UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE CIÊNCIAS APLICADAS A EDUCAÇÃO

DEPARTAMENTO DE CIÊNCIAS EXATAS

LICENCIATURA EM CIÊNCIA DA COMPUTAÇÃO

AVALIANDO O DESEMPENHO DE SISTEMAS ADAPTÁVEIS BASEADOS EM AOM EXPLORANDO O LOM

RIO TINTO - PB

2015

FERNANDO MATEUS DE OLIVEIRA

AVALIANDO O DESEMPENHO DE SISTEMAS ADAPTÁVEIS BASEADOS EM AOM EXPLORANDO O LOM

Monografia apresentada para obtenção do título de Licenciado à banca examinadora no Curso de Licenciatura em Ciência da Computação do Centro de Ciências Aplicadas e Educação (CCAE), Campus IV da Universidade Federal da Paraíba.
Orientadora: Prof. Drª. Ayla Dantas.

RIO TINTO - PB

2015

FERNANDO MATEUS DE OLIVEIRA

AVALIANDO O DESEMPENHO DE SISTEMAS ADAPTÁVEIS BASEADOS EM AOM EXPLORANDO O LOM

Trabalho de Conclusão de Curso submetido ao Curso de Licenciatura em Ciência da Computação da Universidade Federal da Paraíba, Campus IV, como parte dos requisitos necessários para obtenção do grau de LICENCIADO EM CIÊNCIA DA COMPUTAÇÃO.

Assinatura do autor:_____

APROVADO POR:

Orientador: Prof. DSc. Ayla Débora Dantas S. Rebouças

Universidade Federal da Paraíba – Campus IV

Prof. DSc. Juliana de Albuquerque Gonçalves Saraiva

Universidade Federal da Paraíba – Campus IV

Prof. MSc. Rodrigo de Almeida Vilar de Miranda

Universidade Federal da Paraíba – Campus IV

RIO TINTO - PB

2015

O48a Oliveira, Fernando Mateus de.
Avaliando o desempenho de sistemas adaptáveis baseados em AOM explorando o LOM. / Fernando Mateus de Oliveira. – Rio Tinto: [s.n.], 2015.
65f. : il.-

Orientador (a): Profa. Dra. Ayla Dantas.
Monografia (Graduação) – UFPB/CCAE.

1. Software - desenvolvimento. 2. Software - adaptabilidade. 3. Sistemas de informação.

UFPB/BS-CCAE CDU: 004.4(043.2)

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus, que permitiu minha chegada até aqui bem próximo da conclusão da graduação, não permitindo que eu caísse nos momento de fraqueza. Buscarei junto a ele sempre andar pelo caminho estreito.

Agradeço a painho (Nino) e a mainha (Cila) por me darem todo o suporte, incentivo e amor, não apenas durante o a graduação, mas desde toda a vida, não me deixando desistir dos meus objetivos. Sem vocês, talvez, eu não estivesse chegado até aqui.

Agradeço ao meu avô e padrinho Fernando (*in memoriam*), pelo exemplo de simplicidade e amor deixado para toda a família, e a minha amada avó e madrinha Eunice, que esteve sempre preocupada com minhas constantes viagens a Rio Tinto e buscando me confortar diante de todo cansaço.

Agradeço a minha irmã Ana Luíza, ao meu cunhado Wesley (Lelo) e ao meu já tão amado sobrinho(a) por todo apoio e por compreenderem minha ausência durante os encontros familiares.

Obrigado a Fabiana minha tão amada noiva, futura esposa e eterna namorada, que vem acreditando em mim desde que começamos a namorar ainda no tempo do ensino médio. Nunca renunciei os momentos de lazer sozinho, ela sempre esteve ao meu lado quando eu tinha que cumprir minhas atividades.

Aos pais de Fabiana, Lúcia e Geraldo, que sempre buscaram me fornecer algum suporte, mesmo que em sua simplicidade, fizeram a diferença para que eu pudesse chegar até aqui.

Aos meus amigos Alex Costa, José Raul, Maelso Bruno, Marcus Rafael e Sinval Vieira. Raul (Glu Glu) e Maelso (Piru...) por aceitarem dividir a moradia comigo em Rio Tinto, onde passamos bons momentos e muitas conquistas sempre vibrando juntos a cada uma delas, mesmo no calor insuportável. Rafael (Micôca) e Sinval (Banguelo) também por serem nossos vizinhos chatos, que gostavam de sujar e bagunçar nossa casa, mas com os quais dividi vários momentos de aprendizado principalmente em programação, sempre cheios de questionamentos e desafios, aprendemos a programar juntos. E a Alex (Patá) que integrou a nossa "gangue" após certo tempo, mas que passou a ser meu parceiro de trabalho, minha dupla nas disciplinas e também de gasolina. Foram muito bons esses momentos que vivemos durante a graduação.

Agradeço bastante a minha orientadora Ayla Dantas, que me ensinou POO, Pesquisa Aplicada, e mais do que isso durante o TCC foi uma professora ainda mais multidisciplinar, me ensinando estatística, corrigindo alguns erros de português, etc. Ayla é uma das pessoas mais inteligentes e mais humildes que já conheci, tenho muita admiração por ela. Sem ela, esse trabalhado não seria possível de ser realizado.

Ao professor Rodrigo Vilar por todo ensino sobre sistemas adaptáveis e AOM, e por me dar a liberdade de colaborar no desenvolvimento do LOM. Além disso, Vilar foi o cara que me ensinou a refatorar e a aplicar os padrões de projeto, me mostrando que sempre há uma forma melhor de escrever meus códigos. Sem ele, esse trabalho também não seria possível.

Obrigado a professora Pasqueline Dantas, que mesmo no início do curso acreditou no meu potencial e me cedeu a oportunidade em trabalhar no PIBID, meu primeiro contato com a pesquisa científica. Foi muito bom tê-la como orientadora.

Ao professor Rodrigo Rebouças que foi o cara que me ensinou a programar, sempre nos entusiasmando e não deixando nos abater durante os problemas mais complexos que íamos tentando resolver.

Aos demais familiares e colegas, que sempre torceram por mim. Não conseguirei citar todos, mas ainda assim devo destacar alguns deles: Fernandinho, Penha, Filipe e Luís Fernando; Marli, Araújo, Dalila e Dalete; Betânia, Cacá, Juliane e Juninho; Fabinho, Fátima, Flaviana e Fabiana; Flávio e Emily; Mana, Eugênio, Leandro, Léo e Kátia; Flávia, Tadeu, João Bernardo, Filipe, Fabíola e Fernanda; Carlos e Alexa; Lailson e Rayssa; Irineu e Tuane; André Bolívia, Wescley e Nayanne, parceiros das caronas; Ana Carla, Anna Clara, Hélder Vieira, Carlos Filho, Adriano, Raphaell Pinheiro, Yuri Iarley, Erick Rodrigo, Ivo Sandro, Aline Morais, Ednaldo Onofre, Bruno Santos, Jonas Ieno e Diogo Medeiros, do projeto SIM; André Meireles e Neudson Barros, da Pacto Tecnologia; Lettiery, Adilson, Rodolpho, Paula, Mara, Péricles e Flávio, da Ellos Connecti. Tenho muita sorte em ter vocês como amigos!

Muito obrigado a todos! Deus abençoe!

RESUMO

Sistemas adaptáveis baseados em AOM (*Adaptive Object Model*) podem ser bastante flexíveis, pois é possível mudar facilmente o seu comportamento devido ao nível de abstração de tais sistemas, já que as entidades, os relacionamentos e as regras de negócios são representados através de metamodelos. Esses metamodelos devem ser persistidos junto com os metadados da aplicação em banco de dados, por exemplo. No entanto, desenvolver um sistema AOM envolve um certo nível de complexidade que por outro lado pode ser simplificado através do reuso de software, utilizando ferramentas como o arcabouço LOM. Além da complexidade, sempre vêm sendo questionado o desempenho de tais sistemas, isso considerando as constantes manipulações dos metamodelos e metadados, o que implica no constante acesso das aplicações aos bancos de dados. Portanto, como problema de pesquisa, se buscou investigar nesse trabalho o impacto de desempenho ao transformar um sistema orientado a objetos em um sistema adaptável utilizando o estilo arquitetural AOM no seu desenvolvimento. Uma das colaborações relevantes desse trabalho foi o auxílio no desenvolvimento do arcabouço LOM, que é uma alternativa para dar suporte ao desenvolvimento de sistemas AOM. O sistema para gerência de exercícios denominado EducService foi implementado em três diferentes versões: uma versão orientada a objetos e a outras com auxílio do LOM com variações na camada de persistência. A avaliação de desempenho considerou as três versões do EducService, comparando os resultados obtidos através da execução de cenários pré-definidos e aplicando alguns cálculos da estatística nos dados obtidos para melhor analisá-los. Como era esperado, a implementação orientada a objetos, e sem adaptabilidade, apresentou melhor desempenho em todos os cenários. Já entre as variações de implementação baseadas no LOM, a versão que realizou persistência no banco de dados orientado a grafo apresentou melhor desempenho que a versão relacional em apenas um dos seis cenários considerados.

Palavras chave: Adaptabilidade de Software, *Adaptive Object Model*, Metadados, Arcabouço, Avaliação de desempenho.

ABSTRACT

Adaptive systems based on AOM can be quite flexible, because you can easily change their behavior due to the level of abstraction of such systems, as entities, relationships and business rules are represented by meta-models. These metamodels should be persisted together with the application metadata in the database, for example. However, developing an AOM system involves a certain level of complexity, but it can be simplified through software reuse, using tools such as the LOM framework, for example. Besides their complexity, another aspect that is always questioned is the performance of such systems, as they demand the constant manipulation of metadata, which implies constant access to databases. So, the research problem in which we focused was the investigation of the performance impact of transforming an object-oriented system, with little focus on adaptability, in an adaptive system using the AOM architectural style in its development. One of the important contributions of this study was the development of part of the LOM framework, which is an alternative to aid the development of AOM systems. The system to manage exercises called EducService was developed in three different versions: a pure object-oriented version and two other ones with the help of LOM in two versions of the persistence layer. The performance evaluation considered three versions of EducService, comparing the results obtained through the execution of predefined scenarios and applying some statistical calculations on the obtained data in order to better analyze them.

Keywords: Adaptable software; Adaptive Object Model; Metadata; Frameworks; Performance Evaluation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AOM	*Adaptive Object Model*
API	*Application Programming Interface*
DAO	*Data Access Object*
JDBC	*Java Database Connectivity*
JSON	*JavaScript Object Notation*
JVM	*Java Virtual Machine*
LOM	*Living Object Model*
NOSQL	*Not Only SQL*
OO	*Orientação a Objetos*
PROLICEN	Programa de Licenciatura
SGBD	*Sistema Gerenciador de Banco de Dados*
SQL	*Structured Query Language*
XML	*Extensible Markup Language*

SUMÁRIO

1. INTRODUÇÃO

Segundo Buschmann et al. (1996), um sistema adaptável é aquele capaz de apoiar novas versões de sistemas operacionais, plataformas de interface do usuário ou componentes de terceiros e bibliotecas. Ele deve estar projetado para atender às mudanças, diante de novos requisitos que podem surgir, ou às adições de novas extensões. Com isso, projetar uma arquitetura de software para atender a demandas por novos requisitos do usuário, se reflete em diminuir o esforço para se atender a tais demandas. Neste trabalho considera-se adaptabilidade como relacionado a manutenibilidade, embora seja frequente se usar o termo adaptabilidade quando se quer referenciar sistemas que de forma simples são capazes de sofrer mudanças até mesmo em tempo de execução.

Uma das abordagens entre as capazes de proporcionar adaptabilidade a um software, a qual já vem sendo estudada durante alguns anos, é o uso do estilo arquitetural *Adaptive Object Model* (AOM). Através dele, é possível criar entidades, propriedades, relacionamentos e regras de negócio através de metadados (Yoder et al., 1998). Sistemas desenvolvidos usando esse estilo arquitetural podem ser tão adaptáveis chegando ao ponto de serem alterados, inclusive, em tempo de execução (do inglês*, runtime*). No entanto, para entender AOM, é preciso entender um conjunto de padrões de projeto. AOM é caracterizado na literatura como uma arquitetura reflexiva ou meta arquitetura, pois especifica os componentes em nível de metamodelos. Segundo Welicki e colaboradores (2007), AOM também pode ser considerado como uma linguagem de padrões devido à existência desse conjunto de padrões menores que o compõem. Neste trabalho de conclusão de curso ele será tratado como um estilo arquitetural.

Contudo, ao se pensar em sistemas totalmente adaptáveis baseados em metamodelos, interpretados em tempo de execução e que podem sofrer mudanças de requisitos também em tempo de execução, e que é algo que pode ser proporcionado com AOM, sempre se questiona o desempenho de tais sistemas. Segundo Yoder e Foote (1998), quanto mais se proporciona dinamicidade a um sistema, pior pode ser o seu desempenho. Considerando a persistência de seus dados, Matsumoto e Guerra (2012) afirmaram que, devido aos sistemas baseados em AOM necessitarem realizar a persistência do seu modelo através de metadados e também considerando a natureza evolutiva desses dados, bancos de dados relacionais talvez não sejam a melhor alternativa de persistência para tais sistemas.

Sendo assim, como comentado na literatura, sistemas adaptáveis apresentam em geral um desempenho ruim e bancos de dados relacionais não são a melhor alternativa de persistência para eles. No entanto, ainda não se sabe de forma quantitativa a dimensão de tal perda e nem foi feito um estudo mais detalhado sobre a persistência de sistemas AOM usando o modelo relacional comparado com outros modelos. Com isso, questiona-se qual o impacto de desempenho que um sistema sofrerá quando no seu desenvolvimento se optar por utilizar o estilo arquitetural AOM, em vez de uma abordagem orientada a objetos sem AOM. Como forma para minimizar o impacto de desempenho ao utilizar sistemas adaptáveis, uma alternativa ao banco de dados relacional é o uso de um banco de dados *Not Only SQL* (NOSQL) orientado a grafos.

Este trabalho de conclusão de curso tem como objetivo **avaliar o impacto de desempenho de tornar um sistema adaptável com AOM**.

Um estudo caracterizado como um *quasi-experiment* foi desenvolvido e se utilizou do sistema EducService, que foi implementado em três diferentes versões. A primeira foi uma versão totalmente orientada a objetos (OO), sem AOM, e que utilizou na camada de persistência uma implementação para o banco de dados relacional MySQL. A segunda e a terceira versão foram variações de implementações AOM através do arcabouço *Living Object Model* (LOM). A segunda versão recebeu uma implementação na camada de persistência para que os metadados e o metamodelo fossem persistidos no banco de dados MySQL e a terceira versão foi bastante parecida com a segunda, havendo apenas a variação da camada de persistência, que explorou um banco de dados orientado a grafos, o Neo4J.

Os sistemas de banco de dados MySQL e Neo4J foram escolhidos como alternativa de persistência pelo fato de terem sido as tecnologias mais utilizadas de banco de dados relacional e bancos de dados orientado a grafos, respectivamente, segundo o ranking DB-engines (2015). O Neo4J apresenta-se eficiente quando se têm dados altamente conectados (Neo4J, 2015) e se acredita que devido aos relacionamentos existentes entre todo o metamodelo e os metadados esse banco de dados seria uma opção mais eficiente que um banco de dados relacional quando avaliado com sistemas AOM. A implementação do EducService utilizando o Neo4J na sua camada de persistência está categorizada como a implementação *Not Only SQL* (NOSQL) a ser explorada por se considerar a hipótese de que tivesse um desempenho superior às abordagens relacionais (Yishan Li e Manoharan, 2013). Neo4J é um banco de dados NOSQL orientado a grafos baseado na máquina virtual Java, a

Java Virtual Machine (JVM), mas que pode ser implementado por diversas linguagens de programação e não apenas aquelas que também são baseadas na JVM, já que possui uma interface para intercomunicação entre aplicações de diferentes linguagens. Neo4J é considerado milhares de vezes mais rápido do que bancos de dados relacionais quando há existência de dados altamente conectados. No Neo4J é possível criar nós e arestas, onde os nós são utilizados para representação de registros, independentemente de uma estrutura pré-estabelecida como é o caso das tabelas na abordagem relacional, Já as arestas são utilizadas para representação de possíveis relacionamentos entre dois nós, sendo possível também representar atributos recorrentes dos relacionamentos através dessas arestas.

Os bancos de dados NOSQL são tecnologias bastante promissoras que vieram ganhando bastante espaço durante os últimos anos, o que tem diminuído um pouco da popularidade dos bancos de dados relacionais em geral. Por outro lado, acerca de todos os benefícios relatados sobre as melhores condições oferecidas pelos bancos de dados NOSQL em relação aos bancos de dados relacionais, durante a revisão bibliográfica deste trabalho, foi identificado, que implementações de bancos de dados NOSQL nem sempre se apresentam mais eficientes (Yishani Li & Manoharan, 2013). Isso foi motivador para a concretização deste trabalho.

Portanto, uma análise de desempenho considerando as versões do EducService executadas em cenários pré-definidos foi realizada. Nessa análise de desempenho foram calculadas as médias, desvio padrão e margem de erro, a significância através do teste *t-Student* e a razão entre as medidas de diferentes abordagens, para que se pudesse discutir os resultados obtidos.

1.1 OBJETIVO GERAL

O objetivo geral deste trabalho é avaliar o impacto de desempenho de tornar um sistema adaptável com AOM.

1.2 OBJETIVOS ESPECÍFICOS

- Desenvolver um sistema orientado a objetos sem AOM com persistência em um banco de dados relacional;
- Desenvolver um sistema adaptável com AOM em duas diferentes versões apresentando variações na camada de persistência, uma relacional e outra NOSQL orientada a grafos;

- Realizar uma avaliação de desempenho que compare o desempenho das diferentes versões desenvolvidas do mesmo sistema, sendo a primeira comparação entre a versão AOM com a versão não AOM onde ambas utilizam persistência em banco de dados relacional, e a segunda comparação entre as versões AOM, onde uma utiliza persistência em uma tecnologia relacional e a outra em uma tecnologia NOSQL.

1.3 METODOLOGIA

Segundo Gil (2002), uma pesquisa exploratória busca deixar mais explícito o problema a ser investigado. Uma pesquisa bibliográfica ocorre quando se há a necessidade em realizar um levantamento bibliográfico acerca do tema discutido e os trabalhos relacionados. Portanto, esse trabalho é composto por uma pesquisa exploratória, buscando explicitar o impacto do desempenho ao utilizar AOM e bibliográfica já que se fez necessário investigar os padrões de projeto especificados para se construir AOMs.

Para realização dessa pesquisa, foi realizado um *quasi-experiment* considerando o sistema EducService, implementado em diferentes versões a serem comparadas através de uma avaliação de desempenho com a interpretação dos dados, os resultados gerados em tal avaliação de desempenho são dados de caráter quantitativo.

Na caracterização do trabalho como uma pesquisa bibliográfica, a ação inicial foi realizar um levantamento bibliográfico, no qual se buscou identificar como construir sistemas adaptáveis utilizando AOM, e onde se buscou também identificar quais são os padrões de projeto existentes e bem documentados e quais as abordagens de persistência utilizadas em sistemas desse tipo. Com isso, foram identificados alguns problemas sobre o desempenho de tais sistemas e foi formulado o seguinte problema de pesquisa: **Qual o impacto de desempenho de tornar um sistema totalmente orientado a objetos em um sistema adaptável baseado em AOM?**

Até então, segundo Matsumoto e Guerra (2012) considerou-se que sistemas adaptáveis têm pior desempenho que os não adaptáveis e que bancos de dados relacionais não são a melhor alternativa de persistência para esses sistemas. Não se sabe ainda quão pior pode ser o desempenho e isto foi o que motivou a realização de uma avaliação de desempenho. Para que se tivesse uma avaliação de desempenho considerando diferentes formas de implementar o AOM, foi identificada a necessidade de investigar ferramentas que auxiliassem e facilitassem

o desenvolvimento de sistemas AOM, já que, conforme identificado na literatura, desenvolver sistemas AOM nem sempre é uma tarefa fácil para programadores tradicionais (Yoder e Johnson, 2002).

Para que se pudesse selecionar uma ferramenta capaz de auxiliar no desenvolvimento de sistemas AOM entre as existentes, foram definidos alguns critérios de inclusão:

- **CI_1**: Possibilitar o desenvolvimento de um sistema AOM através da linguagem de programação Java;
- **CI_2**: Proporcionar reuso na camada de persistência, para persistir tanto o metamodelo como os metadados em geral;
- **CI_3**: Permitir a mudança de implementação da camada de persistência, para que outras abordagens de bancos de dados fossem utilizadas;
- **CI_4**: Ser uma ferramenta *open source*;

No critério CI_1, a linguagem de programação Java foi estabelecida pelo fato de ser a linguagem de programação dominada pela equipe de pesquisa. Com o CI_2, o objetivo foi poder reutilizar outras implementações já realizadas da camada de dados, e com isso buscar uma proximidade ainda maior com o mundo real. Com o CI_3, se buscou a possibilidade de colaborar com novas implementações da camada de persistência a medida de que outros bancos de dados ainda não implementados pudessem ser avaliados. o CI_4 foi estabelecido devido a falta de recursos para que se pudesse custear uma ferramenta que não fosse disponibilizada de forma gratuita.

A ferramenta definida para auxiliar no desenvolvimento da versão AOM do sistema a ser desenvolvido durante a pesquisa ainda se encontrava em desenvolvimento, então como colaboração deste trabalho, essa ferramenta foi evoluída até que se pudesse chegar a uma versão que atendesse às necessidades do sistema a ser desenvolvido através desse arcabouço nas etapas seguintes do trabalho.

Portanto, foi desenvolvido um sistema em três versões. As versões do sistema foram projetadas de forma que fosse possível realizar uma avaliação de desempenho, coletando o tempo para a execução dos cenários avaliados, e com isso buscar proporcionar uma maior familiaridade com o problema de desempenho de sistemas AOM como uma contribuição.

Foi desenvolvida uma versão orientada a objetos sem muito foco em requisitos de adaptabilidade com um banco de dados relacional na camada de persistência, uma versão

AOM foi desenvolvida através da ferramenta auxiliar selecionada com persistência explorando o mesmo banco de dados relacional da primeira, e uma outra versão AOM foi também desenvolvida utilizando a ferramenta auxiliar selecionada e tendo sua camada de persistência com um banco de dados NOSQL orientado a grafos. Para selecionar as tecnologias de banco de dados a serem utilizadas foi considerado como critério a ferramenta mais popular para a abordagem a ser utilizada.

Ainda como uma pesquisa exploratória, a avaliação de desempenho entre as implementações das versões do sistema desenvolvido, foi realizada de forma que, através de cenários pré-definidos, fossem coletados os valores absolutos dos tempos das operações e com isso fossem aplicados cálculos da estatística que pudessem proporcionar valores quantitativos, como a média, o desvio padrão, o intervalo de confiança de 95%, o teste *t-Student* e a razão entre as versões. Os valores resultantes dos cálculos foram apresentados em tabelas e foram gerados gráficos com os valores absolutos para que pudessem ser discutidos.

Com parte das atividades da pesquisa já definidas, descritas anteriormente, o próximo passo foi buscar definir o design da pesquisa (do inglês, Research Design), ou seja, o processo de seleção de um método para um problema de investigação em particular (Easterbrook et al., 2008). Segundo Easterbrook et al. (2008) experimento controlado é uma investigação de uma hipótese testável, onde uma ou mais variáveis independentes são manipuladas para medir seu efeito em uma ou mais variáveis dependentes. Em um experimento, deve haver um alto nível de controle e quando não se consegue ter várias características de um experimento, mas não todas elas, o estudo pode ser caracterizado como um *quasi-experiment.* Por exemplo, quando os sujeitos (que no caso seriam desenvolvedores das diferentes versões) não são distribuídos aleatoriamente pelos tratamentos, que foi o que ocorreu nessa pesquisa já que houve apenas um desenvolvedor. Portanto, esse a pesquisa realizada neste trabalho de conclusão de curso foi caracterizada por um *quasi-experiment*.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentada uma base teórica da área em que se insere este trabalho, apresentando conceitos relevantes para melhor compreensão do que foi feito e dos resultados alcançados.

2.1 SISTEMAS ADAPTÁVEIS

Os sistemas tendem a evoluir ao longo do tempo, recebendo novas funcionalidades ou alterando parte dos serviços existentes que oferecem. Novas versões de sistemas operacionais tendem a surgir junto a novas plataformas de interface gráfica do usuário, bibliotecas e componentes de terceiros (Buschmann et al., 1996).

A evolução dos sistemas de software está ligada diretamente às mudanças ou ao surgimento de novos requisitos ao longo do tempo. Esses requisitos geram demandas a serem realizadas, muitas vezes em curtos prazos. A preocupação com o custo destinado para tais atividades é outro importante fator que deve ser considerado diante de tal contexto. Investigar técnicas que podem proporcionar vantagens em termos de custo e benefício é uma tarefa constantemente encontrada na Engenharia de Software (Ferreira, 2010).

Contudo, quanto menos adaptável um sistema for, maior será o custo e também o prazo demandado para a realização das atividades recorrentes das mudanças de requisitos comumente encontradas em quase todas as etapas do desenvolvimento de software. Daí a importância de projetar sistemas adaptáveis, considerados neste trabalho como sinônimo de sistemas que apresentam boa manutenibilidade e que podem sofrer algumas das mudanças requeridas até em tempo de execução. Segundo Buschmann et al. (1996), projetar uma arquitetura de software para atender a demandas por novos requisitos do usuário, se reflete em diminuir o esforço para se atender a tais demandas. Segundo Ferreira (2010), adaptabilidade também é uma propriedade que permite ao usuário final, com habilidades limitadas ou quase sem habilidades em programação, realizar alterações no software.

É importante considerar também que, segundo relatos da literatura quanto mais se proporciona dinamicidade a um sistema, pior pode ser o desempenho desse sistema (Foote e Yoder, 1998). Contudo, ao se pensar em sistemas adaptáveis, é necessário compreender os benefícios (flexibilidade, produtividade) e alguns problemas existentes (complexidade, desempenho).

A próxima seção apresenta o conceito de AOM, um estilo arquitetural utilizado durante o desenvolvimento deste trabalho e que possibilita a construção de sistemas adaptáveis. Neste trabalho, o foco será dado a sistemas adaptáveis, sem focar em sistemas adaptativos, que são os que mudam seu comportamento em resposta ao contexto em que se inserem. Nesse sentido, não se tratará aqui de aspectos como monitoração do contexto do software e que desencadeiam mudanças, nem da obrigação de mudanças de requisitos serem feitas em tempo de execução, mas sim em mudanças de requisitos normais e na facilidade de introduzi-las.

2.2 ADAPTIVE OBJECT MODEL - AOM

Adaptive Object-Model é citado na literatura como padrão arquitetural, como estilo arquitetural, como meta arquitetura, como arquitetura reflexiva, ou até mesmo como arquitetura do produto definida pelo usuário, que tem como propósito tornar os softwares mais genéricos, permitindo mudanças de requisitos em tempo de execução (Yoder et al., 1998). Sabendo que um estilo arquitetural define um conjunto de regras de projeto que identificam tipos de componentes e seus conectores (Shaw e Garlan, 1996), neste trabalho AOM será referenciado como estilo arquitetural.

Com AOM, mudanças podem ser feitas em tempo de execução, pois os sistemas são representados através de metamodelos, que podem ser persistidos em bancos de dados ou em arquivos XML, por exemplo. Contudo, o software poderá ser descrito, parcialmente ou completamente por metamodelos, o que permite representar classes, atributos, relacionamentos, regras de negócios e a interfaces gráficas do usuário (Ferreira, 2008).

Devido à possibilidade de realizar modificações na configuração do software em tempo de execução, esse modelo arquitetural foi denominado *Adaptive Object Model, mas* também é possível encontrar a denominação *Active Object Model (*Yoder et al., 1998*)*. Isso devido a ser uma arquitetura que promove a adaptabilidade de forma dinâmica, ou seja, sem que o software seja parado, modificado e executado novamente.

Quando se usa o estilo arquitetural AOM, na verdade se está usando uma série de padrões menores que descrevem como construir o software através de metamodelos, o que segundo Welicki et al.(2007) pode ser considerado como uma linguagem de padrões (*pattern language*). Existem padrões AOM para diversas camadas de um software como, para o

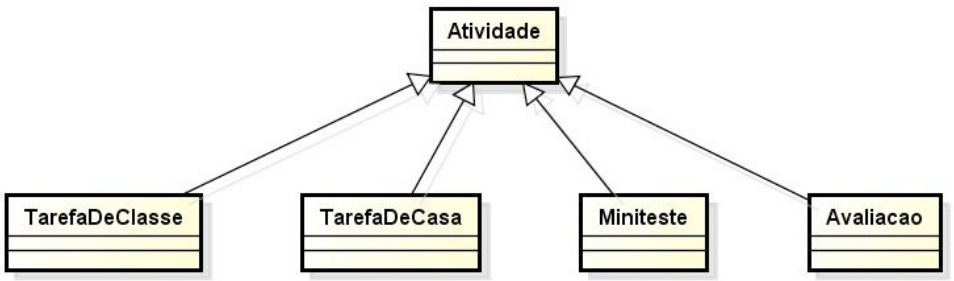
núcleo, para a camada de relacionamentos, para a camada de regras de negócio e para a camada de interface gráfica do usuário.

É possível também desenvolver sistemas AOM através de programação orientada a aspectos (AOP), o pode ajudar a minimizar alguns problemas enfrentados durante o desenvolvimento de um sistema AOM, como o entrelaçamento de código, onde códigos de regras de negócio e os relacionados à adaptabilidade da interface gráfica, por exemplo, podem ser misturados, por exemplo (Dantas et al., 2004). Mesmo usando AOP, é necessário de qualquer forma implementar elementos do estilo arquitetural AOM e que fazem parte de seu núcleo, os quais exploram alguns padrões de projeto responsáveis pela especificação de entidades e propriedades, e que são apresentados a seguir.

2.2.1 Type Object

No modelo orientado a objetos, em arquiteturas mais convencionais, as entidades dos sistemas são expressas através de diversas classes, onde cada uma delas objetiva a representação de um tipo de objeto (Yoder et al., 2001). Isto está ilustrado no diagrama de classes apresentado na Figura 1 e que apresenta um exemplo.

Figura 1- Herança de classes sem o *Type Object* em uma arquitetura orientada a objetos.

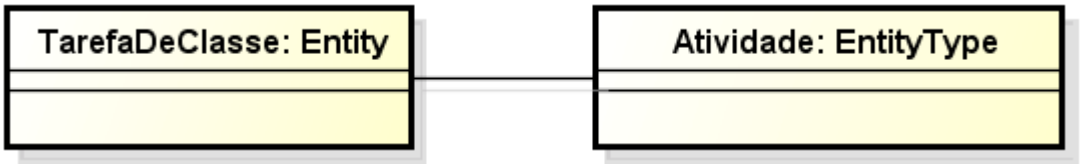


Fonte: o autor (2015)

A Figura 1 apresenta uma hierarquia de classes, através de relações de generalização, onde “Atividade” é uma superclasse e “TarefaDeClasse”, “TarefaDeCasa”, “Miniteste” e “Avaliacao” são as subclasses. Nessa abordagem foi necessário criar cinco classes, e caso um novo tipo de atividade viesse a surgir, por exemplo, uma entidade “ProvaFinal”, uma nova classe teria de ser criada e o sistema teria de ser recompilado para suportar atividades desse novo tipo.

Já em um AOM, é possível através de uma representação genérica, especificada pelo padrão *Type Object,* realizar a manipulação de novos tipos em tempo de execução e também a criação de suas próprias regras de verificação de tipo (Johnson e Wolf, 1997), sem que haja a recompilação. Para que isso seja possível, o padrão em questão especifica duas classes, que podem ser visualizadas na Figura 2.

Figura 2 - Type Object: TarefaDeClasse como uma Atividade.



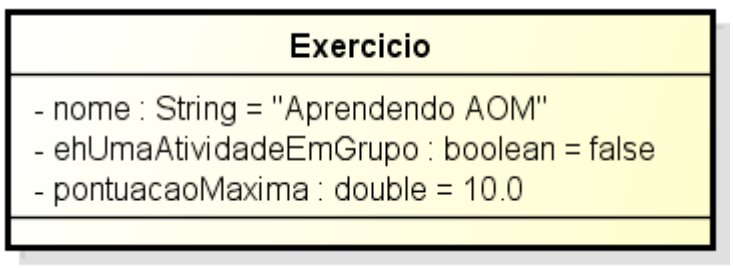
Fonte: o autor (2015)

A Figura 2 ilustra um diagrama de objetos no qual se percebe a existência de classes genéricas utilizadas para representação de entidades através do *Type Object*. São elas: *Entity* e *EntityType*. Como se trata de um diagrama de objetos, através dele é ilustrado que em certo momento da execução a classe *EntityType* foi instanciada de forma a representar uma "Atividade" (representada pela superclasse na Figura 1) e a classe *Entity* foi instanciada de forma a referenciar uma "TarefaDeClasse" (uma das subclasses encontradas na Figura 1).

2.2.2 Property

Esse padrão especifica uma possível forma de representar as propriedades das entidades através de metadados. Em arquiteturas mais convencionais, para especificar uma propriedade, é necessário incluí-la dentro da própria classe a qual pertence, conforme especificado na Figura 3.

Figura 3 - Propriedades da uma entidade Exercício em OO.

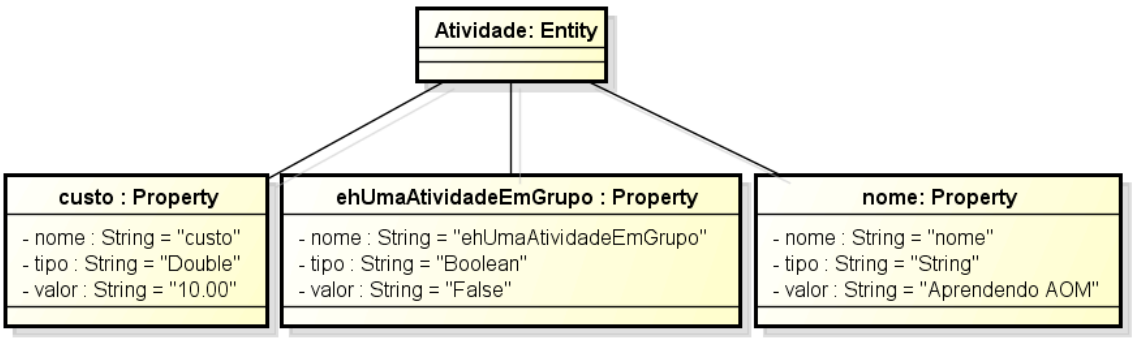


Fonte: o autor (2015)

Na Figura 3 é mostrada uma classe denominada “Exercicio”, que contém as seguintes propriedades: “nome”, do tipo texto e que tem como valor atribuído “Aprendendo AOM”; “ehAtividadeEmGrupo”, do tipo booleano e que tem como valor atribuído “false”; e a propriedade “pontuacaoMaxima”, do tipo número real, e como valor atribuído o “10.00”. Já que o próprio nome do padrão reflete diretamente o seu principal objetivo, o *Property* segundo Yoder e Foote (1998), especifica uma maneira de representar as propriedades de uma entidade através de metadados, atendendo assim a mais um dos objetivos de AOM: tornar dinâmicas as propriedades de uma entidade.

Nesse padrão, uma classe *Property* é especificada para a representação de uma propriedade, que deve estar relacionada com uma *Entity,* assim permitindo expressar à qual entidade uma propriedade pertence, conforme apresentado na figura a seguir.

Figura 4 – Aplicação do padrão Property.



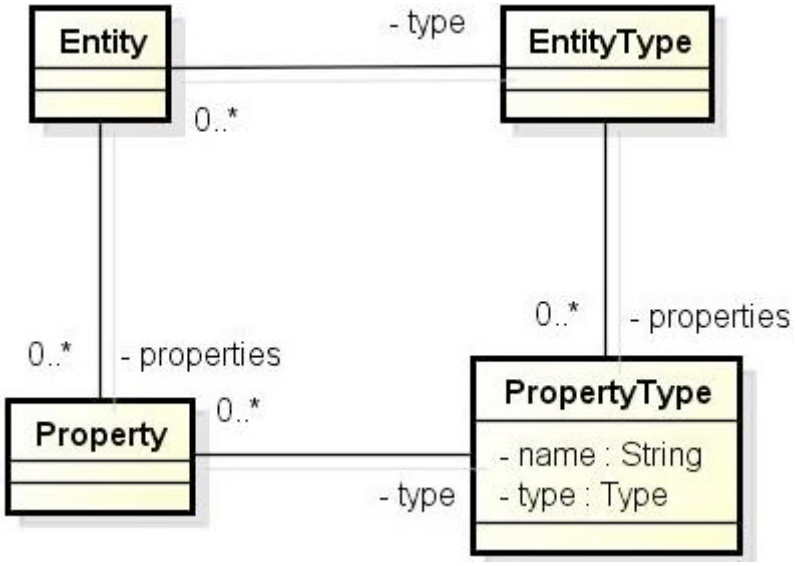
Fonte: o autor (2015)

A Figura 4 apresenta, por meio de um diagrama de objetos, a transformação do modelo orientado a objetos convencional para o modelo especificado para um AOM. A classe *Entity* passou a se relacionar com a classe *Property.* Com isso percebe-se que várias p*roperties* podem estar agregadas a uma mesma *Entity.* Cada propriedade definida no modelo orientado a objetos convencional, a partir da aplicação desse padrão tornou-se uma instância da classe *Property,* e cada característica de uma propriedade passou a ser expressa através de atributos de um *Property*. São eles: “nome”, que deve representar o nome da propriedade; o “tipo", que deve representar o tipo do valor da propriedade (texto, inteiro, número real, etc), e o “valor”, que deve representar o valor de instância da propriedade.

2.2.3 Type Square

A composição dos padrões *Type Object* e *Property,* permite a criação de uma estrutura base que pode ser considerada como o núcleo de um AOM, e que constitui o padrão denominado por *Type Square*, apresentado na Figura 5. No *Type Square*, o padrão *Type Object* foi aplicado mais uma vez, sendo que desta vez objetivando proporcionar um maior dinamismo para *Property*, que passa a se relacionar com um *PropertyType.* Através dessa alteração, as propriedades também passaram a possuir tipos manipuláveis. Um *PropertyType* poderá ter nenhum ou muitos *Properties e cada Property* deve ter apenas um *PropertyType*.

Figura 5 - Representação do padrão Type Square.



Fonte: Yoder et al. (2001)

A Figura 5 apresenta a estrutura do núcleo que um AOM deve ter, o *TypeSquare*. Observa-se a presença de quatro classes e as suas relações. São elas: *Entity*, *EntityType*, *Property* e *PropertyType*. *EntityType* e *PropertyType*, que são modelos para manipulação de novos tipos de entidade e propriedades através de metamodelos, respectivamente. *Entity* e *Property* são utilizadas para representação de instâncias de entidades e instâncias de propriedades através de metadados, respectivamente.

2.3 PERSISTÊNCIA DE DADOS

O termo persistência de dados, nesse trabalho, se refere ao armazenamento de dados eletrônicos de maneira não volátil, ou seja, que diante da ausência ou presença de pulsos elétricos, os dados estarão armazenados. Uma das maneiras mais eficientes para realizar a persistência de dados é através de sistemas de gerência de banco de dados (SGBD). Segundo Heuser (1998), um SGBD é um software que incorpora as funções de definição, recuperação e alteração de dados em um banco de dados.

Um SGBD e uma aplicação de software são dois sistemas integráveis, que geralmente trabalham em conjunto, onde o software poderá utilizar os serviços de persistência oferecidos pelo SGBD. Na literatura, podem ser encontradas algumas maneiras padronizadas para se implementar uma camada de persistência de acordo com necessidades geralmente comumente exigidas no projeto do software. O padrão de projeto *Data Access Object* é uma delas, a qual propõe que a aplicação de software apresente interfaces para manipulação de dados de forma que seja possível modificar facilmente a implementação escolhida para realização da persistência.

2.4 PADRÃO DATA ACCESS OBJECT - *DAO*

Para muitas aplicações, o armazenamento persistente dos dados é implementado com diferentes mecanismos distintos e há diferenças nas APIs usadas para acessar esses mecanismos de armazenamento persistente (Sun, 2002). Segundo Buschmann et al. (2007), o padrão DAO permite encapsular o acesso e a manipulação de dados persistentes em uma camada separada das demais camadas de lógica da aplicação. Contudo, entende-se que esse padrão tem como objetivo especificar uma maneira de implementação da camada de persistência que gere baixo acoplamento entre as demais camadas existentes.

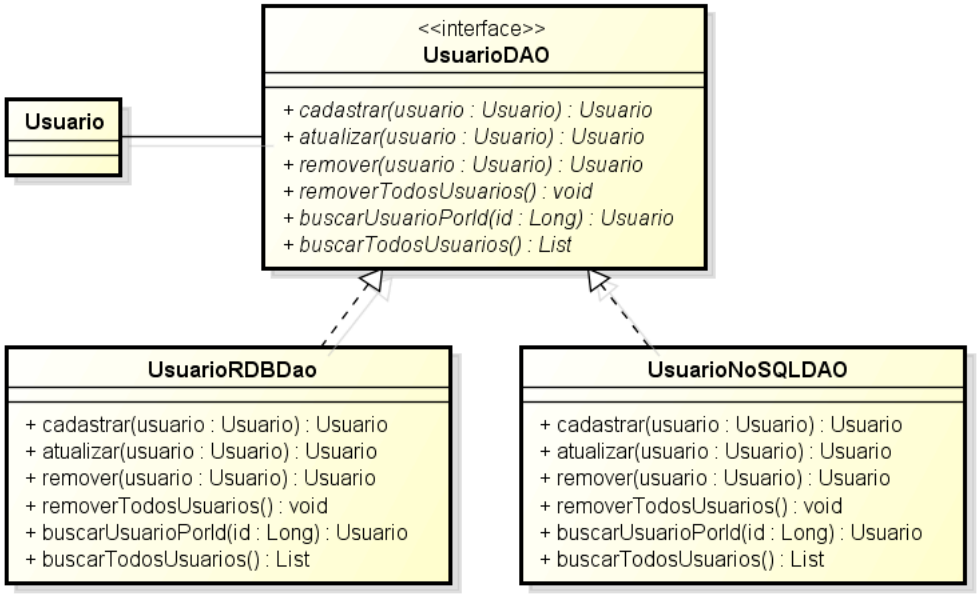
O padrão de projeto DAO (Sun Microsystems, 2015) especifica que uma interface deve ser criada para cada tipo de entidade a ser persistida. Através de cada interface, será possível implementar diversas maneiras de persistir os dados de tal entidade, como por exemplo, em um banco de dados relacional, em um esquema de dados baseados em XML, ou até mesmo em um banco de dados orientado a grafos.

Guerra (2012, p. 40), acerca das interfaces a serem definidas para cada tipo de entidade do modelo do software, afirma.

> *A interface DAO define os métodos gerais de acesso a dados que serão disponibilizados, como para recuperação, gravação e exclusão de entidades. Essa interface precisará ser implementada para cada entidade do sistema, para a criação dessas operações de cada uma.*

Na Figura 6, apresentada a seguir, é mostrado um exemplo de implementação do padrão DAO em um contexto contendo uma classe de entidade que deve ser serializada, o que implica na existência de uma interface e as implementações para persistência dessa entidade através de duas formas, uma utilizando banco de dados relacional e outra utilizando um banco de dados NOSQL.

Figura 6- Implementação do padrão DAO para diferentes bancos de dados.



Fonte: o autor (2015)

Na Figura 6, uma interface chamada “UsuarioDAO” foi especificada, na qual se encontram alguns métodos a serem desenvolvidos pelas classes que a implementarem. No caso da Figura 6, as classes “UsuarioRDBDao” e “UsuarioNoSQLDAO” implementaram a interface e é perceptível a presença dos métodos especificados pela interface, o que torna possível a persistência e manipulação de registros de usuário tanto em bancos de dados relacionais como em bancos de dados NOSQL.

2.5 NEO4J

Segundo Hecht e Jablonski (2011), no passado, até antes do surgimento dos bancos de dados NOSQL, todos os modelos de dados, geralmente, eram armazenados em bancos de dados relacionais, e isso, algumas vezes, ocasionava um aumento de complexidade da aplicação a ser desenvolvida. Frameworks para mapeamento dos dados com custo elevado eram adotados, além de algoritmos de alta complexidade terem de ser criados.

Existem diversos tipos de bancos de dados NOSQL e que assumem diferentes abordagens, mas o que possuem em comum é que eles não são relacionais. Esses bancos de dados lidam com dados não estruturados (Leavitt, 2010). Segundo Yishan e Manoharan (2013), o sucesso dos bancos de dados não relacionais proprietários deu início a uma série de outros empreendimentos de fonte aberta e de banco de dados não relacional de código fechado. Esses bancos de dados NOSQL cresceram em popularidade por causa da facilidade de acesso, velocidade e escalabilidade.

Miller (2013) afirma sobre os bancos de dados orientados a grafo, que diante da modelagem de objetos e as relações entre eles, quase tudo pode ser representado em um grafo correspondente. Bancos de dados orientados a grafos são do tipo NOSQL, nos quais os dados são armazenados com o formato de nós, onde as relações são representadas por arestas e não existe um modelo padrão definido para armazenamento dos dados. Por exemplo, em um nó é possível armazenar um aluno com nome “Fernando”, já em outro nó é possível armazenar uma disciplina “Engenharia de Software”, por fim é possível especificar uma aresta chamada “está matriculado em”, que liga os nós “Fernando” e “Engenharia de Software” como forma de expressar a relação: “Fernando está matriculado em Engenharia de Software”. Perceba através da Figura 7, que o modelo de informações armazenadas nos nós podem variar conforme o exemplo citado.

Figura 7 - Exemplo de grafo representado em NOSQL

Fonte: o autor (2015)

É perceptível que, em relação ao modelo relacional, o qual apresenta tabelas com campos pré-definidos, no NOSQL orientado a grafo o armazenamento dos dados, independentemente do tipo, ocorre através dos nós e das arestas. Em alguns casos, propriedades também podem ser adicionadas nas arestas, representado atributos recorrentes de relacionamentos.

Para manipulação dos dados no modelo relacional, existe a linguagem SQL (*Structured Query Language*). Já para os bancos de dados orientados a grafo, segundo Miller (2013) existe a necessidade de padronizar uma linguagem para esse propósito e devido a essa falta de padronização foram elencadas uma série de linguagens e *frameworks.*

Para o Neo4J, ainda conforme Miller (2013) existem duas linguagens primárias utilizadas para manipulação dos dados: Gremlin e Cypher. Gremlin é uma DSL, implementada em Groovy, que opera também sobre a JVM. Já a Cypher é uma linguagem mais específica para travessia dos grafos, bastante semelhante à sintaxe e semântica de SQL.

2.5.1 Implementando persistência de dados com Java e Neo4J com Cypher

Existem várias maneiras de implementar a comunicação de uma aplicação de software com o banco de dados Neo4J, seja através de sua interface REST (*Representational State Transfer)* ou através das APIs Java disponibilizadas para uma série de linguagens de programação. Em Java é possível utilizar uma implementação do *Embedded Neo4J*, o qual é incorporado em um processo da JVM. Essa é uma forma de implementação aconselhada para implementações de testes de unidade como também para avaliações de desempenho, o que elimina a necessidade de uso de rede para comunicação para com o Neo4J.

Ao utilizar uma implementação *Embedded Neo4J,* a aplicação de software deverá utilizar *Neo4J Core-Java-API.* Essa API permite a execução de scripts Cypher para manipulações dos dados que serão persistidos ou daqueles que já se encontram persistidos.

Figura 8 - Persistindo e buscando dados no Neo4J.

```
public EntityType create(EntityType entityType) {
    try (Transaction tx = connector.iniciarTransacao()) {
        Node noEntityType = connector.getGraphDatabaseService().createNode(NodeType.ENTITY_TYPE);
        noEntityType.setProperty("id", ++autoIncrementId);
        noEntityType.setProperty("version", 0);
        noEntityType.setProperty("namespace", entityType.getNamespace().toLowerCase());
        noEntityType.setProperty("name", entityType.getName().toLowerCase());
        tx.success();

        entityType.setId(autoIncrementId);
        entityType.setVersion(0);
        return entityType;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

public List<EntityType> listAll() {
    List<EntityType> entitiesTypes = new LinkedList<EntityType>();

    try (Transaction tx = connector.iniciarTransacao();
            Result result = connector.getGraphDatabaseService().execute(
                    "MATCH (et:" + NodeType.ENTITY_TYPE + ") return et")) {

        Iterator<Node> iterator = result.columnAs("et");
        for (Node node : IteratorUtil.asIterable(iterator)) {
            EntityType entityType = newEntityType(node);
            entitiesTypes.add(entityType);
        }
    }

    return entitiesTypes;
}
```

Fonte: o autor (2015)

Na Figura 8 se encontram dois métodos, o primeiro realiza inserções e o segundo realiza consulta de todos os registros de grafos com *label* "*ENTITY_TYPE*". No segundo método é possível perceber a presença de um *script Cypher* o qual foi utilizado como *query* para obtenção de todos os registros com o *label* especificado.

3. LIVING OBJECT MODEL - LOM

Como uma alternativa já conhecida e com um número considerável de trabalhos desenvolvidos encontrados na literatura, se optou por utilizar AOM para explorar o desenvolvimento de sistemas adaptáveis. No entanto, desenvolver um sistema AOM não é uma tarefa fácil, principalmente para desenvolvedores convencionais, acostumados a lidar apenas com o paradigma orientado a objetos sem o nível de abstração de metadados (Yoder e Johnson, 2002). Lidar com metamodelos e metadados, os quais poderão ser manipulados em tempo de execução e onde qualquer ação de mudança sobre eles refletirá diretamente no comportamento do software, exige uma série de cuidados, principalmente se o software já se encontrar em ambiente de produção. Sendo assim, considerou-se importante o suporte de um arcabouço.

Segundo Roberts e Johnson (1997) arcabouços (do inglês, *framework*) são modelos reutilizáveis de todo ou parte de um sistema de software descrito por um conjunto de classes abstratas e a forma como as ocorrências dessas classes colaboram. O uso de arcabouços pode reduzir o custo de desenvolvimento de um aplicativo por uma ordem de magnitude, pois permite a reutilização de design e código. Portanto, o *Living Object Model (*LOM*)* utilizado neste trabalho, é um arcabouço *open source,* desenvolvido na linguagem de programação JAVA*,* com o qual se pretende proporcionar mais qualidade e produtividade através do reuso de código para a construção de sistemas AOM. Através do LOM será possível desenvolver sistemas AOM completos através de um conjunto de classes abstratas que colaboram entre si de uma forma específica, o que o caracteriza como arcabouço, segundo a definição acima. Na versão V.0.0.9, utilizada nesse trabalho, já é possível desenvolver o núcleo e a camada de persistência. Posteriormente, serão disponibilizadas funcionalidades para criação de relacionamentos, regras de negócio e interface gráfica do usuário.

O LOM faz parte do trabalho de doutorado do professor Rodrigo Vilar (Oliveira et al., 2015) e uma contribuição deste trabalho foi a implementação de parte da versão V.0.0.9., a qual foi utilizada para o desenvolvimento do estudo de caso.

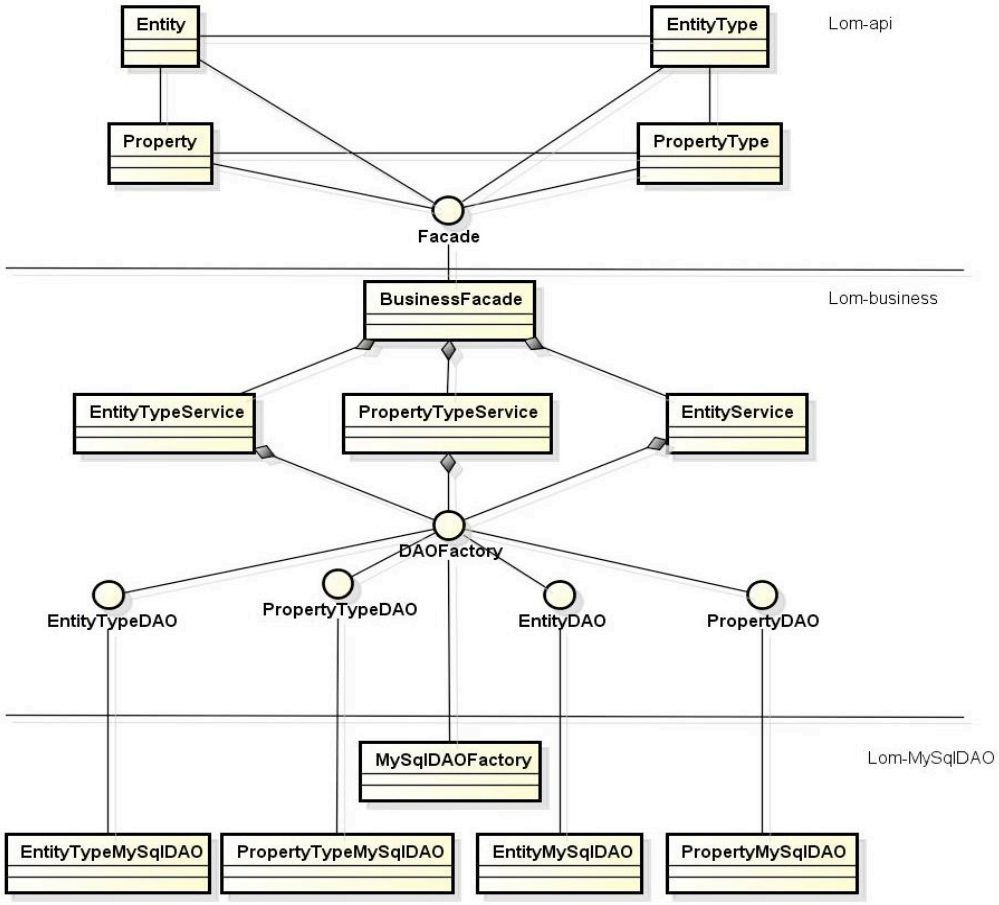
3.1 MÓDULOS

O LOM encontra-se dividido em três camadas, conforme ilustrado na Figura 9. São elas: (i) a camada LOM-api que contém a implementação do padrão *Type Square*, composto

pelos padrões *Type Object* e *Property*; (ii) a camada LOM-business que contém a implementação das operações de cadastro, atualização, remoção e consultas envolvendo as entidades presentes na camada LOM-api. Essas operações estão disponibilizadas através de uma fachada especificada, através de uma interface, no módulo LOM-api e implementada no LOM-business. Ainda no LOM-business estão especificadas as interfaces a serem implementadas pela camada que cuidará da persistência dos metadados.

Para a camada de persistência de dados (iii) existem diferentes implementações as quais podem ser facilmente substituídas devido ao baixo acoplamento existente entre as implementações. Na camada de persistência, cada interface DAO especificada na camada LOM-business deverá ser implementada. Para selecionar quais seriam os bancos de dados que seriam utilizados nas implementações dessa camada, conforme critério definido na metodologia, o banco de dados relacional MySQL e o Neo4J foram os mais populares nas categorias de banco de dados relacional e banco de dados orientado a grafos, respectivamente, de acordo com DB-Engines Ranking (2015). Foi escolhido um banco de dados orientado a grafos pela curiosidade em explorar essa abordagem que vem se popularizando cada vez mais.

Figura 9 - Possível interação entre as implementações das camadas do LOM



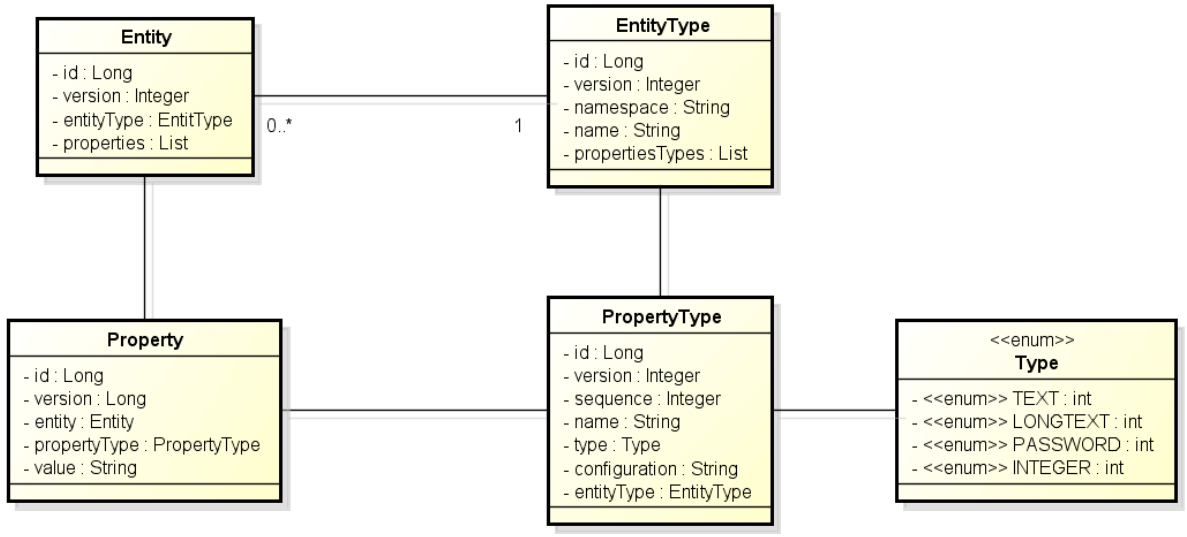
Fonte: o autor (2015)

A Figura 9 ilustra uma possível interação entre implementações das três camadas mencionadas anteriormente, o *LOM-api*, o *LOM-business* e o LOM-MySQLDAO, sendo o último uma implementação para realização da persistência no banco de dados MySQL. Para a camada de persistência, é possível visualizar que é possível trocar para outra abordagem implementada facilmente devido à existência do padrão de projeto DAO. É possível observar que essa divisão de camadas deixou evidente a separação de responsabilidades e buscou evitar o entrelaçamento de código. Nas seções a seguir, a implementação de cada camada será melhor detalhada.

3.1.1 LOM-api

Conforme especificado pelo padrão *Type Square,* no *LOM-api* existem quatro principais entidades implementadas. São elas: *Entity*, *EntityType*, *Property* e *PropertyType.*

Figura 10 - Classes do LOM-api.



Fonte: o autor (2015)

Na Figura 10 é possível visualizar que todas as quatro principais entidades citadas, possuem como características comuns duas propriedades: o *id* e o *version*. O *id* é uma propriedade para identificação única de cada instância. Já o *version* é uma propriedade utilizada para representar a quantidade de operações de escrita realizadas para cada instância.

Um *EntityType* tem por objetivo armazenar um tipo de entidade. Isso implica dizer que criar um *EntityType* é equivalente a criar uma classe de entidade em programação orientada a objetos. No entanto, as propriedades de um *EntityType* também se assemelham às propriedades de uma classe de entidade. O *namespace* é equivalente ao nome do pacote em que uma entidade se encontra. Por exemplo, uma entidade que se encontra num pacote denominado por *model,* no LOM deverá ser instanciado um objeto *EntityType* e atribuir o valor *"model" ao namespace.* Já a propriedade *name* tem por objetivo armazenar o nome da entidade a ser representada através de metadados usando o LOM, ou seja, armazenar o valor "usuario", por exemplo, é equivalente a denominar uma classe de entidade por "Usuario".

Um *PropertyType*, assim como um *EntityType,* tem por objetivo representar um tipo, mas desta vez para representar um tipo de propriedade. Isso quer dizer que, para representar o nome de um aluno, por exemplo, é necessário criar um *PropertyType* e atribuir o valor "nome" para a propriedade do *PropertyType* denominada por *name*.

Em um *PropertyType*, o *sequence* é uma propriedade utilizada para armazenar a ordem em que tal *PropertyType* se encontrará quando relacionado com um *EntityType* (e.g. a primeira propriedade).

O *type,* uma propriedade do tipo *enum*, poderá assumir um entre os valores pré-definidos apresentados em *Type.* A Figura 10 ilustra seu propósito é armazenar qual o tipo de valor que tal *PropertyType* poderá receber. Em programação orientada a objetos, é comum entre algumas principais linguagens a existência dos tipos de propriedades *int, double* e *float,* por exemplo. O objetivo do *type* é o mesmo, representar se tal tipo armazenará um dos valores: *TEXT, LONGTEXT, PASSWORD, INTEGER* ou *REAL*.

A propriedade *configuration* encontra-se presente em *PropertyType* para que simples regras de negócio já possam ser especificadas durante a definição das propriedades através de metadados. Elas devem estar expressas através de *JavaScript Object Notation* (JSON) (Crockford, 2006), na qual é possível especificar uma série de restrições e os valores para tais restrições, por exemplo, dizer que uma propriedade, quando for persistida, deverá receber obrigatoriamente um valor não nulo e que esse valor deverá ser maior ou igual a zero, é o mesmo que atribuir a configuração *{'mandatory': true, 'minvalue' : 0}* no campo *configuration* de um *PropertyType.*

Um *Entity* está implementado de modo que se possa representar uma instância de uma entidade. O *EntityType* deve ser utilizado para especificar um tipo de entidade. Já para persistir um registro de entidade, é necessário utilizar o *Entity.* Um *Entity* está relacionado com um *EntityType*, sendo assim possível, através de metadados, representar a qual tipo a instância de uma entidade pertence.

Um *Property,* da mesma forma que um *Entity,* foi especificado para ser utilizado para a criação de instâncias, sendo que o *Property* tem por objetivo possibilitar a criação de instâncias de propriedades. Isso quer dizer que, quando a instância de uma entidade for criada, consequentemente, as instâncias de suas propriedades também devem ser criadas, e para cada instância de propriedade, um valor, válido ou nulo, deverá ser atribuído. Entretanto, vale destacar a presença dos atributos *entity e propertyType,* e da propriedade *value* dentro da classe *Property.* O *entity* está especificado para representar a qual instância de entidade tal instância de propriedade pertence, o *propertyType* para indicar a qual o tipo de propriedade tal instância de propriedade está relacionado e a propriedade *value* para armazenar o valor bruto da propriedade quando instanciada.

3.1.2 LOM-business

Conforme comentado brevemente anteriormente, o *LOM-business* é uma implementação da camada de negócio do arcabouço LOM. Através da Figura 10 é possível perceber a existência da classe *BusinessFacade,* uma implementação concreta da interface *Facade.*

Deve ser através de uma instância da *BusinessFacade* que o programador que utilizar o arcabouço LOM para auxiliar no desenvolvimento do seu sistema AOM, deverá interagir.

A interface *Facade* especifica uma série de métodos úteis para operações de cadastro, atualização, consulta e remoção das classes do *LOM-api,* úteis para a representação de metadados que estarão ou serão armazenados em uma unidade persistente. A importância da utilização dessa fachada é que, caso as propriedades ou atributos das classes do *LOM-api* tenham sido preenchidas de maneira incorreta. As regras de validações implementadas no LOM-business realizarão as verificações necessárias, o que elimina a necessidade de um programador reescrever todo o módulo AOM de seu sistema e implementar essas regras que podem ser complexas quando não há ainda muita afinidade com esse estilo arquitetural.

Durante a instanciação de uma classe *BusinessFacade,* há a necessidade de se instanciar também uma classe que implemente a interface *DAOFactory.* Essa instância indicará qual implementação da camada de persistência será utilizada e em qual abordagem de banco de dados serão persistidos os metadados e o metamodelo.

Durante a realização desse trabalho, foram realizadas duas implementações para a camada de persistência dos metadados. Uma para possibilitar a persistência no banco de dados MySQL e a outra para a persistência no banco de dados orientado a grafo Neo4J, as quais serão melhor apresentadas nas próximas seções.

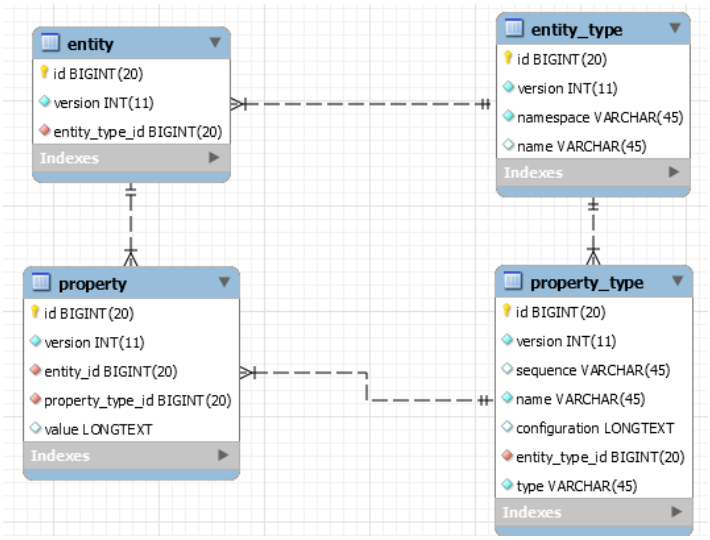
3.1.3 LOM-MySQLDAO

MySQL é um sistema gerenciador banco de dados relacional, no qual os dados são persistidos através de tabelas. A cada nova instância de uma entidade que é persistida, uma nova linha em uma ou mais tabelas são criadas, e as propriedades dividas através de colunas. Nesse tipo de banco de dados, as tabelas existentes se relacionam através de atributos chaves, os quais são identificadores únicos e denominados por chave primária quando se encontram armazenados na sua tabela de origem. Quando se encontram em coluna(s) de outra tabela, objetivando expressar um relacionamento, são denominados por chave estrangeira (Heuser,

1998). Nessa implementação da camada de persistência, o código existente foi desenvolvido para realizar a persistência dos metadados no banco de dados MySQL através da API *Java Database Conectivity* (JDBC) e de scripts em *Structured Query Language* (SQL).

Cada entidade do *LOM-api,* quando persistida no banco de dados *MySQL,* de acordo com a implementação realizada, se refletiu em uma tabela, conforme pode ser visualizado na Figura 11.

Figura 11 - Estrutura banco de dados do LOM no MySQL.



Fonte: o autor (2015)

A chave primária de cada tabela foi denominada por *id* e as chaves estrangeiras são as propriedades que possuem o sufixo *id*, são elas: *entity_type_id* nas tabelas *property_type* e *entity; entity_id* e *property_type_id* na tabela property. É importante enfatizar que, qualquer sistema AOM que for criado com auxílio do LOM e que utilizar o módulo *LOM-MySQLDAO,* apresentará a mesma estrutura de tabelas e colunas para representação dos metadados, levando a reuso.

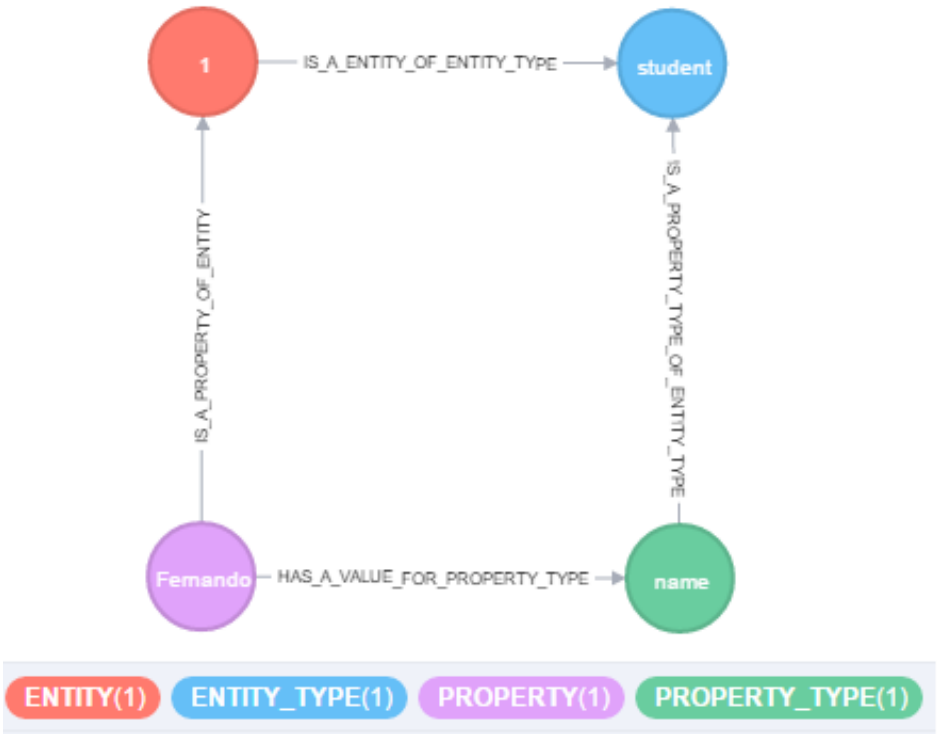
3.1.4 LOM-Neo4JDAO

Outra forma implementada para realizar persistência dos metadados foi através do módulo *LOM-Neo4JDAO*, que realiza a persistência em um banco de dados orientado a grafo denominado Neo4J. No Neo4J é possível representar registros através de nós, os quais podem possuir um rótulo (*label)*, e uma série de propriedades que não necessitarão estar previamente pré-definidas, como no caso das tabelas e das colunas nos bancos de dados relacionais. Os

relacionamentos entre os nós são representados por arestas, as quais também podem ter propriedades, mas não foram necessárias na implementação atual.

No caso do LOM-Neo4JDAO, cada tipo de entidade persistida e relacionada com outras entidades também apresentarão a mesma estrutura. O que pode variar é a quantidade de nós criados, pois à medida que a quantidade de metadados persistidos cresce, a quantidade de nós e relacionamentos criados também aumenta. As linhas das tabelas do MySQL são equivalentes aos nós no Neo4J. Já as tabelas, não podem ser comparadas da mesma forma, pois no Neo4J não há nada equivalente.

Figura 12 - Grafo persistido no Neo4J



Fonte: o autor (2015)

Através da Figura 12 é possível visualizar um exemplo de grafo criado pelo LOM-Neo4JDAO. Diante de registros das entidades do *LOM-api.* Esse grafo apresenta apenas um nó referente a cada uma das entidades (*EntityType, PropertyType, Entity e Property*) e seus respectivos relacionamentos. Visando ter um maior controle durante a avaliação de desempenho realizada neste trabalho, descrita nas próximas seções, a implementação realizada para esse módulo utilizou o recurso *embedded* do Neo4J, conforme indicado pela documentação dessa tecnologia, no qual o mesmo passa a utilizar um diretório especificado

pelo programador para realizar as atividades de persistência, eliminando possíveis vieses caso houvesse a comunicação via rede na implementação tradicional.

4. O SISTEMA EDUCSERVICE

O desenvolvimento do sistema EducService[1] foi proposto como atividade no projeto Programa de Licenciatura (PROLICEN) do Centro de Ciências Aplicadas e Educação da Universidade Federal da Paraíba intitulado "Estendendo os Limites da Sala de Aula com o apoio da Internet" visando o desenvolvimento de um sistema Web para gerenciar exercícios online. Considerou-se interessante aproveitar esse mesmo sistema para a pesquisa realizada neste trabalho, por ser um sistema real e por seu escopo ser conhecido pelo autor, além da facilidade de acesso a documentação do mesmo. Para a realização do *quasi-experiment*, este sistema foi implementado novamente em diferentes versões, de forma que fosse possível avaliar e comparar seu desempenho posteriormente.

O EducService apresenta alguns requisitos de adaptabilidade já que seu objetivo é gerenciar exercícios e questões e em um sistema assim poderão surgir vários novos requisitos, como novos tipos de questões, formas de apresentá-las, etc. Atualmente, ele suporta apenas questões de múltipla escolha, dissertativas e de verdadeiro ou falso. O EducService tem por objetivo permitir que exercícios (conjunto de questões) de vários tipos possam ser facilmente gerenciados em um ambiente online, no qual alunos possam respondê-los, receber um *feedback* imediato sobre suas respostas de forma automática e onde se possa também gerar para os professores criadores dos exercícios algum *feedback* sobre o desempenho dos estudantes, para assim poderem, por exemplo, realizar intervenções em sala de aula direcionadas a deficiências identificadas.

Esse capítulo apresenta como o sistema EducService foi desenvolvido em suas versões, de forma que se possa apresentar os detalhes relevantes em cada versão, além de demonstrar como foi possível desenvolvê-lo utilizando o LOM na versão V.0.0.9, a qual foi desenvolvida para a elaboração desse trabalho.

4.1 VERSÕES DO EDUCSERVICE

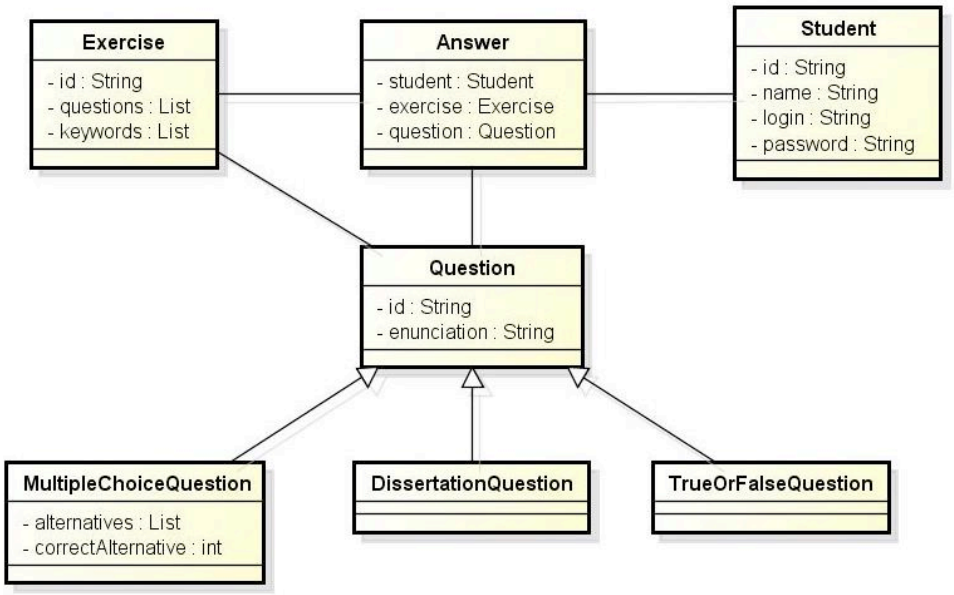
Para esse estudo, alguns requisitos funcionais do EducService foram implementados em três versões pelo próprio autor do trabalho. A primeira foi (i) a implementação da versão totalmente orientada a objetos sem foco em requisitos de adaptabilidade, desenvolvida através

[1] EducService, disponível em: http://www.educservice.com. Acesso em: 02 dez 2015.

da linguagem de programação JAVA e com persistência realizada no banco de dados relacional MySQL; (ii) a segunda foi uma implementação AOM através do arcabouço LOM com a camada de persistência do metamodelo e dos metadados configurada para o banco de dados MySQL; e (iii) a terceira também sendo uma nova compilação da segunda versão mas com uma variação na configuração da camada de persistência, que dessa vez foi configurada para persistência dos metadados e do metamodelo no banco de dados orientado a grafo Neo4J.

A versão totalmente orientada a objetos sem foco em requisitos de adaptabilidade e com persistência no banco de dados *MySQL* foi desenvolvida para que na avaliação de desempenho se pudesse verificar o impacto do desempenho com a segunda implementação, que foi com AOM através do arcabouço LOM e com persistência também no banco de dados *MySQL*. Então, assim tornou-se possível avaliar e comparar o desempenho de um sistema orientado a objetos sem foco em requisitos de adaptabilidade com persistência em um banco de dados relacional com um sistema AOM também com persistência em um banco de dados relacional. Já a terceira versão foi desenvolvida para que se pudesse verificar avaliar e comparar o desempenho da versão AOM utilizando um banco de dados relacional com a versão AOM utilizando um banco de dados NOSQL orientado a grafos. O MySQL e o Neo4J foram os bancos de dados que apresentaram maior popularidade em suas categorias, banco de dados relacional e banco de dados orientado a grafos.

Figura 13 - Diagrama de classes do EducService.



Fonte: o autor (2015)

Por limitações de recursos, como tempo e a quantidade de colaboradores no projeto, apenas parte do escopo do sistema EducService pôde ser coberta. A Figura 13 apresenta o diagrama de classes gerado diante dos requisitos coletados para serem implementados.

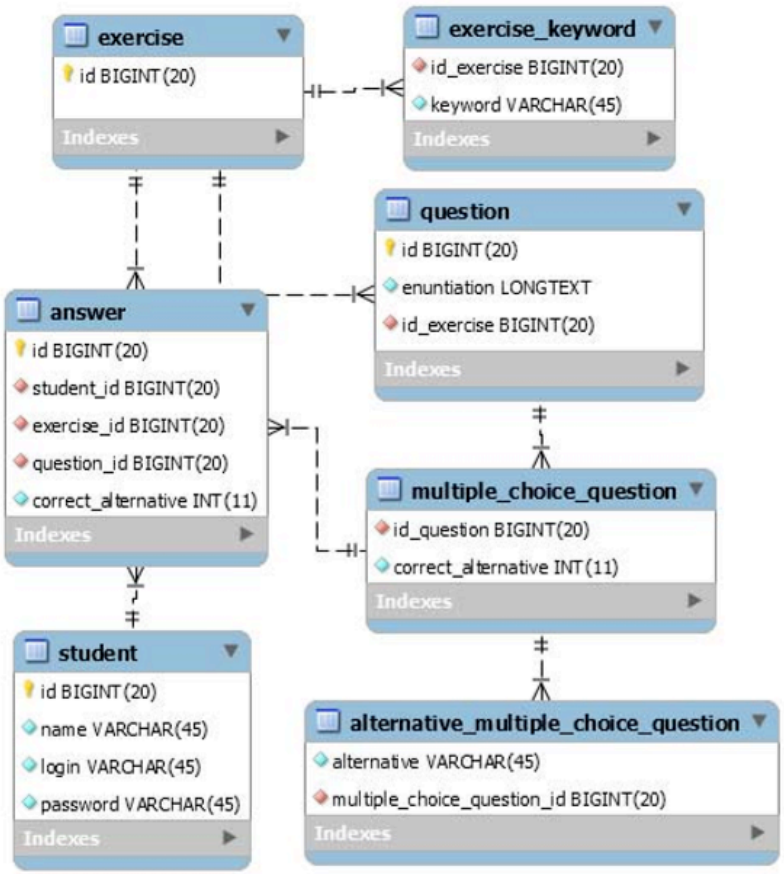
No diagrama, *Student* foi modelado para representar um aluno no sistema; um *Exercise* para representar exercícios (um conjunto de questões); *Answer* para representar a resposta de um aluno em uma questão vinculada a um exercício; *Question* está modelada como uma classe abstrata, a qual não poderá ser instanciada, mas sendo apenas uma classe contém propriedades comuns para os diversos tipos de questões que o sistema; *MultipleChoiceQuestion* representando questão de múltipla escolha que possui as propriedades que foram herdadas de *Question*, além de uma lista de alternativas e uma propriedade para indicar em qual índice da lista se encontra a resposta correta.

As funcionalidades selecionadas para serem implementadas no EducService foram: cadastro de aluno, cadastro de questão de múltipla escolha, cadastro de exercício, pesquisa de aluno por id, pesquisa de exercício por id, pesquisa de questão de múltipla escolha por id.

4.1.1 EducService-OO

Na implementação orientada a objetos do EducService, o diagrama de classes apresentado na Figura 13 foi concretizado além dos requisitos funcionais. Sobre a camada de persistência, esta foi implementada usando o padrão DAO com auxílio da API JDBC para comunicação e persistência no banco de dados MySQL. No banco de dados MySQL foi projetada a persistência dos dados gerados pelo EducService-OO. O Modelo Relacional desenvolvido é apresentado na Figura 14.

Figura 14 - Modelo Relacional do EducService-OO.



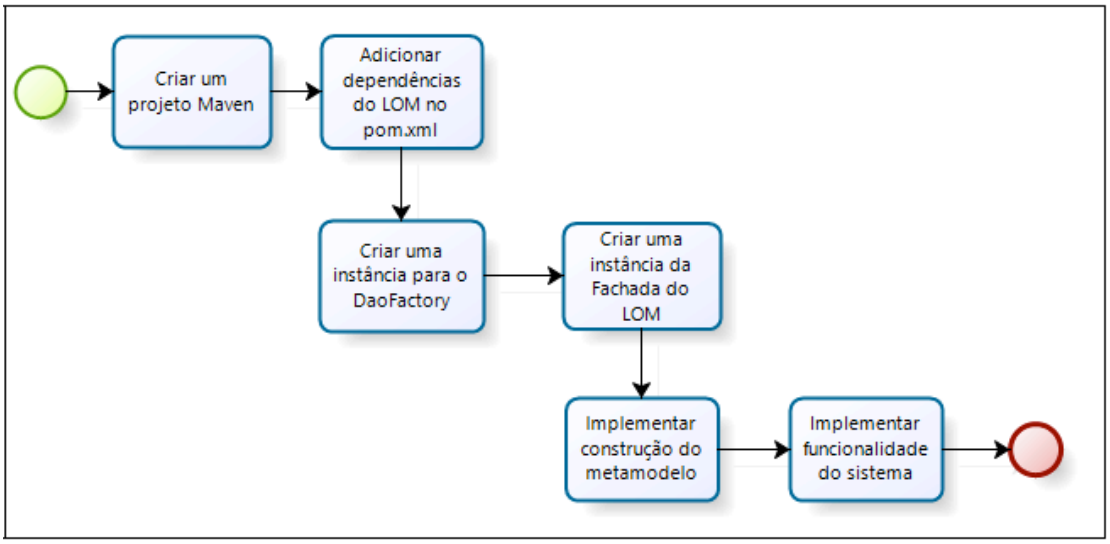
Fonte: o autor (2015)

Na Figura 14 é possível verificar que foi necessário representar o modelo criado através de sete tabelas interligadas entre si para expressarem os relacionamentos existentes entre as entidades e as propriedades. Contudo, se houvesse a necessidade de realizar a persistência de outras entidades, consequentemente haveria a necessidade da criação de novas tabelas e novas colunas.

4.1.2 EducService-LOM

Nas implementações AOM realizadas para o EducService com auxílio do LOM, denominadas por EducService-LOM:MySQL e EducService-LOM:Neo4J, um processo de negócio contendo algumas atividades importantes foi estabelecido.

Figura 15- Processo de negócio para criação do EducService-LOM.



Fonte: o autor (2015)

Na Figura 15, na qual está especificado o processo de negócio utilizado para a criação do EducService-LOM, foram definidas seis etapas, algumas das quais merecem ser detalhadas. Ao criar e configurar um projeto, como no caso do EducService-LOM, um projeto que possibilitou gerenciar as dependências através da ferramenta Apache Maven, é necessário definir em qual banco de dados se pretende realizar a persistência entre as implementações persistentes disponíveis. No entanto, é necessário garantir que haja, em todo sistema, apenas uma instância do *DaoFactory* e que a mesma seja incluída na instância da *BusinessFacade* do LOM, a qual também deve ter apenas uma instância em toda a aplicação.

Figura 16 – Instanciando o DaoFactory e o LOMFacade.

```
public void init() {
    daoFactory = new MySqlDaoFactory();
    daoFactory.createDatabase();

    Facade lomFacade = new BusinessFacade(daoFactory);
    AdaptableEducServiceFacade.createEducServiceSchema(lomFacade);

    this.facade = new AdaptableEducServiceFacade(lomFacade);
}
```

Fonte: o autor (2015)

A Figura 16 apresenta um exemplo das atividades que descrevem a criação de instância da fachada do LOM e do *DaoFactory*. Nesse caso, o *DAOFactory* foi instanciado de modo a utilizar a implementação da camada de persistência para o banco de dados MySQL.

Para que a implementação da camada de persistência fosse substituída, o *DAOFactory* deveria ser instanciado com outro objeto, por exemplo, o Neo4JDAOFactory para o banco de dados Neo4J, implementação também utilizada neste trabalho.

Outro fato importante descrito como uma atividade que foi realizada no processo estabelecido, é a implementação da construção do metamodelo do sistema. Nessa etapa, teve que ser garantida que tal implementação deverá ser a primeira funcionalidade a ser executada quando o software for inicializado, para verificar se o metamodelo está criado ou criá-lo, caso contrário.

Figura 17 - Metamodelo do EducService-LOM:Mysql cadastrado.

id	version	namespace	name
1	3	br.ufpb.educservice	student
2	5	br.ufpb.educservice	exercise
3	8	br.ufpb.educservice	multipleChoiceQuestion
4	4	br.ufpb.educservice	answer

id	name	configuration	entityType_id	type
1	name	{"mandatory" : true}	1	TEXT
2	login	{"mandatory" : true}	1	TEXT
3	password	{"mandatory" : true}	1	PASSWORD
9	enunciation	{"mandatory" : true}	3	LONGTEXT
10	correctAlternative	{"mandatory" : true, "minvalue" : 1}	3	INTEGER

Fonte: o autor (2015)

Através da Figura 17 se pode observar que na implementação do EducService-LOM com persistência no banco de dados MySQL, o que eram as tabelas, colunas e relacionamentos apresentados na Figura 13, nessa através do LOM passaram a ser metadados persistidos em duas tabelas: *entity_type* e *property_type,* respectivamente apresentadas verticalmente na imagem. Da mesma forma, na implementação que realiza a persistência no Neo4J, o metamodelo estaria representado através de nós com o rótulo *ENTITY_TYPE*, os quais estariam relacionados com seus respectivos nós do tipo *PROPERTY_TYPE,* através da relação *IS_A_PROPERTY_TYPE_OF_ENTITY_TYPE.* Nessas abordagens para persistência dos metadados, caso novas entidades ou propriedades necessitassem ser cadastradas, na abordagem relacional seria necessária a inserção de registros nas tabelas entity_type e

property_type, respectivamente. Já na abordagem orientada a grafos, novas entidades ou propriedades também poderiam ser manipuladas diretamente no banco de dados incluindo novos nós do tipo ENTITY_TYPE e PROPERTY_TYPE, respectivamente. Para a criação das funcionalidades, o programador faz manipulações através dos objetos *Entity* e *Property,* principalmente.

Figura 18 - Funcionalidade de cadastro de aluno no EducService-OO.

```
public Student createStudent(String name, String login, String password) {
    Student student = new Student();
    student.setNome(name);
    student.setLogin(login);
    student.setSenha(password);

    return studentDAO.createStudent(student);
}
```

Fonte: o autor (2015)

Figura 19 - Funcionalidade de cadastro de aluno no EducService-LOM.

```
public Student createStudent(String name, String login, String password) {
    Entity studentEntity = new Entity();
    studentEntity.setEntityType(this.studentEntityType);

    List<Property> properties = new LinkedList<Property>();

    Property nomeProperty = newProperty(
            this.namePropertyType, name, studentEntity);
    properties.add(nomeProperty);

    Property loginProperty = newProperty(
            this.loginPropertyType, login, studentEntity);
    properties.add(loginProperty);

    Property senhaProperty = newProperty(
            this.passwordPropertyType, password, studentEntity);
    properties.add(senhaProperty);

    studentEntity.setProperties(properties);
    return this.convertStudent(this.lomFacade.create(studentEntity));
}
```

Fonte: o autor (2015)

A funcionalidade de cadastro de aluno na implementação OO comparada com a implementação LOM pode ser visualizada nas Figura 18 e Figura 19, respectivamente. Na implementação LOM, conforme mencionado anteriormente, é possível identificar que para cadastrar um aluno foi necessário instanciar um objeto do tipo *Entity* (representação da entidade aluno) e três objetos do tipo *Property* (um para representar cada propriedade de

aluno). É perceptível o aumento na complexidade de lidar com os objetos de AOM, o que por outro lado pode ser compensado quando se opta por um estilo arquitetural que é capaz de proporcionar mudanças de requisitos em tempo de execução, sem que o sistema seja parado, alterado, compilado e executado novamente, mas apenas alterações nos metadados que refletirão diretamente no comportamento do sistema.

5. AVALIAÇÃO DE DESEMPENHO

Esta avaliação de desempenho tem como intuito verificar e comparar o desempenho de três diferentes versões de um sistema, uma quando desenvolvido em uma versão totalmente orientada a objetos com persistência em um banco de dados relacional, denominada EducService-OO, uma quando desenvolvido através do LOM com persistência em um banco de dados relacional, denominada EducService-LOM:MySQL e uma quando desenvolvido através do LOM com persistência em um banco de dados orientado a grafo, conhecida por EducService-LOM:Neo4J.

5.1 HIPÓTESES

Segundo Yoder e Foote (1998), quanto mais se proporciona dinamicidade a um sistema, pior pode ser o seu desempenho. Entende-se com isso que um sistema pode se apresentar mais lento quando desenvolvido através dos padrões AOM do que em sua implementação totalmente orientada a objetos, já que mudar um AOM parece ser bem mais fácil que alterar um OO convencional por não demandar mudanças no esquema do banco de dados. Isso pode acontecer em tempo de execução sem que o sistema seja parado, alterado, recompilado e executado novamente, graças ao uso dos metadados. Para este trabalho considerou-se relevante avaliar o impacto de desempenho ao usar AOM no desenvolvimento de uma parte do escopo do sistema EducService, explorando a hipótese nula de que não há diferença significativa no desempenho da versão não AOM comparada com a versão AOM.

Matsumoto e Guerra (2012) afirmaram que, devido aos sistemas baseados em AOM necessitarem realizar a persistência do seu modelo através de metadados, considerando a natureza evolutiva desses modelos, bancos de dados relacionais talvez não seja a melhor alternativa para persistência de tais sistemas. Entretanto, como uma alternativa de implementação de banco de dados NOSQL, sabendo que o Neo4J, uma implementação do banco de dados NOSQL orientado a grafos é uma alternativa adequada quando se tem um contexto onde dados do modelo se encontram altamente conectados, no caso das implementações AOM, considerou-se a hipótese de que essa implementação teria melhor desempenho que a implementação LOM que usa MySQL. Nesse contexto, foi proposta a hipótese nula (H0) de que não há diferença significativa de desempenho entre a implementação LOM com MySQL e a que usa Neo4J e utilizou-se o T-Test para refutá-la.

Sendo assim, foram consideradas duas hipóteses nulas (H_{0A} e H_{0B}) e duas possibilidades de hipóteses alternativas para cada:

- **H_{0A}**: Não haverá diferença significativa de desempenho entre o EducService-OO e o EducService-LOM:MySQL.
- **H_{1A}**: O desempenho do EducService-OO será melhor que o do EducService-LOM:MySQL.
- **H_{2A}**: O desempenho do EducService-OO será pior que o do EducService-LOM:MySQL.
- **H_{0B}**: O desempenho do EducService-LOM:MySQL não apresentará diferença significativa quando comparado com desempenho do EducService-LOM:Neo4J.
- **H_{1B}**: O desempenho do EducService-LOM:MySQL será melhor que o desempenho do EducService-LOM:Neo4J.
- **H_{2B}**: O desempenho do EducService-LOM:MySQL será pior que o desempenho do EducService-LOM:Neo4J.

5.2 VARIÁVEIS

Conforme demonstrado anteriormente, o sistema EducService foi implementado em três diferentes versões, portanto como variável independente foi considerada a **versão do EducService** a qual será variada durante a execução dos cenários para a avaliação de desempenho em: EducService-OO, EducService-LOM:MySQL e EducService-LOM:Neo4J. Já como variável dependente, aquela que terá o valor influenciado quando a variável dependente for alterado, foi considerado o **desempenho**, obtido através da métrica tempo de execução (em milissegundos) de algumas operações exercitadas através de diferentes cenários.

5.3 SUJEITOS E OBJETOS

Durante a avaliação de desempenho o sujeito a ser considerado foi Fernando Mateus (autor deste trabalho de conclusão de curso), responsável por projetar, implementar e executar os cenários para coleta dos tempos de execução dos cenários. Os objetos foram os cenários definidos a serem executados sobre o sistema EducService em suas três diferentes versões, apresentadas anteriormente.

5.4 CENÁRIOS

A avaliação de desempenho foi composta por seis cenários, executados por todas as versões do sistema, para que os tempos pudessem ser coletados e posteriormente pudessem ser aplicados cálculos da estatística sobre esses dados para obter algumas conclusões. O valor padrão para a quantidade de execuções foi definido inicialmente como um valor considerado alto o suficiente para observar uma degradação de desempenho em todos os bancos de dados à medida que a quantidade de registros aumentasse. Devido à limitação de tempo para mais execuções dos cenários definidos a seguir, não se pôde posteriormente aumentar o valor amostral de 10.000 registros que foi inicialmente considerado.

Três cenários foram especificados para avaliação de desempenho através de operações de escrita e outros três foram especificados para que se pudesse fazer operações de leitura dos dados:

- Cenário #1: Através do EducService, cadastrar 10.000 alunos. Ao cadastrar cada aluno, coletar em milissegundos o tempo gasto na operação;
- Cenário #2: Através do EducService, cadastrar 1 exercício e vinculado a esse exercício, cadastrar 10.000 questões de múltipla escolha sendo coletado em milissegundos o tempo para cada cadastro de questão de múltipla escolha;
- Cenário #3: Através do EducService, cadastrar 10.000 exercícios, para cada exercício cadastrado cadastrar 10 questões de múltipla escolha. O tempo em milissegundos deverá ser coletado diante do cadastro cada exercício;
- Cenário #4: Através do EducService, após 10.000 alunos cadastrados, pesquisar do primeiro ao o último aluno, sendo assim coletado o tempo em milissegundos da pesquisa de cada um dos 10.000 alunos;
- Cenário #5: Através do EducService, após o cadastro de 10.000 questões de múltipla escolha vinculadas a um exercício, pesquisar da primeira até a última questão de múltipla escolha vinculada ao exercício. Totalizando a coleta de 10.000 tempos de pesquisa, em milissegundos, de questão de múltipla escolha;
- Cenário #6: Através do EducService, após o cadastro de 10.000 exercícios, cada um contendo 10 questões, coletar em milissegundos o tempo de pesquisa da primeira a décima questão de cada um dos 10.000 exercícios. Para este cenário realizou-se, portanto, por questão de reuso das execuções da inserção

de dados, a coleta de 100.000 tempos de pesquisa de questão já que essa foi a quantidade de registros de questões cadastradas quando os 10.000 exercícios foram inseridos .

5.5 AMBIENTE DE EXECUÇÃO

Para a execução dos cenários especificados, um ambiente foi montado de forma que a execução dos seis cenários fosse controlada ao máximo visando eliminar possíveis vieses como, por exemplo, a execução de processos desnecessários no sistema operacional ou até interrupções devido a requisições de rede.

Portanto, foi utilizado um computador com as seguintes características de hardware:

- Processador dual core com 2.1GHz;
- Memória RAM com 4GB DDR3 1333MHz (2x2GB);
- Disco Rígido SATA de 640 GB (5400RPM).

E como características de software as seguintes:

- Sistema operacional Ubuntu 14.04.2 LTS x86;
- Java version "1.7.0_80" Java(TM) SE Runtime Environment (build 1.7.0_80-b15) Java HotSpot(TM) Server VM (build 24.80-b11, mixed mode);
- MySql Server v.14.04 Distribuição 5.5.43, for debian/gnu (x86).

Como requisito de software, o servidor de banco de dados do Neo4J não se fez necessário ser instalado, já que uma versão *embedded* foi utilizada na versão EducService-LOM:Neo4J para realizar a persistência.

5.6 APRESENTAÇÃO DOS DADOS

Após a execução dos cenários nas três versões do EducService, o que tornou possível a coleta dos dados, a próxima fase da avaliação de desempenho foi realizar a análise. Para essa etapa foram definidos os métodos utilizados para possibilitar possíveis comparações entre as versões avaliadas. Cálculos da estatística como média, desvio padrão, intervalo de confiança de 95% e a razão foram utilizados e os resultados obtidos foram dispostos em tabelas, as quais podem ser verificadas a seguir nas Tabelas 1, 2, 3 e 4.

Através do cálculo da média, foi possível determinar o comportamento de cada cenário em cada versão do EducService. Com o desvio padrão, foi possível perceber qual foi a variação dos tempos brutos acerca da média calculada. O intervalo de confiança de 95% foi

utilizado para calcular a margem de erro diante da amostra coletada, o que indica, em termos percentuais, qual foi a variação da média em decremento ou em suprimento. Já com a razão foi possível apontar quantas vezes uma versão foi mais eficaz que a outra para cada cenário.

Tabela 1- Resultados EducService-OO

EducService-OO				
Cenário	**Média (ms)**	**Desvio Padrão (ms)**	**M. Erro (ms)**	**Erro percentual (%)**
#1	56,49	18,00	0,35	0,62
#2	388,16	81,09	1,59	0,41
#3	4311,21	296,10	5,80	0,13
#4	3,11	1,14	0,02	0,72
#5	3,29	1,20	0,02	0,71
#6	6,05	2,66	0,02	0,27

Fonte: o autor (2015)

Tabela 2- Resultados EducService-LOM:MySQL

EducService-LOM:MySQL				
Cenário	**Média (ms)**	**Desvio Padrão (ms)**	**M. Erro (ms)**	**Erro percentual (%)**
#1	417,52	50,27	0,99	0,24
#2	1145,11	81,59	1,60	0,14
#3	12188,74	233,32	4,57	0,04
#4	37,22	8,30	0,16	0,44
#5	54,97	13,83	0,27	0,49
#6	81,79	21,58	0,13	0,16

Fonte: o autor (2015)

Tabela 3- Resultados EducService-LOM:Neo4J

EducService-LOM:Neo4J				
Cenário	**Média (ms)**	**Desvio Padrão (ms)**	**M. Erro (ms)**	**Erro percentual (%)**
#1	504,82	255,77	5,01	0,99
#2	865,46	508,28	9,96	1,15
#3	14569,68	5804,48	113,77	0,78
#4	435,74	101,15	1,90	0,44
#5	445,77	121,19	2,27	0,51
#6	450,14	128,67	0,76	0,17

Fonte: o autor (2015)

Com os valores absolutos coletados e os cálculos da média, desvio padrão e intervalo de confiança de 95% realizados, o próximo passo foi calcular a razão entre as versões a serem comparadas que apontaram resultados descritos na Tabela 4.

Tabela 4- Razão entre as três versões do EducService

Razão		
Cenário	**ES-LOM:My/ES-OO**	**ES-LOM:N4J/ES-LOM:My**
#1	7,39	1,21
#2	2,95	0,76
#3	2,83	1,20
#4	11,96	11,71
#5	16,72	8,11
#6	13,52	5,50
Legenda: ES-OO (EducService-OO); ES-LOM:My (EducService-LOM:MySQL); ES-LOM:N4J (EducService-LOM:Neo4J).		

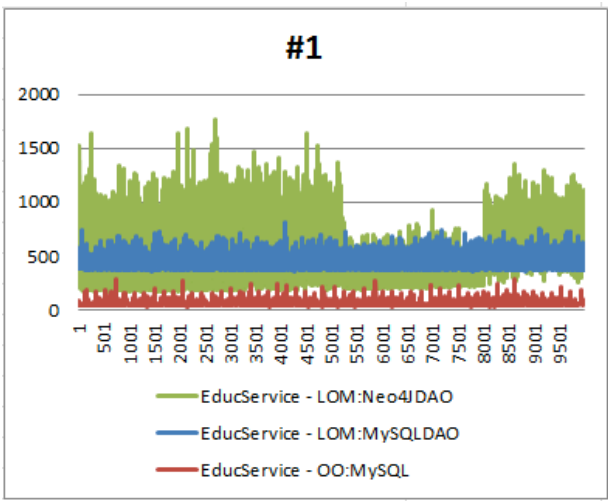
Fonte: o autor (2015)

Com intuito de verificar se havia diferença significante entre as médias obtidas foi realizado um teste *t-Student*, através do qual as versões foram comparadas de duas em duas, sendo a primeira comparação realizada entre o EducService-OO e o EducService-LOM:MySQL, e a segunda comparação entre o EducService-LOM:MySQL e o EducService-LOM:Neo4J. Como ferramenta auxiliar foi utilizado o GraphPad[2] Software online. Portanto, foi possível verificar que para todos os cálculos do teste t-Student o valor obtido foi sempre menor que 0.0001, o que implica dizer que por critérios convencionais, as diferenças são consideradas significantes.

Como reflexo dos cálculos estatísticos realizados, foram gerados gráficos a partir dos dados brutos coletados, e que são apresentados nas Figuras 20, 21 e 22. A ideia destes gráficos é proporcionar outra visualização do comportamento de cada abordagem durante a execução dos cenários.

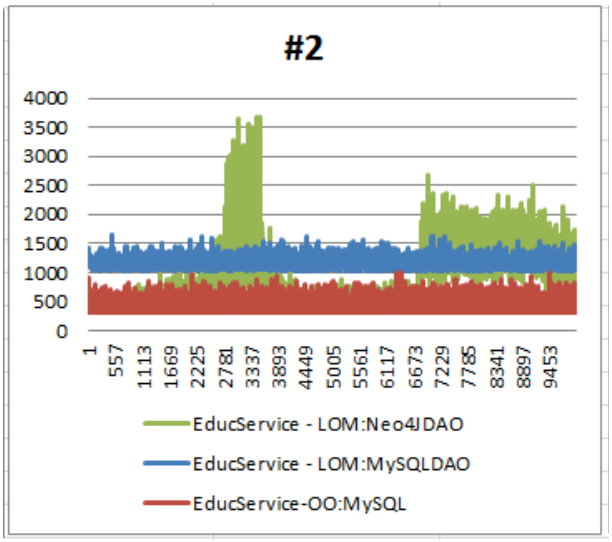
[2] GRAPHPAD. T Test Calculator. Disponível em http://graphpad.com/quickcalcs/ttest1.cfm. Acesso em: 02 dez. 2015..

Figura 20- Gráfico dos dados brutos da operação de escrita #1



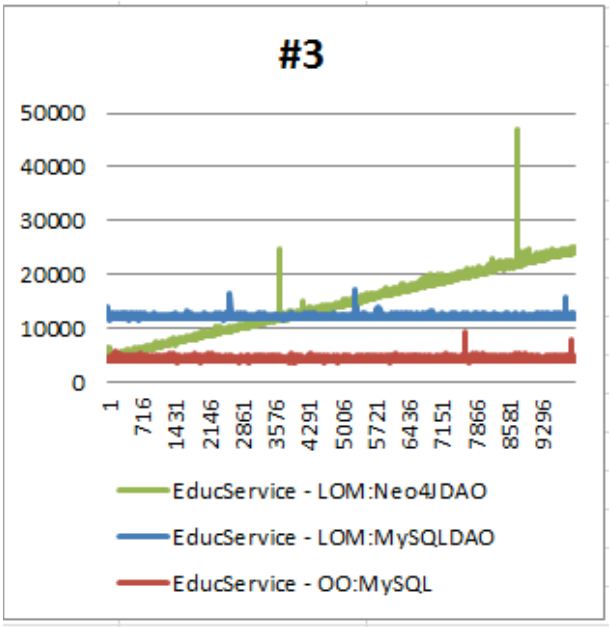
Fonte: o autor (2015)

Figura 21 – Gráfico dos dados brutos da operação de escrita #2



Fonte: o autor (2015)

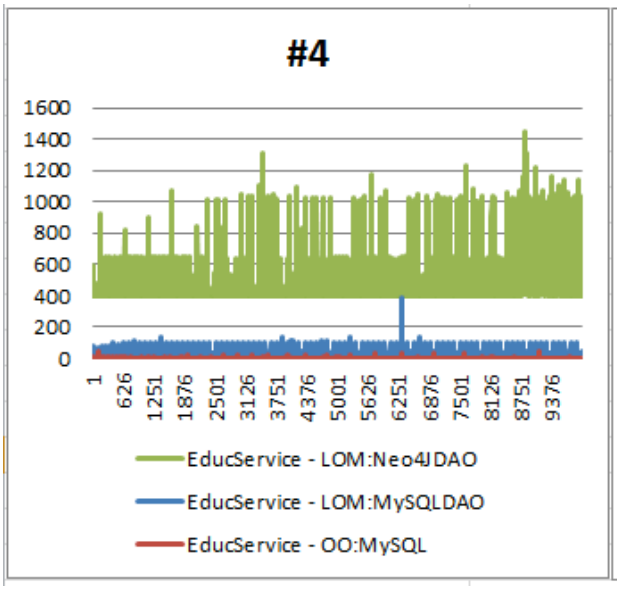
Figura 22 - Gráfico dos dados brutos da operação de escrita #3



Fonte: o autor (2015)

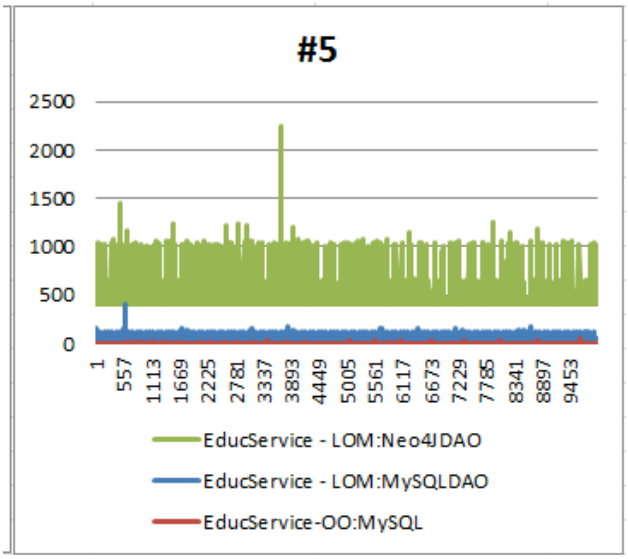
Nas Figuras 20, 21, e 22 que apresentam os gráficos gerados diante dos dados brutos coletados durante a execução dos cenários de escrita, é perceptível uma padronização no comportamento da implementação do EducService orientado a objetos e do EducService baseado no LOM que realizou a persistência no banco de dados MySQL. Para essas implementações não se percebeu degradação linear, exponencial ou quadrática. Não se pode dizer o mesmo, porém, da implementação do EducService que realizou a persistência no banco de dados orientado a grafo, e que no cenário #3 apresentou um aumento linear a medida que a quantidade de registros cresceu.

Figura 23- Gráficos dos dados brutos da operação de leitura #4



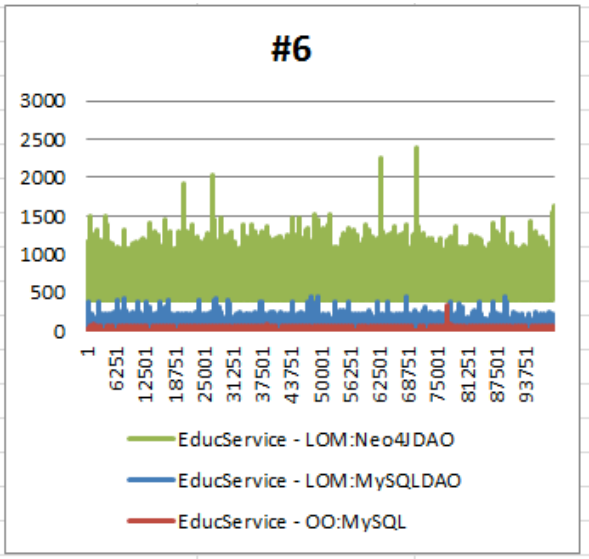
Fonte: o autor (2015)

Figura 24 - Gráficos dos dados brutos da operação de leitura #5



Fonte: o autor (2015)

Figura 25 - Gráficos dos dados brutos da operação de leitura #6



Fonte: o autor (2015)

Em contrapartida, nos cenários de leitura cujos gráficos estão representados nas Figuras 23, 24 e 25, embora a implementação que realizou a persistência no banco de dados Neo4J tenha apresentado pior desempenho em todos os casos, não foi perceptível qualquer aumento identificado em um dos cenários de escrita ao analisar os dados brutos. Desta vez, houve um pouco mais de padronização do comportamento de todas as implementações e se observa claramente que a implementação explorando o Neo4J apresentou os piores resultados nos cenários considerados.

5.7 RESULTADOS

Sobre a primeira comparação realizada entre o EducService-OO e o EducService-LOM:MySQL, considerando o melhor desempenho do EducService-OO tanto os cenários de escrita (#1, #2 e #3) como os cenários de consulta de dados (#4, #5 e #6) e através da diferença significativamente considerável, apontados para todos os casos, obtida através do *test t-Student*, é possível afirmar que a hipótese H_{0A} foi rejeitada para todos os cenários, e observou-se a confirmação da hipótese H_{1A}.

Dessa vez, considerando a comparação realizada entre as versões AOM, o EducService-LOM:MySQL e o EducService-LOM:Neo4J, para os cenários #1, #3, #4, #5 e #6, onde o EducService-LOM:MySQL se apresentou mais eficiente apontando também diferença significativamente considerável, negando a hipótese nula H_{0B} e validando a hipótese

alternativa H_{1B} para estes cenários. Nesta mesma comparação entre as versões AOM, apenas no cenário #2 o EducService-LOM:Neo4J foi mais eficiente e através da diferença significativamente considerável apontada, a hipótese alternativa H_{2B} pôde ser validada.

Buscando identificar o motivo da não padronização dos comportamentos observados através dos gráficos, nos casos de escrita na versão que persistiu os metadados no Neo4J se deve considerar que no cenário #1 o registro para a entidade que foi cadastrada (aluno) possuía 4 propriedades simples, do tipo texto. No cenário #2 para a entidade cadastrada (questão de múltipla escolha) esse número de propriedades foi decrementado para 2, mas compensado devido a uma delas ser do tipo lista (palavras-chave), para a qual foram inseridos 5 registros, totalizando 6 registros de propriedades. No cenário #3, o número de propriedades voltou a crescer quando considerado o exercício cadastrado com as suas 10 questões. Nesse caso o comportamento apresentado foi bastante diferenciado dos demais casos, apresentando indícios de que o Neo4J passou a apresentar degradação à medida que o volume de dados aumentava devido ao crescimento linear apresentado. Nas implementações que utilizaram o MySQL os comportamentos não apresentaram muitas variações para esses casos.

Nos casos de leitura, os comportamentos não apresentaram muitas variações, assim como observado nos casos de escrita. Acredita-se que o Neo4J apresente otimizações mais eficazes quando a necessidade é consultar dados. Outro fato que deve ser considerado é que talvez, devido à ausência dos padrões de relacionamento, regras de negócio e interface gráfica do usuário na implementação das versões AOM, talvez o nível de interconexão entre os metadados tenha sido baixo, o que fez o Neo4J não apresentar seu melhor desempenho.

No MySQL, como descrito, o comportamento se manteve padronizado tanto nos casos de escrita como nos casos de leitura de dados. Portanto, como o MySQL é um produto que se encontra no mercado há bastante tempo, sendo ainda o banco de dados mais utilizado de acordo com o ranking DB-Engines (2015), talvez apresente uma série de otimizações que não tenham permitido sua degradação à medida que o volume de dados fosse crescendo, até o volume máximo definido para este trabalho.

Com tais resultados, foi possível verificar que quanto mais se proporciona dinamicidade a um sistema, pior pode ser o desempenho desse sistema (Foote e Yoder, 1998), afirmação base para a elaboração da hipótese H_{1A}, a qual foi validada em todos os cenários executados. Na avaliação realizada observou-se que tal impacto de desempenho em tornar um sistema orientado a objetos em um sistema adaptável baseado em AOM pode variar entre 2,95

e 16,72 vezes ao comparar a versão orientada a objetos com persistência no MySQL com a versão AOM com persistência também no MySQL.

No entanto, em apenas um dos seis cenários descritos foi possível verificar a expectativa de que o uso de bancos de dados relacionais fosse uma escolha ruim para persistência de tais sistemas (Matsumoto e Guerra, 2012), afirmação base para a elaboração da hipótese H_{2B}, pois as implementações que utilizaram banco de dados relacionais na camada de persistência foram as que apresentaram quase sempre melhor desempenho, validando em uma proporção maior a hipótese alternativa H_{2A}, ao menos quando comparadas com a implementação com o Neo4J e considerando o sistema EducService. Com o banco de dados Neo4J para realizar a persistência dos metadados da versão AOM, a razão entre o desempenho da versão Neo4J quanto a MySQL foi inferior a 1 (0,76) em apenas um dos cenários descritos. Nos demais, o tempo de execução dos cenários com Neo4J levaram de 1,20 a 11,71 vezes mais tempo que as execuções dos cenários com MySQL.

6. CONSIDERAÇÕES FINAIS

Sabendo que o estilo arquitetural *Adaptive Object Model* tem como propósito tornar os sistemas mais fáceis de mudar, inclusive em tempo de execução, sendo chamados por isso de sistemas adaptáveis, e considerando que o uso do arcabouço LOM promove reuso na implementação de sistemas AOM, este trabalho se focou inicialmente no desenvolvimento e uso desse arcabouço e na avaliação de desempenho de um sistema desenvolvido através do LOM comparado com sua versão orientada a objetos sem foco em adaptabilidade. Acredita-se que o arcabouço proporciona reuso das implementações dos padrões AOM inclusive da camada de dados, o que é um diferencial de outros encontrados na literatura, e que ele ajuda a promover abstração até certo nível da complexidade para o desenvolvimento de tais sistemas.

Confirmando tal afirmação de Yoder e Foote (1998) que quanto mais se proporciona dinamicidade a um sistema, pior pode ser o seu desempenho, os resultados apresentados na avaliação de desempenho mostraram que as versões AOM apresentaram perda considerável de desempenho em relação à versão orientada a objetos sem adaptabilidade. No entanto, com relação ao uso de abordagem relacional ou não relacional para implementar AOM, foi possível verificar em apenas um dos seis cenários observados que a abordagem relacional não foi a mais adequada para um sistema AOM, como sugerido por Matsumoto e Guerra (2012), pelo menos em comparação à abordagem baseada em grafos com Neo4J e no sistema analisado.

Diante dos dados apresentados pela avaliação de desempenho, viu-se que a abordagem NOSQL através de do banco de dados orientado a grafo Neo4J quase sempre não se apresentou mais eficaz do que a abordagem relacional MySQL. Portanto, é importante destacar que tal conclusão considera o contexto da implementação dos padrões *Type Object*, *Property* e *Type Square* através do sistema EducService implementado na sua versão AOM com auxílio do LOM.

Como trabalho futuro, está prevista uma nova avaliação de desempenho, desta vez considerando um maior número de registros, buscando identificar possíveis degradações dos bancos de dados à medida que a quantidade de registros cresce utilizando uma análise estatística mais detalhada.

Outra importante variação na avaliação de desempenho seria considerar um sistema com maior número de entidades e relacionamentos, além de novos cenários de consulta de dados como pesquisas envolvendo "*joins*". Ainda seria interessante avaliar outras

implementações de bancos de dados NOSQL, como, por exemplo, bancos de dados orientados a documentos, como o MongoDB, como foi o caso do trabalho desenvolvido por Matsumoto e Guerra (2012).

REFERÊNCIAS BIBLIOGRÁFICAS

BUSCHMAN, F.; HENNEY, K.; SCHMIDT. **On Patterns and Pattern Language: Pattern-Oriented Software Architecture.** vol. 5. Inglaterra: John Wiley & Sons, 2007. cap. 16: Referenced Patterns, p.402.

BUSCHMAN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. **A System of Patterns: Pattern-Oriented Software Architecture.** vol. 1. Inglaterra: John Wiley & Sons, 1996. cap. 2: Architectural Patterns, p.169-170.

CROCKFORD, D. **The application/json Media Type for JavaScript Object Notation (JSON)**. 2006. Disponível em: <https://www.ietf.org/rfc/rfc4627.txt>. Acesso em 12 jul. 2015.

DANTAS, A.; BORBA, P.; YODER, J.; JOHNSON, R. **Using Aspects to Make Adaptive Object-Models Adaptable.** In: RAM-SE'04. ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution, 2004, Oslo, Noruega.

EASTERBROOK, S.; SINGER, J.; STOREY, M. A.; DAMIAN, D. Selecting Empirical Methods for Software Engineering Research. 2008. In: Journal Empirical Software Engineering, Hingham, Massachusetts, Estados Unidos.

FERREIRA, H. S. **Adaptive object-modeling: Patterns, tools and applications.** 2010. 222f. Tese (Doutorado) - University of Porto, Faculty of Engineering, Porto, Portugal.

FERREIRA, H. S.; CORREIA, F. F.; WELICKI, L.; **Patterns for data and metadata evolution in adaptive object-models.** In: Conference on Pattern Languages of Programs, 15th, 2008, Nashville, Tennessee.

GIL, A. C.; **COMO ELABORAR PROJETOS DE PESQUISA. .** 4 ed. São Paulo: Atlas, 2002.

GUERRA, E. **Design Patterns com Java: Projeto orientado a objetos guiado por padrões.** Brasil: Casa do Código, 2012, cap. 2: Reuso através de Herança, p. 40.

HECTH, R., JABLONSKY, STEFAN. **NoSQL Evaluation: A Use Case Oriented Survey.** In: International Conference on Cloud and Service Computing, 2011, Hong Kong.

HEUSER, C. A. **Projeto de Banco de Dados.** 4 ed. Rio Grande do Sul: Sagra Luzzato, 1998.

JOHNSON, M. R.; WOLF, B. **The Type Object Pattern.** In: Pattern Languages of Program Design 3, 1998, p.47-65, Boston, MA, USA.

KNOWLEDGE BASE OF RELATIONAL AND NOSQL DATABASE MANAGEMENT SYSTEMS. **DB-Engines Ranking.** Disponível em: <http://db-engines.com/en/ranking>. Acesso em: 23 set. 2015.

LEAVITT, N. **Will NoSQL Databases Live Up to Their Promisse?** In: IEEE Computer, 2010, vol. 43, issue. 2, pp. 12–14.

MATSUMOTO, P; GUERRA, E. **An Architectural Model for Adapting Domain-Specific AOM Applications.** In: Proceedings of the SBCars, 6th, 2012, Natal, Rio Grande do Norte.

MILLER, J. J. **Graph Database Applications and Concepts with Neo4j.** In: Proceeding of the Southern Association for Information Systems Conference, 2013, Atlanta, GA, USA.

NEO4J. **Get Started**. Disponível em: < http://neo4j.com/developer/get-started/>. Acesso em: 02 dez. 2015.

OLIVEIRA, F. M.; DANTAS, A.; VILAR, R. **Evaluating the Performance of Adaptable Systems based on AOM: A Case Study exploring LOM.** In: Proceedings of the SBCars, 9th, 2015, Belo Horizonte, Minas Gerais.

ROBERTS, D.; JOHNSON, R. **Evolving Frameworks: A Pattern Language for Developing Object-Oriented Framework.** In: Proceedings of the Conference on Pattern Languages and Programming, 3th, 1996.

SHAW, M., GARLAN, D. **Software Architecture: perspectives on an emerging discipline**. 1 ed. Nova Jersey. Prentice-Hall, 1996.

SUN MICROSYSTEMS. **Core J2EE Patterns - Data Access Object.** Disponível em: <http://www.oracle.com/technetwork/java/dataaccessobject-138824.html>. Acesso em: 15 abr. 2015.

WELICKI, L.; YODER, J. W.; WIRFS-BROCK, R.; E JOHNSON, R. E. **Towards a pattern language for adaptive object models.** In: ACM SIGPLAN Object Oriented Programming Systems and Applications Conference, OOPSLA '07, 2007, Montreal, Canadá.

YISHAN LI; MANOHARAN, S.; **A performance comparison of SQL and NoSQL databases.** In: Communications, Computer and Signal Processing (PACRIM), 2013, Victoria, B.C., Canadá.

YODER, J. W.; BALAGUER, F.; JOHNSON, R. **Architecture and Design of Adaptive Object Models.** In: Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA ’01, 2001, Tampa Bay, Flórida, USA.

YODER, J. W.; FOOTE, B.; **Metadata and Active Object Models**. In: Conference on Patterns Languages of Programs, 5th, 1998, Monticello, Illinois.

YODER, J. W.; FOOTE, B.; **The Adaptive Object-Model Architectural Style**. In: Proceeding of The Working IEEE/IFIP Conference on Software Architecture (WICSA3 ’02). World Computer Congress, 2002, Montreal, CA.