AHUJA, S. P.; MANI, S. The state of high performance computing in the cloud. **Journal of Emerging Trends in Computing and Information Sciences**, v. 3, n. 2, p. 262–266, 2012.

AL-ALAM, W. G. **Estudo Sobre Programação Orientada a Componentes**. 2010. Disponível em: <http://components-oriented-programming.blogspot.com.br/>. Acesso em: 27 Fev 2017.

AL-ALAM, W. G. **Alocação de recursos em uma nuvem de Serviços de computação de alto desempenho Baseada em componentes através de contratos Contextuais.** Tese (Doutorado) — Universidade de Federal do Ceará, 2013. (*abstract*).

BAGESTAN, F. H. **Estudo Comparativo de Frameworks PHP, com Enfoque no CodeIgniter**. 2016. Monografia (Bacharel em Informática), Passo Fundo (Instituto Federal de Educação, Ciência e Tecnologia Sul-Riograndense), Rio Grande do Sul, Brazil.

BARKER, A.; HEMERT, J. V. Scientific workflow: a survey and research directions. In: **International Conference on Parallel Processing and Applied Mathematics**. [S.l.]: Springer, 2007. p. 746–753.

BAUDE, F.; BERGEL, A.; CAROMEL, D.; HUET, F.; NANO, O. *et al.* Ic2d: Interactive control and debugging of distribution. In: SPRINGER. **International Conference on Large-Scale Scientific Computing**. [S.l.], 2001. p. 193–200.

BAUDE, F.; CAROMEL, D.; HUET, F.; MESTRE, L.; VAYSSIÈRE, J. Interactive and descriptor-based deployment of object-oriented grid applications. In: IEEE. **High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on**. [S.l.], 2002. p. 93–102.

BERNICHE, E. F.; NERIS, V. P. de A. Análise comparativa entre bibliotecas javascript para o desenvolvimento de interfaces web ricas. **Revista TIS**, v. 2, n. 1, 2013.

BRAMLEY, R.; CHIU, K.; DIWAN, S.; GANNON, D.; GOVINDARAJU, M.; MUKHI, N.; TEMKO, B.; YECHURI, M. A component based services architecture for building distributed applications. In: **High-Performance Distributed Computing, 2000. Proceedings. The Ninth International Symposium on**. [S.l.]: IEEE, 2000. p. 51–59.

BUYYA, R.; BROBERG, J.; GOSCINSKI, A. M. **Cloud computing:** Principles and paradigms. [S.l.]: John Wiley & Sons, 2010. v. 87.

CRNKOVIC, I.; CHAUDRON, M.; SENTILLES, S.; VULGARAKIS, A. A classification framework for component models. **Software Engineering Research and Practice in Sweden**, p. 3, 2007.

CRUZ, F. **Python,** escreva seus primeiros programas. [S.l.]: Casa do Código, 2015.

DIAKOPOULOS, N.; CASS, S. **Interactive:** The Top Programming Languages 2016. 2016. Disponível em: <http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2016>. Acesso em: 8 Abr 2017.

FLANAGAN, D. **JavaScript:** The definitive guide: Activate your web pages. [S.l.]: "O'Reilly Media, Inc.", 2011.

GERASIMOV V.V.; PONOMAREV, I.; LISHCHENKO, Analysis of main technologies for development of web applications on java platform. **System technologies**, 2015.

HARRISON, A.; TAYLOR, I.; WANG, I.; SHIELDS, M. Ws-rf workflow in triana. **International Journal of High Performance Computing Applications**, SAGE Publications, v. 22, n. 3, p. 268–283, 2008.

HOLLINGSWORTH, D.; HAMPSHIRE, U. Workflow management coalition: The workflow reference model. **Document Number TC00-1003**, Citeseer, v. 19, 1995.

KRAFZIG, D.; BANKE, K.; SLAMA, D. **Enterprise SOA:** service-oriented architecture best practices. [S.l.]: Prentice Hall Professional, 2005.

KUNDU, A.; BANERJEE, A.; SAHA, P. Introducing new services in cloud computing environment. **International Journal of Digital Content Technology and its Applications, AICIT**, p. 143–152, 2010.

MATTOSO, M.; WERNER, C.; TRAVASSOS, G.; BRAGANHOLO, V.; MURTA, L.; OGASAWARA, E.; OLIVEIRA, F.; MARTINHO, W. Desafios no apoio à composição de experimentos científicos em larga escala. **Seminário Integrado de Software e Hardware, SEMISH**, v. 9, p. 36, 2009.

MAVEN. **What is Maven?** 2017. Disponível em: <https://maven.apache.org/what-is-maven.html>. Acesso em: 9 Nov 2017.

MELL, P.; GRANCE, T. *et al.* **The NIST definition of cloud computing**. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011.

OINN, T.; ADDIS, M.; FERRIS, J.; MARVIN, D.; SENGER, M.; GREENWOOD, M.; CARVER, T.; GLOVER, K.; POCOCK, M. R.; WIPAT, A. *et al.* Taverna: a tool for the composition and enactment of bioinformatics workflows. **Bioinformatics**, Oxford Univ Press, v. 20, n. 17, p. 3045–3054, 2004.

PHP. **O que é o PHP?** 2017. Disponível em: <https://secure.php.net/manual/pt_BR/intro-whatis.php>. Acesso em: 8 Abr 2017.

ROGERS, Y.; SHARP, H.; PREECE, J. **Design de interação:** ém da interação humano-computador. [S.l.]: Bookman, 2013.

SHEN, X.; LIAO, W.-K.; CHOUDHARY, A. An integrated graphical user interface for high performance distributed computing. In: **Database Engineering and Applications, 2001 International Symposium on.** [S.l.]: IEEE, 2001. p. 237–242.

SILVA, J. d. C. **Um arcabouço para a construção de aplicações baseadas em componentes sobre uma plataforma de nuvem computacional para serviços de computação de alto desempenho**. Tese (Doutorado) — Universidade de Federal do Ceará, 2016.

STALLINGS, W. **Arquitetura e Organização de Computadores 8a Edição**. [S.l.]: Prentice-Hall, Pearson, 2010.

WANG, A. J. A.; QIAN, K. **Component-oriented programming**. [S.l.]: John Wiley & Sons, 2005.

YAN, Y. **Scheduling scientific workflow applications in computational grids**. [S.l.]: Citeseer, 2007. v. 68.

APÊNDICE A – DOCUMENTO DE REQUISITOS

UNIVERSIDADE
FEDERAL DO CEARÁ

Documento de Requisitos

Responsável
Alysson Gomes de Sousa

Quixadá
2017

Histórico de Revisão

Data	Versão	Descrição	Autor
30/05/2017	1.0	Versão inicial do documento.	Alysson Gomes de Sousa
04/06/2017	2.0	Inclusão dos requisitos funcionais e não funcionais	Alysson Gomes de Sousa

1. Introdução

Este documento tem o objetivo de especificar os requisitos do sistema de apoio a desenvolvedores e usuários da plataforma HPC-*Shelf*, denominado Interactive *SHELF* (I*Shelf*).

2. Descrição do Problema

O problema consiste na inexistência de um sistema de apoio à desenvolvedores e usuários de aplicações de computação de alto desempenho na plataforma HPC-*Shelf*.

3. Descrição Geral do Sistema

O sistema I*Shelf*, é uma ambiente *web* que funciona como camada de *software* que atua sobre a plataforma HPC-*Shelf*, e deverá prover funcionalidades para criação e gerenciamento de componentes, aplicações e plataformas de execução de forma interativa e intuitiva.

4. Finalidade

O sistema I*Shelf* permitirá que desenvolvedores e especialistas da computação de alto desempenho tenham a capacidade de criar aplicações com alta demanda computacional em um plataforma em nuvem de forma amigável e interativa.

5. Convenções e Termos

5.1. Identificação dos requisitos

Por convenção, os requisitos serão referenciados por siglas, sendo [RFXX] para requisitos funcionais [RNFXX] para requisitos não-funcionais e [RDXX] para requisitos de domínio, sendo ‘XX’ a representação de uma sequência numérica iniciando de ‘01’ para cada categoria de requisitos.

5.2. Prioridade dos requisitos

Para estabelecer as prioridades dos requisitos em relação ao sistema como um todo, foram adotadas as seguintes denominações: **Baixa**, **Média** e **Alta**.

Baixa: requisitos com essa prioridade não afetam a funcionalidade básica do sistema, ou seja, o sistema pode funcionar de forma satisfatória sem eles. Esses requisitos podem ser implementados em futuras versões do sistema, caso o tempo de desenvolvimento seja curto.

Média: O sistema funciona sem esse requisito, mas de forma não satisfatória. Esses requisitos devem ser implementados na versão inicial do sistema, mas se não forem, o sistema pode ser implantado sem eles.

Alta: São os requisitos sem os quais o sistema não entra em funcionamento. O sistema não pode ser implantado sem esses requisitos.

6. Requisitos

Nesta seção os requisitos são apresentados, eles estão divididos em três subseções, Requisitos funcionais, Requisitos não-funcionais e Requisitos de domínio.

6.1. Requisitos funcionais

Esses requisitos são refletidos como funcionalidades no sistema, eles descrevem exatamente como o sistema deve funcionar operar. Os requisitos do sistema de agendamento são descritos a seguir:

Identificador	Prioridade	Descrição
RF01	Alta	O sistema deve conter um controle de acesso para distinguir cada tipo de usuário.
RF02	Alta	O sistema deve permitir que um usuário do tipo desenvolvedor, possa criar e editar um componente abstrato.
RF03	Alta	O sistema deve listar os componentes disponíveis.
RF04	Media	Para cada componente selecionado, o sistema deve mostrar as informações do componente escolhido.
RF05	Media	O sistema deve permitir que o desenvolvedor inclua um ou mais funcionalidades no componente, representado por uma unidade abstrata.
RF06	Media	O sistema deve permitir que outros componentes possam ser associados ao componente criado.
RF07	Media	O sistema deve permitir que o desenvolvedor inclua características no componente criado, representadas por

		parâmetros de contexto, de qualidade, de custo e de ranking.
RF08	Baixa	O sistema deve permitir que o desenvolvedor faça o *upload* de arquivos de código-fonte para uma determinada unidade.
RF09	Alta	O sistema deve permitir que instâncias (contratos contextuais) dos componentes possam ser criadas. Além de permitir que esta instância seja salva em um arquivo para o desenvolvedor.
RF10	Alta	O sistema deve permitir que as instâncias dos componentes possam ser executadas. Além disso, deve permitir que o resultado do processamento da instância seja salvo em um arquivo.
RF11	Alta	O sistema deve permitir que componentes de tipos específicos possam ser criados (Ex.: componente da espécie plataforma).
RF12	Media	O sistema deve permitir que os montadores possam criar a arquitetura de aplicações baseadas em componentes.
RF13	Media	O sistema deve permitir que o fluxo dos *workflows* possa ser definido a partir de sua arquitetura.

6.2. Requisitos não-funcionais

Esses requisitos são refletidos como restrições e qualidade do sistema, e são descritos a seguir:

Identificador	Descrição
RNF01	O sistema deverá ser desenvolvido na linguagem de programação Java e JavaScript, usando paradigma orientado a objetos juntamente com o padrão MVC.
RNF02	O sistema deverá ser uma aplicação web.
RNF03	A comunicação com a plataforma HPC-*Shelf* deve ser feita através do *web service* através da linguagem XML.

6.3. Requisitos de domínio

Esses requisitos são refletidos como características do domínio do sistema, e são descritos a seguir:

Identificador	Descrição
RD01	Cada componente deve possuir um super componente (componente pai)

7. Casos de Uso do Projeto

7.1. Atores

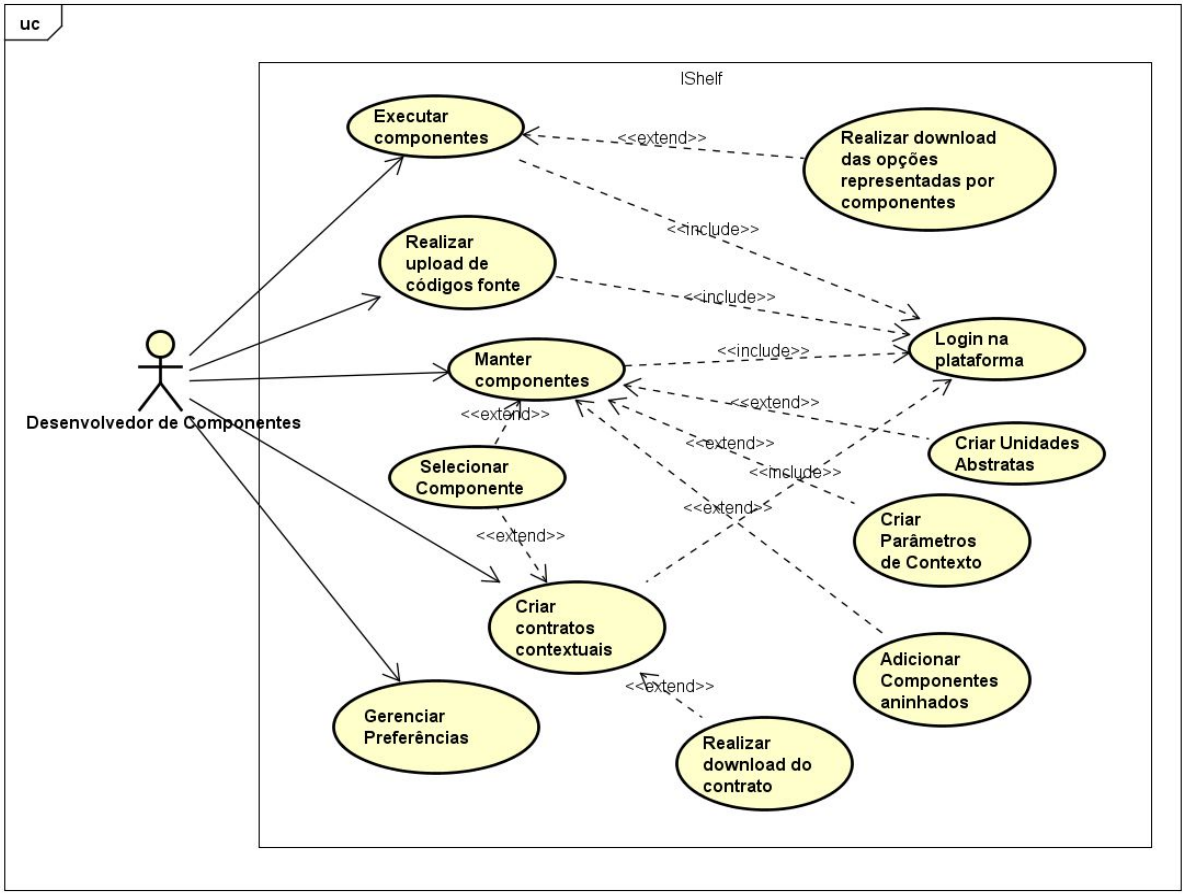
Identificador	Ator	Descrição
A01	Desenvolvedor	Desenvolvedor de componentes abstratos.
A02	Montador	Montador de aplicações..
A03	Mantenedor	Mantenedor das plataformas físicas de execução.

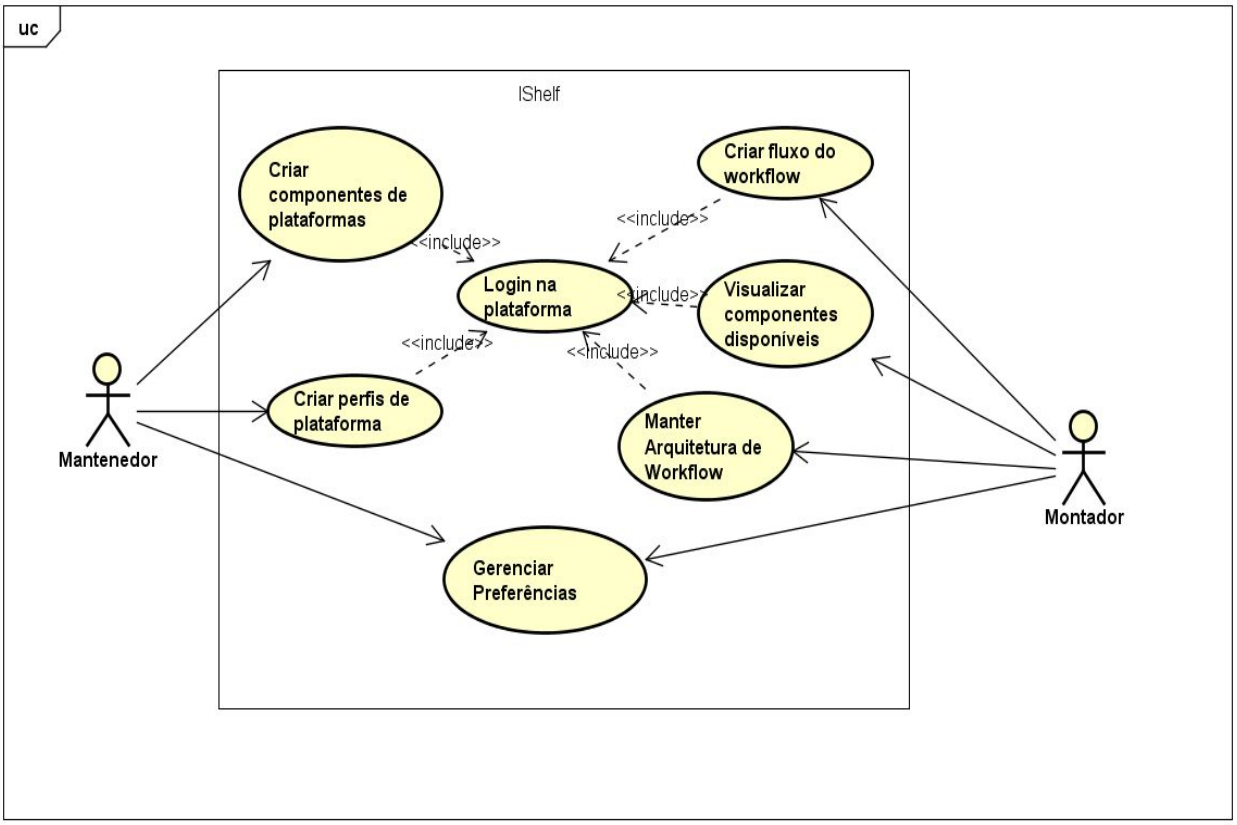
7.2. Casos de Uso

Identificador	Caso de Uso	Descrição
UC01	Login na plataforma	Os usuários deveram realizar login para ter acesso ao sistema.
UC02	Manter componentes	O usuário poderá cadastrar e editar componentes.
UC03	Criar unidades abstratas	O usuário poderá adicionar unidades abstratas no componentes que está criando.
UC04	Criar parâmetros de contexto	O usuário poderá adicionar parâmetros de contexto no componentes que está criando.
UC05	Adicionar componentes aninhados	O usuário poderá adicionar outros componentes existentes no componentes que está criando.
UC06	Selecionar componentes	O usuário poderá selecionar um componente já existente para editá-lo ou adicioná-lo ao componente que está criando.

UC07	Criar contrato contextual	O usuário poderá criar uma instância de um determinado componente previamente selecionado pelo mesmo.
UC08	Realizar *download* do contrato contextual	Após a criação da instância, o usuário poderá salvar em um arquivo.
UC09	Realizar o *upload* do código-fonte	O usuário poderá associar a uma unidade concreta (que está associada a uma unidade abstrata) um arquivo contendo o código-fonte.
UC10	Executar o componente	A partir de uma instância de um componente (previamente selecionada ou a partir de um arquivo contendo a instância) o usuário poderá falar a resolução da instância e assim implantá-la em uma plataforma de execução.
UC11	Realizar o *download* das opções	Após a resolução da instância, o usuário poderá salvar o resultado em um arquivo.
UC12	Visualizar os componentes disponíveis	O sistema deve listar os componentes disponíveis organizados por espécies (tipos)
UC13	Criar componente de plataformas	O usuário poderá criar componentes de espécies específicas, com base no componente pai, neste caso, do tipo plataforma.
UC15	Criar Perfil de plataforma	O usuário poderá criar uma instância a partir do componente da plataforma especificada pelo mesmo.
UC15	Manter a arquitetura de *workflows*	O usuário poderá criar e editar a arquitetura dos componentes que se unem para formar o *workflow*.
UC16	Criar fluxo do *workflow*	O usuário poderá criar e editar os fluxo de dados e procedimentos do *workflow*.
UC17	Gerenciar Preferências	O usuário poderá editar os valores que já foram padronizados segundo suas preferências
UC18	Criar perfis de plataforma	O usuário poderá criar instâncias componentes do tipo plataforma a partir do seu parque de recursos.

7.3. Diagrama de Casos de Uso




uc
IShelf
Criar componentes de plataformas
Criar fluxo do workflow
<<include>>
<<include>>
Login na plataforma
<<include>>
Visualizar componentes disponíveis
<<include>>
<<include>>
Criar perfis de plataforma
Manter Arquitetura de Workflow
Mantenedor
Montador
Gerenciar Preferências
powered by Astah

APÊNDICE B – DOCUMENTO DE ARQUITETURA

UNIVERSIDADE
FEDERAL DO CEARÁ

Documento de Arquitetura de *Software*
Interactive *Shelf* (I*Shelf*)

Histórico de Revisões

Data	Versão	Descrição	Autor
22/05/2017	0.1	Elaboração do Documento	Alysson Gomes
04/06/2017	0.2	Edição da visão de pacotes	Alysson Gomes
01/11/2017	0.3	Edição da visão de implementação	Alysson Gomes

1. Introdução

1.1. Finalidade

Este documento apresenta a arquitetura proposta para a solução *web* de apoio ao desenvolvimento de aplicações na plataforma de computação de alto desempenho HPC-*Shelf*. O objetivo deste documento é capturar e comunicar as decisões arquiteturais significativas que foram tomadas em relação ao sistema.

1.2. Escopo

Este documento define a arquitetura da solução *web* de apoio ao desenvolvimento de aplicações na plataforma HPC-*Shelf*.

1.3. Definições, Acrônimos e Abreviações

High Performance Computing	HPC
Requisito Arquitetural	RA
Unified Modeling Language	UML
Web services	WS

1.4. Visão Geral

Este documento está organizado em tópicos relacionados às diferentes visões arquiteturais abordadas.

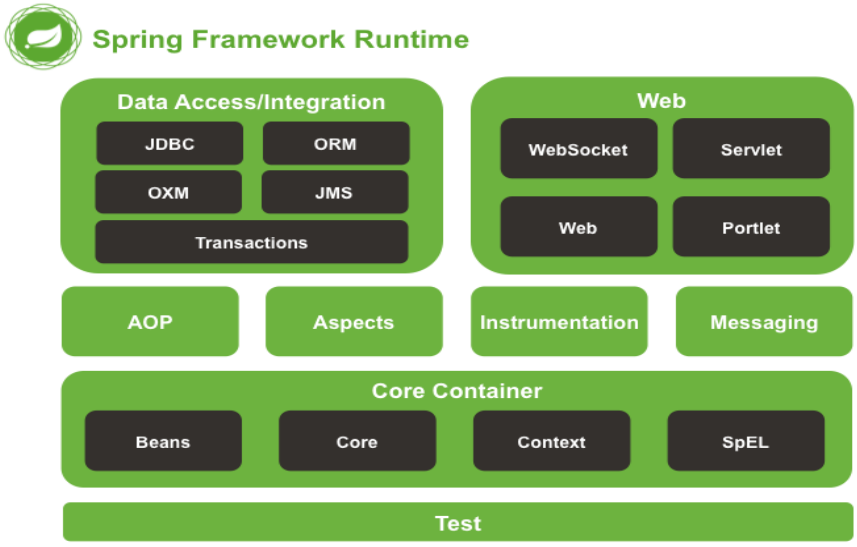
2. Arquitetura da Aplicação

Esse documento apresenta a arquitetura como uma série de visualizações/visões: visão lógica, visão de processos e visão de execução. Essas visualizações são apresentadas utilizando-se UML.

2.1. Representação arquitetural

O sistema foi desenvolvido tendo como base a arquitetura MVC adotada pelo Spring *Framework*. A Figura 1 representa a estrutura do *framework*.

Figura 1 –Versão básica do Spring *Framework*



Fonte:http://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/html/overview.html#overviewcore-containe

2.2. Objetivos e Restrições da Arquitetura

A arquitetura proposta tem como objetivo separar os interesses de cada camada da aplicação, disponibilizar um sistema funcional e com potencial de adaptabilidade a mudanças.

2.3. Critérios da Avaliação Arquitetural

Os critérios utilizados para a seleção da solução arquitetural foram:

- Modularidade;
- Manutenibilidade;

3. Metas e Restrições da Arquitetura

Para a execução do sistema será requisitado a instalação do(s) seguinte(s) software(s):

- Mozilla Firefox v. 38 ou superior / Google Chrome v.43 ou superior;

Caso algum *software* indicado não seja instalado, a aplicação pode não funcionar como esperado, ou não funcionar.

Existem alguns importantes requisitos e restrições do sistema que possuem uma influência significativa na arquitetura: São elas:

- Todas as funções devem estar disponíveis através dos navegadores mais populares;
- O sistema deve possuir diferentes visualizações para diferentes níveis hierárquicos.

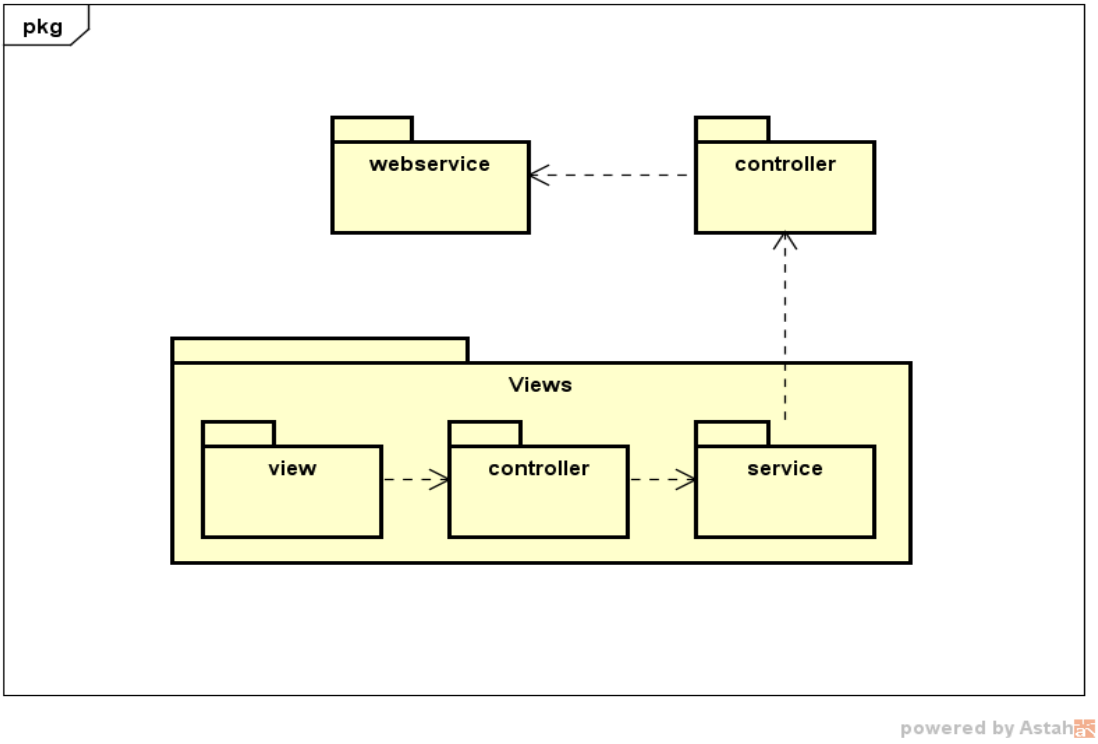
4. Visualização Lógica

Esta seção apresenta diferentes tipos de visões lógicas, com níveis de abstrações distintas, para que se possa obter uma visibilidade de comunicação entre pacotes e classes.

4.1. Visão de pacotes

A visão lógica da solução é composta por 4 diretórios principais:

Figura 2 - Visão de pacotes



Fonte: Próprio Autor

- ***Webservice***

Neste pacote está contido as classes de mapeamento com o *web service* da plataforma HPC-*Shelf*. Através do arquivo de descrição das funcionalidades do *web service* (arquivo WSDL), foi utilizada a ferramenta Jax-WS (KALIN, 2013), contido no pacote de bibliotecas padrão do Java, para criar as classes que fazem a comunicação direta com o *web service*.

- ***Controller***

Neste diretório foi implementada as classes que são nossos controladores, esses são responsáveis por conhecer as *urls* da aplicação e que lógica aplicar a cada uma delas, bem como seu retorno. Além disso, são responsáveis por realizar o controle das seções dos usuários.

- ***Views***

Para garantir mais dinamismo na aplicação, parte do processamento da mesma ficará no lado do cliente (*client side*), e consequentemente, parte da lógica de negócio da aplicação nesta parte.

  - ***Service***

Neste pacote contém os *scripts* JavaScript que fazem a comunicação com o servidor da aplicação. Ex.: *ComponentService*. Todas as requisições a respeito de informações de componentes, contratos, etc, devem ser feitas através deste *script*.

  - ***Controller***

Neste pacote está implementado os *scripts* que contém a outra parte da lógica de negócio da aplicação, relativo ao controle dos componentes das interfaces.

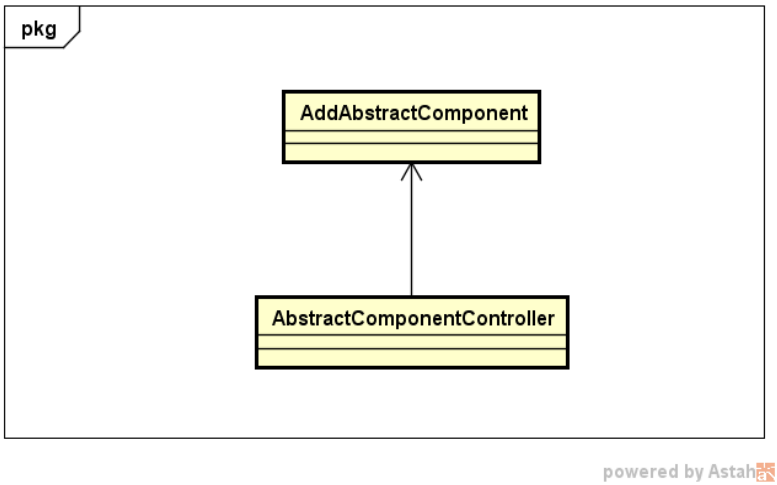
  - ***View***

Por fim, este pacote contém as interfaces de usuário, onde é exibido todo o conteúdo requisitado pelos *services*.

4.2. Visão de Classes

Apresenta como as classes se comunicam entre si na aplicação. Neste caso, é abordado o caso de um registro de componente, com respeito aos pacotes ***controller*** **e** ***web service***.

Figura 3 - Visão do relacionamento entre classes

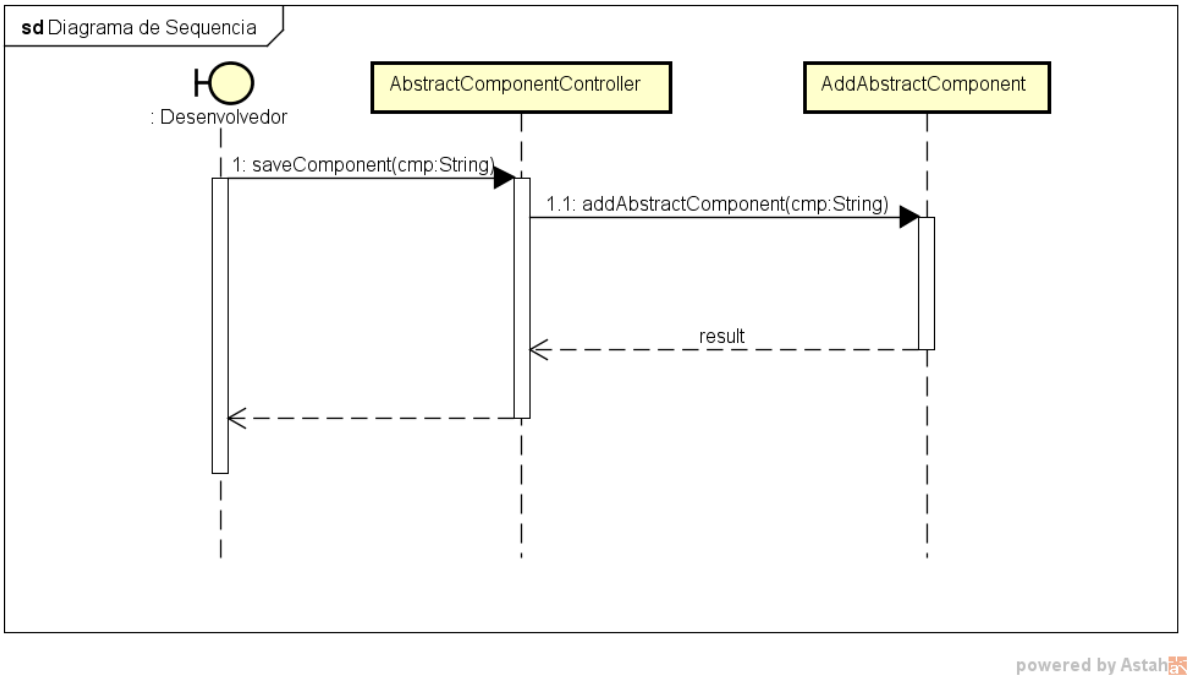


Fonte: Próprio Autor

5. Visão de Processos

Esta seção descreve a decomposição do sistema em processos, sendo evidenciado até então, o processo de registro de um novo componente no sistema. No qual o desenvolvedor logado no sistema faz uma requisição para criar um novo componente, o fluxo é representado na Figura 4.

Figura 4 - Visão de chamadas em sequência



Fonte: Próprio Autor

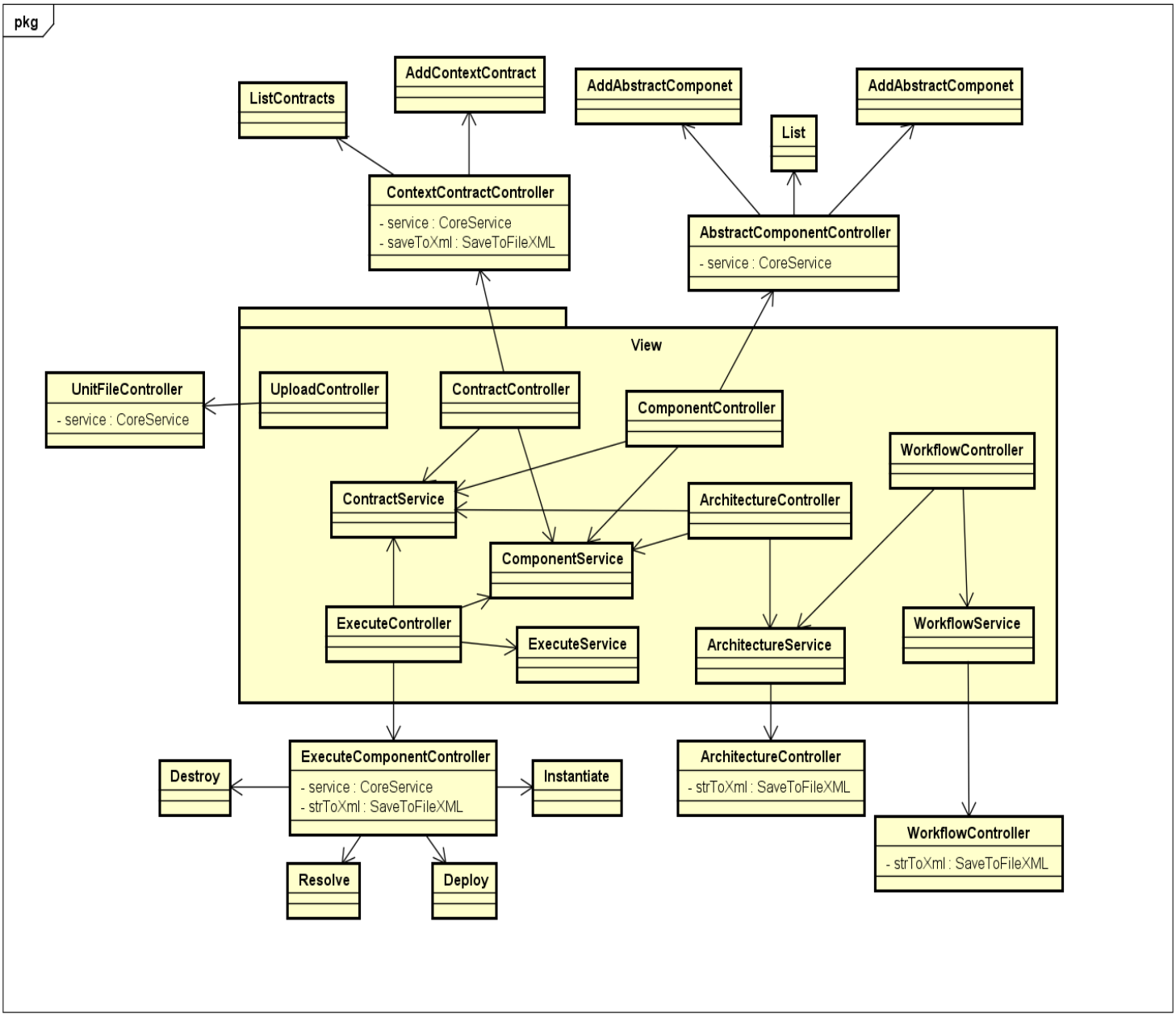
6. Visão de Implementação

Esta seção descreve a estrutura geral do modelo de implementação, as subdivisões arquiteturais e os componentes mais significativos da arquitetura.

6.2. Visão de Registro de Componentes Abstratos

Na Figura 5 estão representadas todas as entidades dos controladores do projeto, com seu respectivos atributos e relacionamentos.

Figura 5 - Visão Geral de Classes

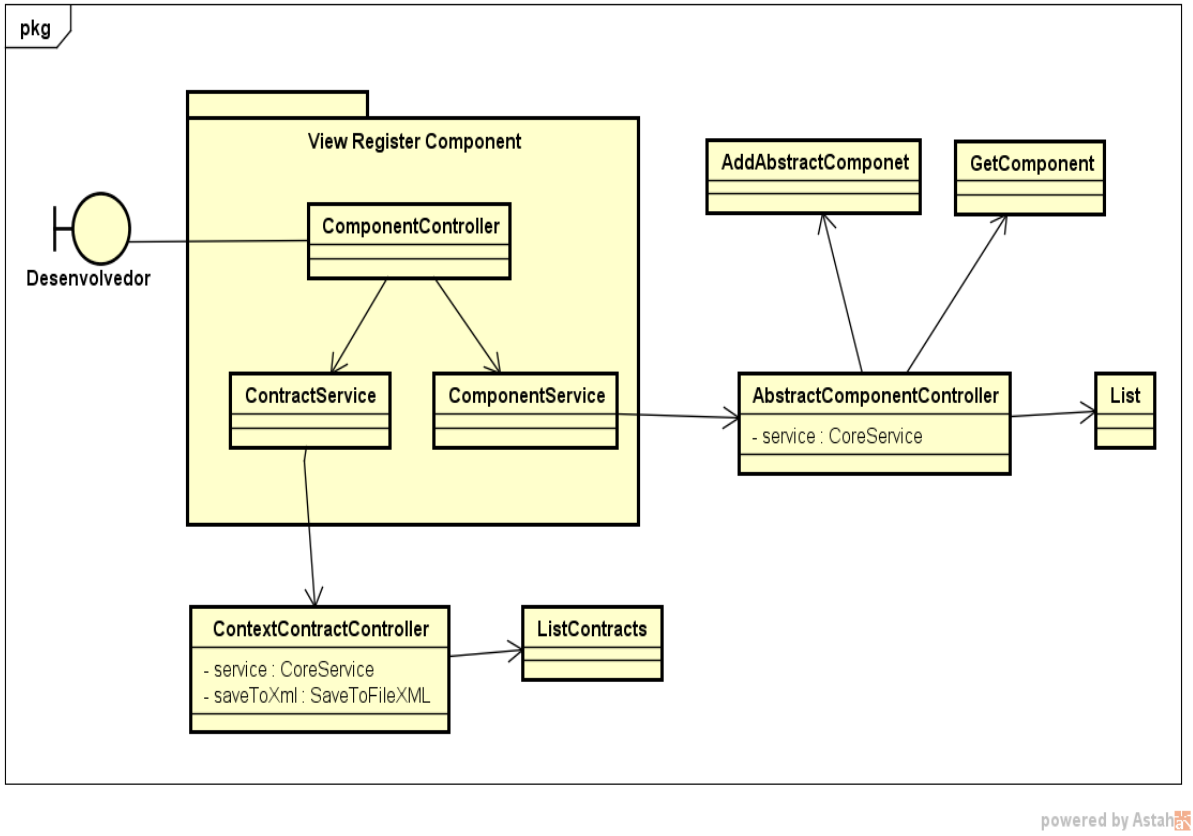


Fonte: Próprio Autor

6.3. Visão de Registro de Componente

Na Figura 6 são apresentadas as entidades que fazem parte do processo de criação de componentes abstratos.

Figura 6 - Classes do processo de registro de componentes

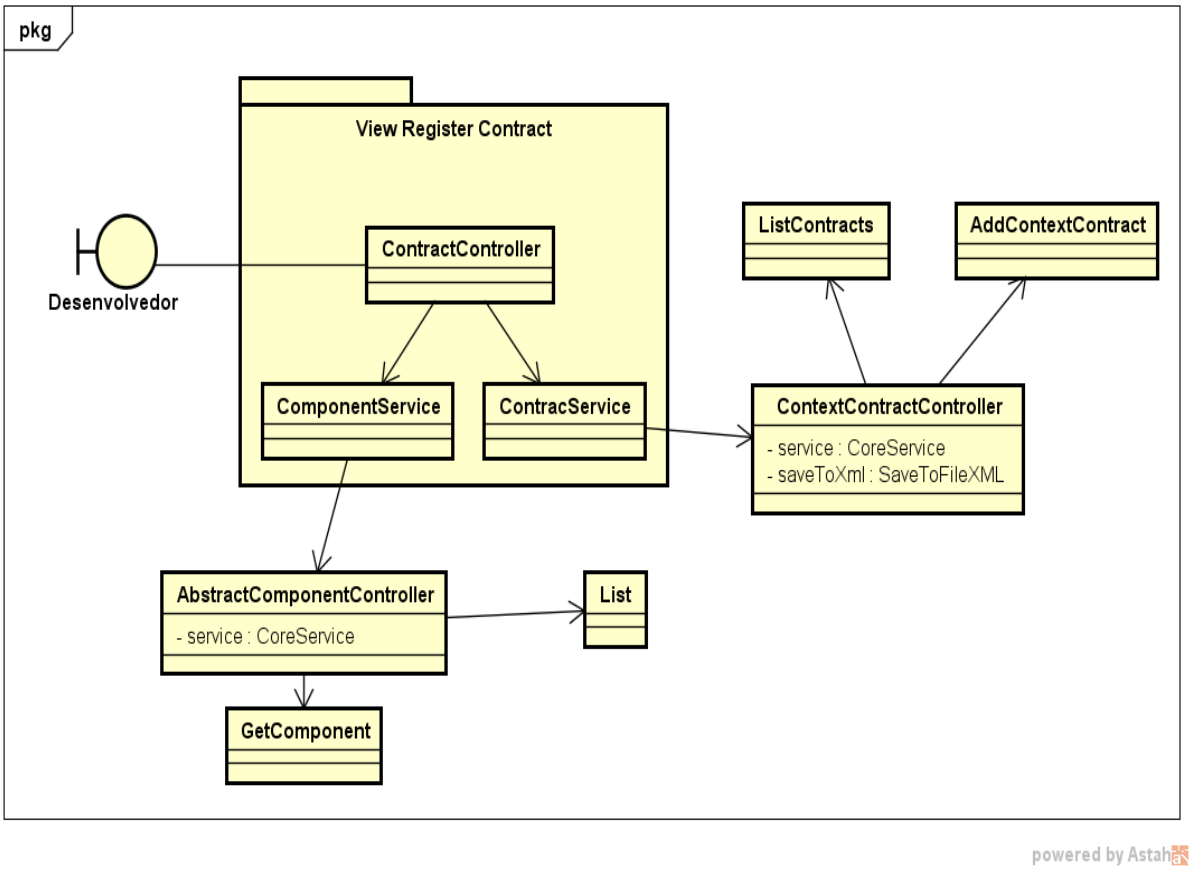


Fonte: Próprio Autor

6.4. Visão de Registro de Contrato Contextual

Na Figura 7 são apresentadas as entidades que fazem parte do processo de criação de contratos contextuais.

Figura 7 - Classes do processo de registro de contrato

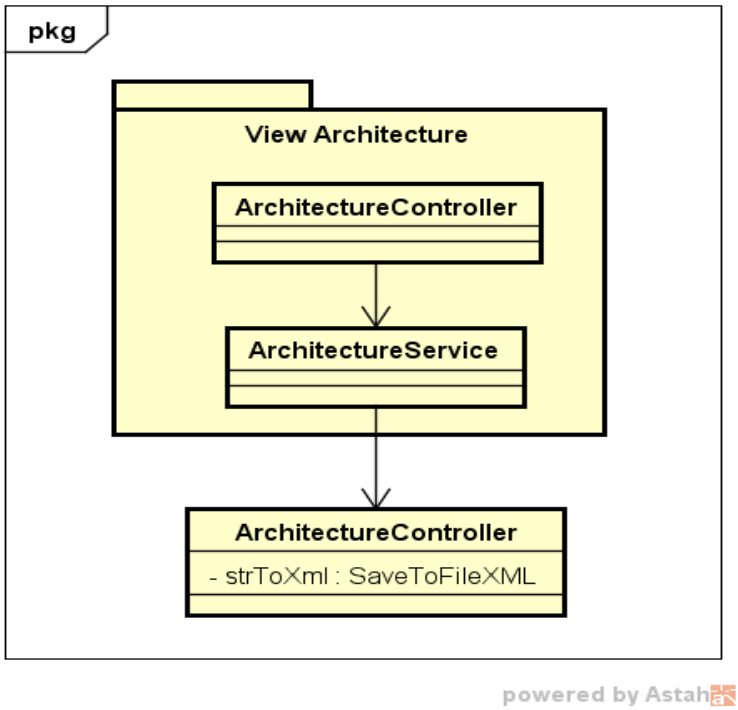


Fonte: Próprio Autor

6.5. Visão de Montagem da Arquitetura

Na Figura 8 são apresentadas as entidades que fazem parte do processo de montagem de arquiteturas de aplicações.

Figura 8 - Classes do processo de montagem de arquitetura

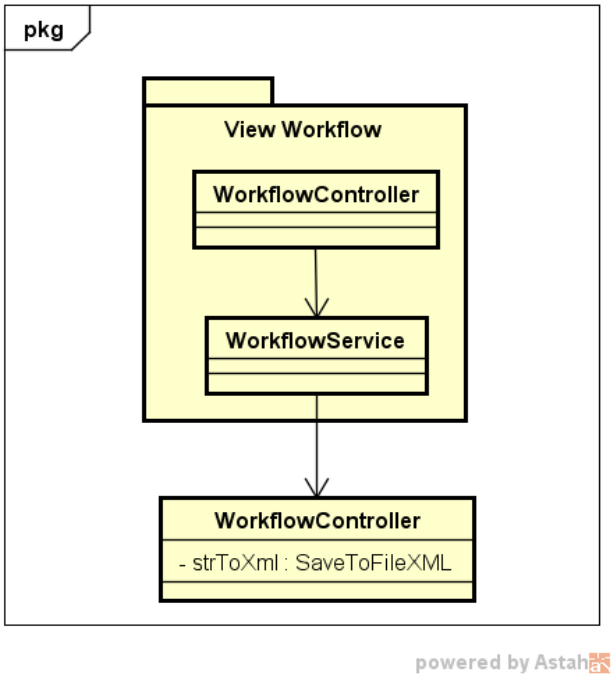


Fonte: Próprio Autor

6.5. Visão de Montagem da *Workflow*

Na Figura 9 são apresentadas as entidades que fazem parte do processo de criação de *workflow*.

Figura 9 - Classes do processo de montagem de arquitetura



Fonte: Próprio Autor

7. Qualidade

- **Modularidade:**
  - Como Meyer (1988) define, modularidade pode ser vista como a habilidade de conceber um sistema a partir de “peças” encaixáveis através de uma interface coerente, melhorando assim a reusabilidade e a extensibilidade. Por isso, os componentes estão separados por seus estereótipos, cada classe pertencente ao sistema tem um pacote específico que agrupa classes que possuem o mesmo estereótipo.

- **Manutenibilidade:**
  - Segundo a IEEE (1990), manutenibilidade é a facilidade com que um sistema ou componente de software pode ser modificado para se corrigir falhas, melhorar desempenho (ou outros atributos), ou ser adaptado a mudanças no ambiente. Por isso, através da separação de interesses no momento de se criar um componente ou classe, seguindo padrões de nomenclatura, buscando manter a coesão entre pacotes, facilita-se assim a manutenibilidade da arquitetura.

REFERÊNCIAS

WEISSMANN, Henrique Lobo. **Vire o jogo com Spring Framework**. São Paulo, SP: Casa do Código, 2012. p. 133.

KALIN, Martin. **Java Web Services: Up and Running: A Quick, Practical, and Thorough Introduction**. " O'Reilly Media, Inc.", 2013.

IEEE Std. 610.12-1990. **IEEE Standards Collection: Software Engineering**. IEEE, 1993.

MEYER, Bertrand. **Object-oriented software construction**. New York: Prentice hall, 1988.

APÊNDICE C – RELATÓRIO DE MEDIÇÃO

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Relatório de Medição de Qualidade

Responsável: Alysson Gomes de Sousa

Quixadá - CE
Setembro/2017

ÍNDICE

GLOSSÁRIO

Siglas	Definição
UFC	Universidade Federal do Ceará
IShelf	*Interative Shelf*

HISTÓRICO DE REVISÕES

Data	Versão	Descrição	Responsável
01/09/2017	1.0	Criação do documento.	Alysson Gomes de Sousa
05/09/2017	1.1	Inclusão de novos cenários	Alysson Gomes de Sousa
09/10/2017	1.2	Mudança nos procedimentos de avaliação	Alysson Gomes de Sousa

1. INTRODUÇÃO

O público alvo deste documento consiste nos leitores deste trabalho de pesquisa além do próprio autor do trabalho.

1.1. Descrição do produto a ser avaliado

A aplicação denominada *Interative Shelf* (I*Shelf*) é um ambiente gráfico que dá suporte a criação de componentes e aplicações para plataforma de computação de alto desempenho HPC-*Shelf*. A plataforma utiliza a abstração de componente para criar aplicações distribuídas de alto desempenho, esses componentes possuem atributos e relacionamentos com outros componentes, além disso, os componentes são orquestrados por meio de um *workflow* que irá controlar a aplicação como um todo. O I*Shelf* fornece elementos gráficos correspondentes às abstrações utilizadas na plataforma, assim, ele permite que os componentes sejam modelados e os *workflows* sejam desenvolvidos de uma forma intuitiva e interativa.

1.2. Objetivos da avaliação

Objetivo desta avaliação é validar a usabilidade dos produto citado, buscando identificar possíveis necessidades de usuários, problemas de interação, avaliando o quanto a interface afeta a forma que os usuários realizam as principais funcionalidades solicitadas para serem realizadas.

A avaliação será realizada conforme as funcionalidades da aplicação escolhida, verificando se as mesmas atendem as necessidades dos usuários e se a maneira como é realizada fica clara e de fácil compreensão. Avaliando também a qualidade da interação usuário-sistema, se o mesmo é fácil de usar, aprender.

Analisar	Aplicação I*Shelf*
Para o propósito de	Avaliar a realização das tarefas escolhidas.
Com respeito a	Usabilidade.
Do ponto de vista	De um usuário com experiência média ou baixa em aplicações desta natureza
No contexto de	da Universidade.

2. MÉTODO

Nesta Seção apresentaremos o método e as métricas que iremos utilizar para avaliar a ferramenta.

2.1. Ambiente de avaliação

Tarefas selecionadas para os participantes da pesquisa executarem:

- Criar um novo componente abstrato;
- Criar um contrato contextual;
- Executar um componente;
- Criar uma arquitetura;
- Criar um workflow.

As tarefas avaliadas foram executadas com os convidados que participaram da pesquisa na próprio universidade onde se encontra o pesquisador, no ambiente real, em um computador utilizando conexão com internet e navegadores Firefox e Google Chrome.

2.2. Procedimentos da Avaliação

Inicialmente os participantes foram recepcionados e receberam as informações a respeito da pesquisa com respeito ao perfil do pesquisador (graduando do curso de Sistemas de Informação da UFC *Campus* Quixadá) e o objetivo da pesquisa (medir a usabilidade da aplicação informada). Em seguida, os voluntários foram solicitados a responder perguntas sobre seu perfil: idade, gênero, e experiência com o uso de ferramentas similares a esta avaliada, disponível no *link* https://goo.gl/forms/bX3JxlAHJaFg9K6l1.

Após a apresentação, os participantes receberam os detalhes do que foi medido, para que os mesmo não tivessem qualquer tipo de preocupação com o tempo de realização das tarefas. Em seguida foi apresentados os dispositivos no qual foram realizadas as tarefas, informando que as tarefas solicitadas foram registradas e utilizadas como material para obtenção do resultado final da pesquisa. Por fim, na etapa onde os usuários fizeram as tarefas, foi utilizado uma ferramenta para automatizar a coleta de dados, sendo ela o *Camtasia Studio.* O *Camtasia* Studio se trata de uma ferramenta de criação de vídeo, que permite que o usuário crie vídeos, como tutoriais, através da gravação da tela. No contexto do nosso trabalho será gravada a tela em cada tarefa separadamente.

Vale ressaltar, que o usuários voluntários foram previamente informados do uso dessas ferramentas, e que nenhum dado do usuário como email, ou senha, foram utilizados durante a execução das tarefas.

2.3. Cenários

Tarefa 1: Você acabou de logar na aplicação, e quer modelar um novo componente abstrato para utilizar em uma aplicação. Você deve localizar na aplicação a funcionalidade de criação de novos componentes, modelar o novo componente e registrá-lo.

Tarefa 2: Você está logado e acabou de criar um componente abstrato, agora você deseja criar uma nova instância do componente. Você deve localizar na aplicação a funcionalidade de criação de contrato contextual, especificar o contrato e salvá-lo.

Tarefa 3: Você ainda está logado e terminou de especificar o contrato contextual, você deseja executar o componente para testá-lo. Você deve localizar na aplicação a funcionalidade de execução de componentes, e executar o componente.

Tarefa 4: Você acabou de logar na aplicação, e quer construir uma nova arquitetura para uma aplicação. Você deve encontrar a funcionalidade de criação de arquitetura, modelar a arquitetura e realizar o *download* da arquitetura.

Tarefa 5: Você acabou de criar uma arquitetura para uma aplicação e deseja criar um fluxo de trabalho (*workflow*) para esta arquitetura. Você deve localizar a funcionalidade de criação de *workflow*, estruturar o *workflow* e realizar o *download* do mesmo.

2.4. Participantes

Perfil dos Participantes: Graduados em cursos da área de computação, ou graduandos da mesma área que estejam cursando pelo menos o 4º (quarto) semestre e que já tenham tido contato com algum ferramenta de modelagem como Astah, BrModelo e afins. Apesar do público alvo da aplicação não ser exatamente o mesmo público descrito neste perfil, partiremos da seguinte premissa: se um estudante de computação com uma experiência média em ferramentas de modelagem consegue modelar e executar um componente, então um profissional da área terá a capacidade de fazer o mesmo.

2.5. Medidas de *Software*

2.5.1. Eficácia

Nome	**Descrição**	**Função de Medição**	**Método**	**Método de Coleta**	**Interpretação**
Completude da tarefa	Qual a proporção das tarefas são concluídas corretamente?	X = A/B A = Número de tarefas concluídas B = Número total de tarefas que o usuário tentou executar	Teste com usuários	Vídeo gravado durante a execução	Quanto mais próximo de 1 melhor.
Frequência de erros	Qual é a frequência de erros cometidos pelo usuário em relação a um valor-alvo?	X = A A = número de erros cometidos pelos usuário.	Teste com usuários	Vídeo gravado durante a execução	Quanto mais próximo de 0 melhor.

2.5.2. Eficiência

Nome	**Descrição**	**Função de Medição**	**Método**	**Método de Coleta**	**Interpretação**
Tempo da tarefas	Quanto tempo leva para o usuário completar uma tarefa?	X = tempo da tarefa Note: Você pode comparar com o tempo de um usuário especialista	Teste com usuário	Vídeo gravado durante a execução	Quanto mais próximo do tempo médio melhor.
Quantidade de *Clicks*	Quantos *clicks* foram necessários para completar a tarefa?	X = quantidade de *clicks*	Teste com usuário	Coleta na gravação do vídeo	Varia de acordo com a média de *clicks* para cada tarefa.

2.6. Procedimentos de Interpretação

Antes das avaliações, um usuário mais experiente com este tipo aplicação fez um *homerun* em todas as tarefas, onde foi realizada todas as medições para mantermos uma média de quais os resultados ideais no fim de cada métrica. Estes valores foram utilizados para fazermos a comparação com os dos voluntários da pesquisa.

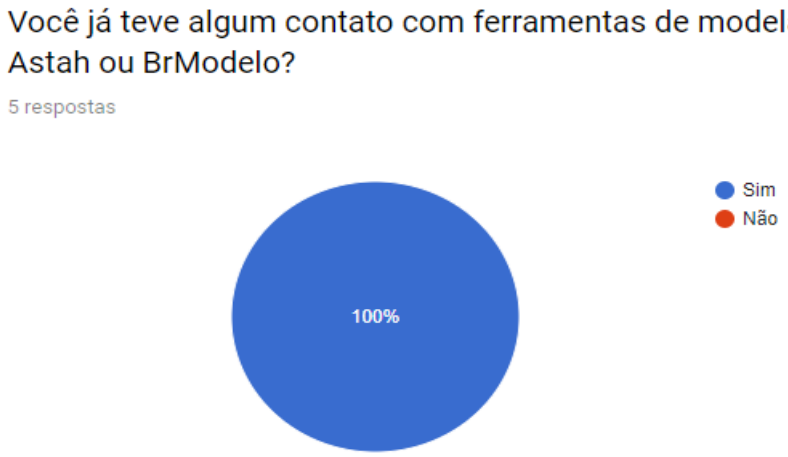
Para facilitar a compreensão e análise dos dados, foi utilizados gráficos para apresentação dos resultados obtidos através das medições.

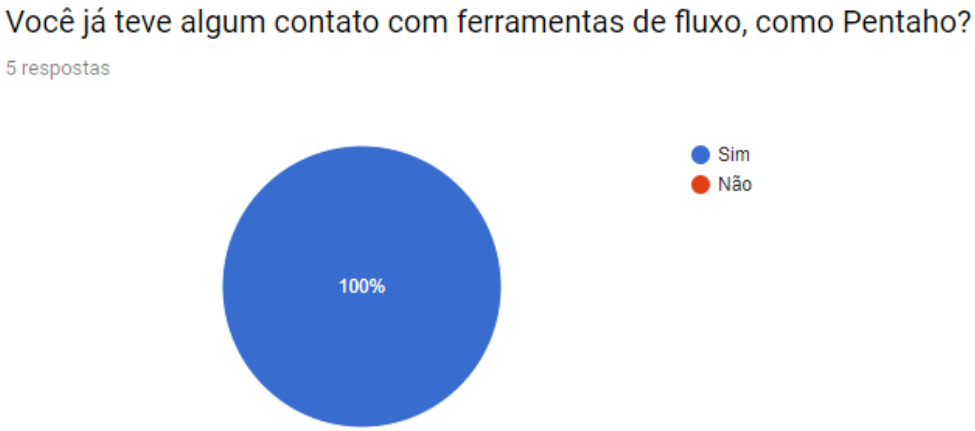
3. RESULTADOS

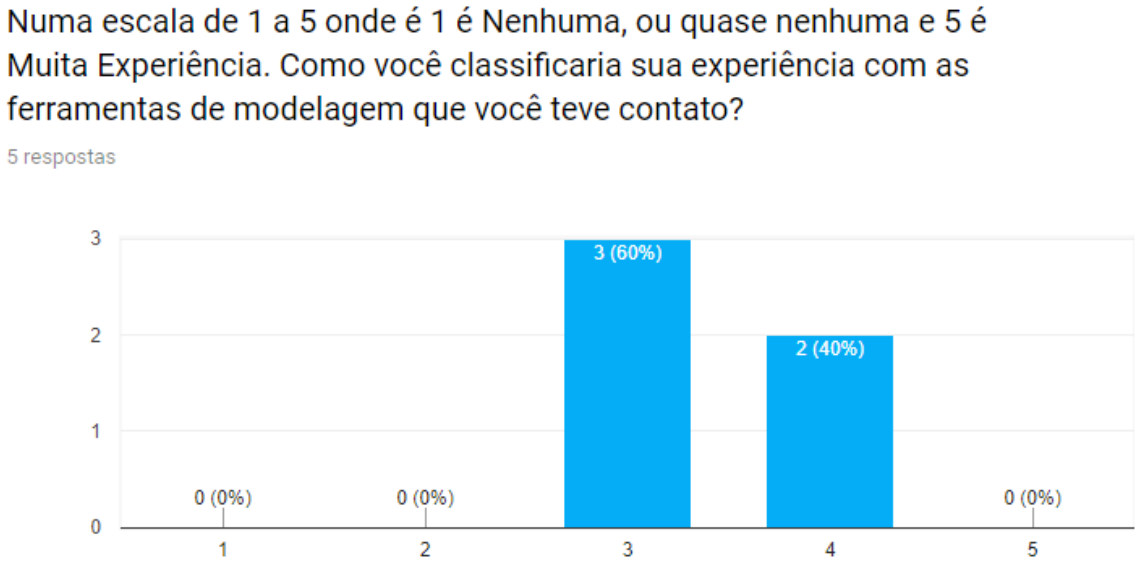
Nesta seção apresentamos os dados e as análises dos dados.

3.1. Perfil dos Participantes

Logo abaixo segue as informações dos participantes, como: contato com ferramentas de modelagem, contato com ferramentas de fluxo de execução e experiências com as ferramentas.







3.2. Resultados dos Usuários Voluntários por Tarefa

Segue os dados dos usuários quanto aos valores obtidos durante cada atividade.

Usuário 1					
Met/Ativ	Atividade 1	Atividade 2	Atividade 3	Atividade 4	Atividade 5
Clicks	174	60	8	23	76
Tempo	25:46	14:08	02:02	05:53	10:42
Erros	2	1	0	3	1

Usuário 2					
Met/Ativ	Atividade 1	Atividade 2	Atividade 3	Atividade 4	Atividade 5
Clicks	122	80	7	21	69
Tempo	19:34	17:35	1:49	02:52	07:09
Erros	1	2	0	0	2

Usuário 3					
Met/Ativ	Atividade 1	Atividade 2	Atividade 3	Atividade 4	Atividade 5
Clicks	131	66	7	18	96
Tempo	19:10	08:23	01:10	02:22	07:03
Erros	0	1	0	1	3

Usuário 4					
Met/Ativ	Atividade 1	Atividade 2	Atividade 3	Atividade 4	Atividade 5
Clicks	185	108	7	33	125
Tempo	25:27	12:47	01:14	03:32	09:23
Erros	3	2	1	1	6

Usuário 5					
Met/Ativ	Atividade 1	Atividade 2	Atividade 3	Atividade 4	Atividade 5
Clicks	137	77	7	21	65
Tempo	12:17	07:25	00:50	2:19	3:13
Erros	1	0	0	0	2

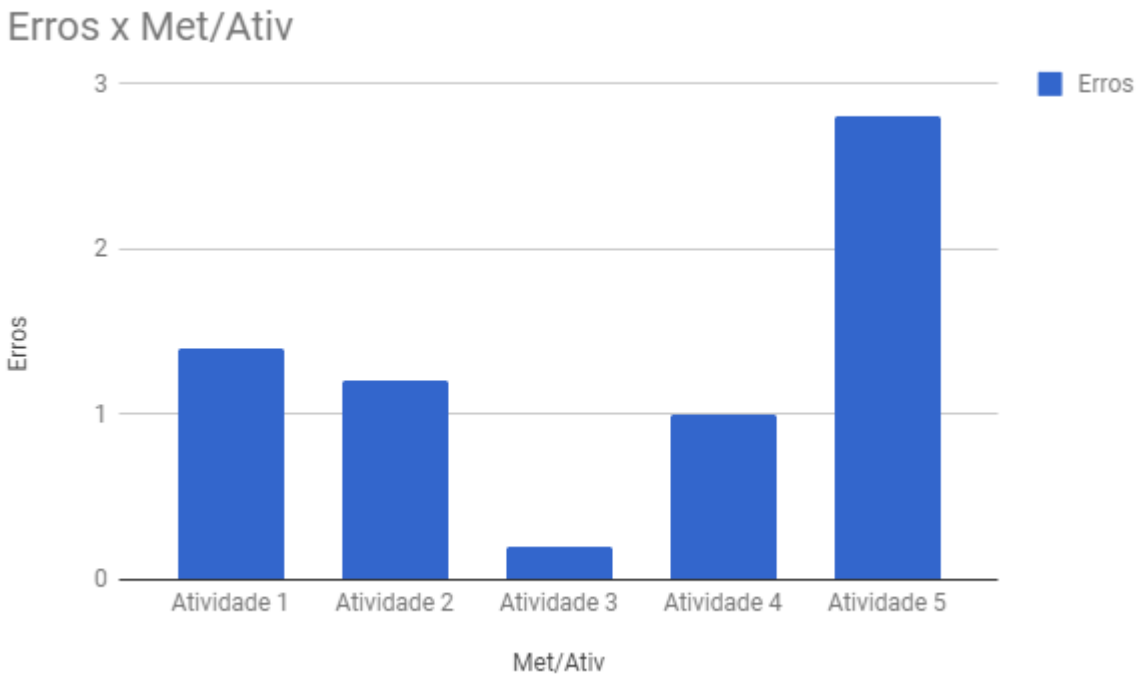
3.3. Média de Resultados das Tarefas

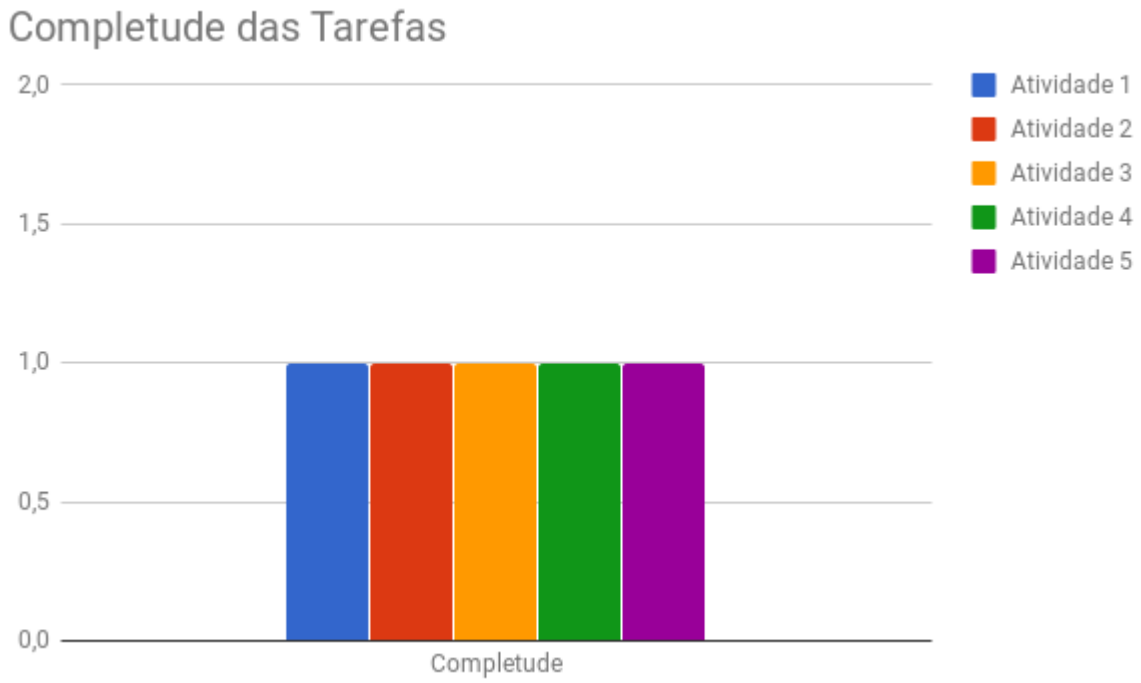
Realizamos um cálculo de médias por atividades para podermos fundamentar algumas conclusões. Como podemos observar nos dados abaixos a atividade na qual se levou mais tempo foi a Atividade 1, enquanto a Atividade 3 foi a realizada em menos tempo. Também podemos analogamente concluir que o mesmo vale quanto a quantidade de cliques.

Médias de Atividades					
Met/Ativ	Atividade 1	Atividade 2	Atividade 3	Atividade 4	Atividade 5
Clicks	149	78	7	23	86
Tempo	20:26	11:27	1:25	3:35	7:18
Erros	1.4	1.2	0.2	1	2.8

3.4. Eficácia

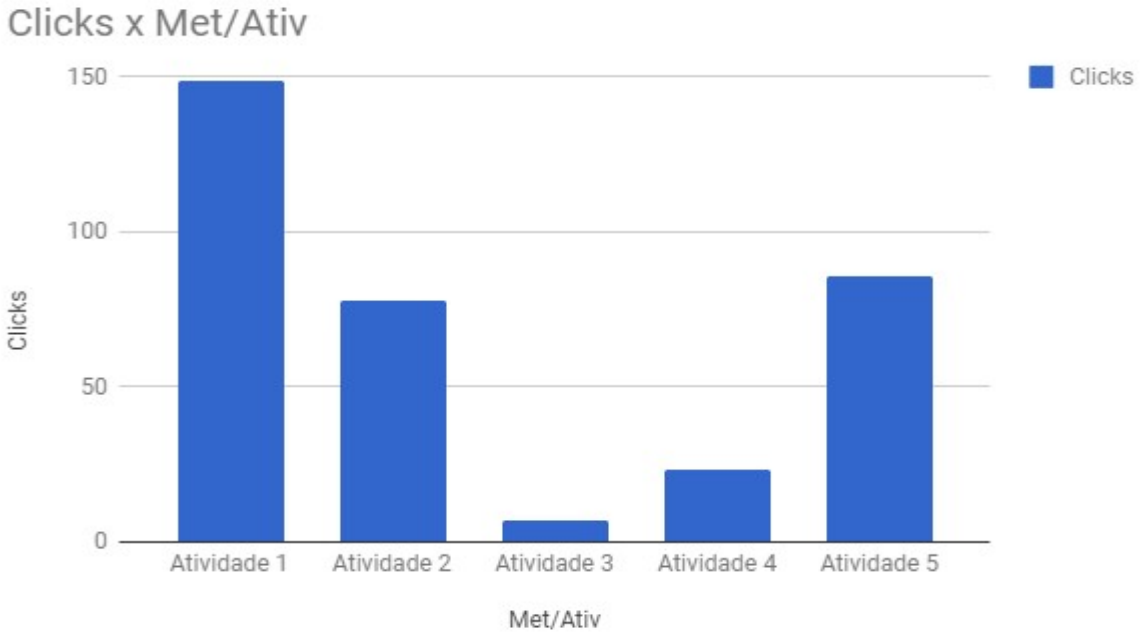
Nos gráficos abaixo podemos ver que todas as atividades foram completadas, no entanto, a atividade 5 possui a maior quantidade de erros.

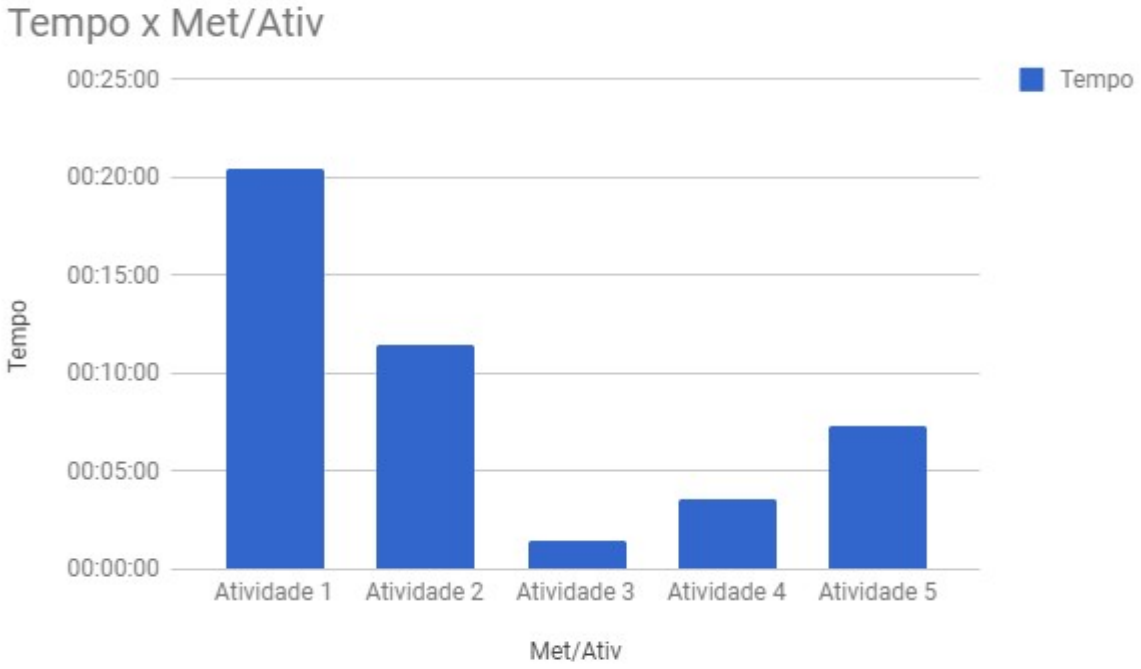


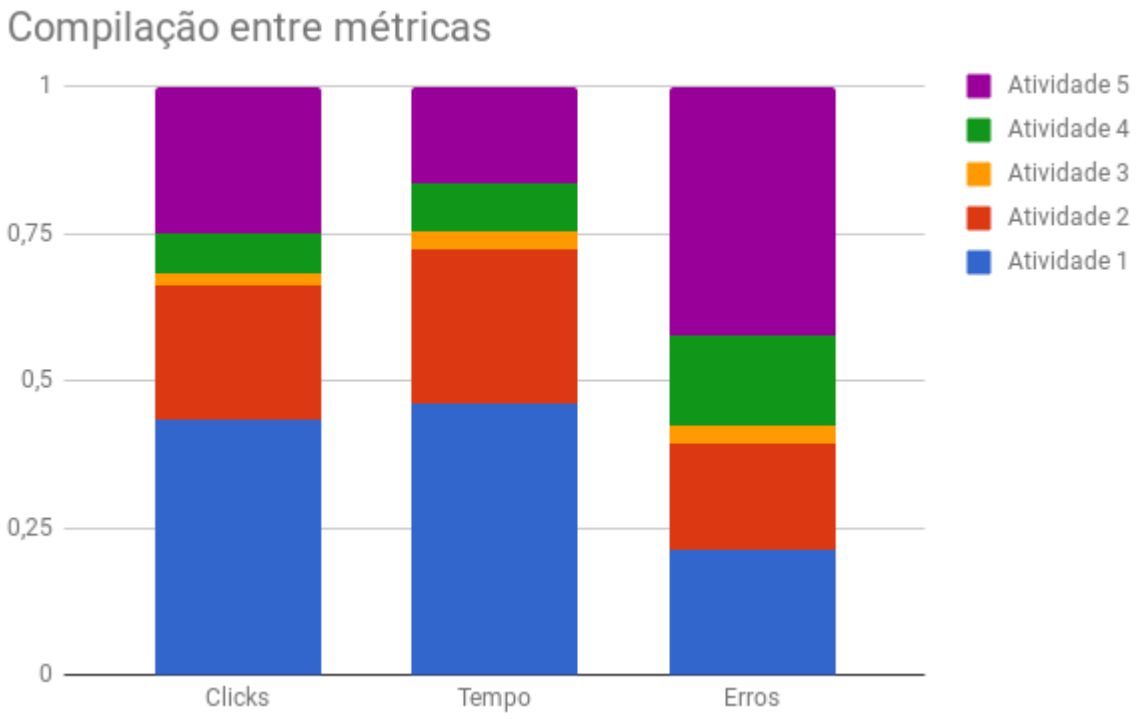


3.5. Eficiência

Nos gráficos abaixo podemos ver o esforço necessário para realizar cada atividade, claramente a atividade 1 se mostrou mais custosa, como mostra a quantidade de *clicks* e o tempo necessário para realizar a tarefa.







4. CONCLUSÕES

Com base nos dados levantados, descritos na Seção 3.2 e compilados nas Seções 3.3,3.4 e 3.5, chegamos às seguintes conclusões: (i) apesar da atividade 1 ter sido a mais custosa (tendo em vista o tempo e a quantidade de *clicks*), a atividade 5, se mostrou mais complexa devido a quantidade de erros, isso demonstra há um certa dificuldade trabalhar na interface de montagem de *workflow*, pois possui uma curva de aprendizado maior do que as demais telas, (ii) embora a modelagem de componentes não tenha sido muito complexa em relação a outras atividades, o custo de realizar as tarefas ainda é alto, e isso pôde ser averiguado nas gravações das interações dos usuários, o que nos leva a concluir que os elementos ainda não estão bem dispostos na interface, o que contribui para o aumento dos *clicks* e a quantidade de tempo.

REFERÊNCIAS

ISO/IEC 25000. Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE. v. 2005, 2005.

ISO/IEC 9126. Software Engineering – Product Quality – Part 1. 2001